

# Ontology based image retrieval system

- **Project aim**

In this project we have designed a system for retrieval of images related to Indian classical dance form *Bharatnatyam*, especially focussing on *hastamudra* (hand gestures). This system provides a systematic way of maintaining the domain knowledge in the form of an OWL ontology with tools for conveniently annotating the contents of the stored images and retrieving the results of both structured and natural language queries.

- **Introduction**

In this document we summarize different aspects of our work including a detailed walkthrough of the different major components of the software (files, functions, input, output), repository structure, instructions to use the different functionalities, sample results and limitations.

- **Overview of implementation and functionalities**

We split this section into two broad concepts: **Ontology, Natural language query processing**.

## 1. Ontology

```
./ontology_editing
├── dance_image_ontology.owl
├── OntoEditModule.py
├── AddImageAnnotationGUI.py
├── Pre_process.py
├── MudraMatch.py
├── OntoQueryModule.py
└── QueryGUI.py
```

`dance_image_ontology.owl`

The domain knowledge related to hastamudra is maintained as an OWL ontology. It is stored in rdfxml format as `dance_image_ontology.owl`. The ontology IRI is: [http://www.semanticweb.org/rounak/ontologies/2022/5/dance\\_image\\_ontology](http://www.semanticweb.org/rounak/ontologies/2022/5/dance_image_ontology). The ontology is developed using Protégé. HermiT 1.4.3.456 reasoner was used to validate the ontology. The different classes with their descriptions and properties are:

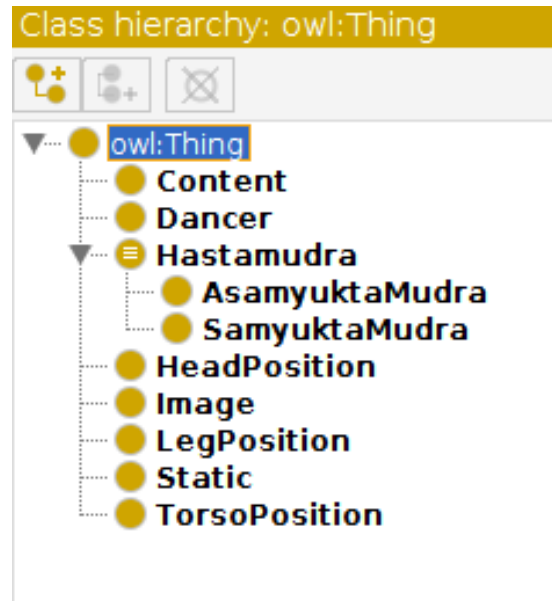


Figure 1(a)

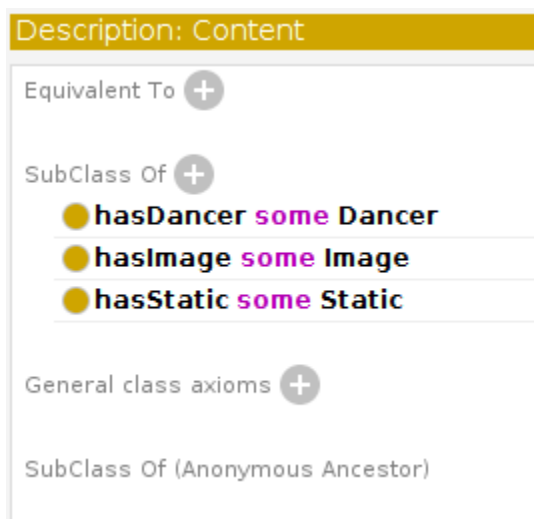


Figure 1(b)

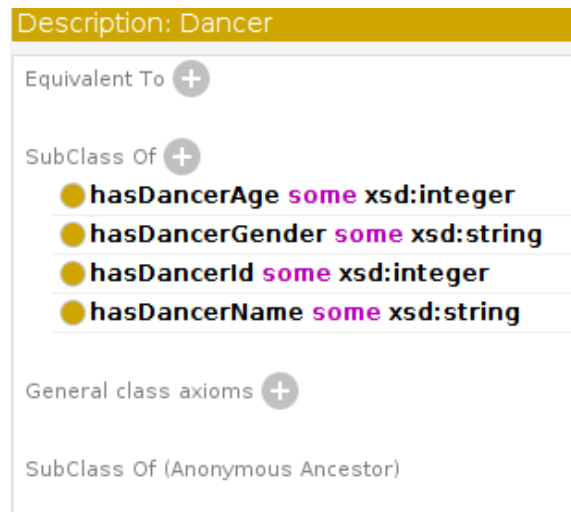


Figure 1(c)

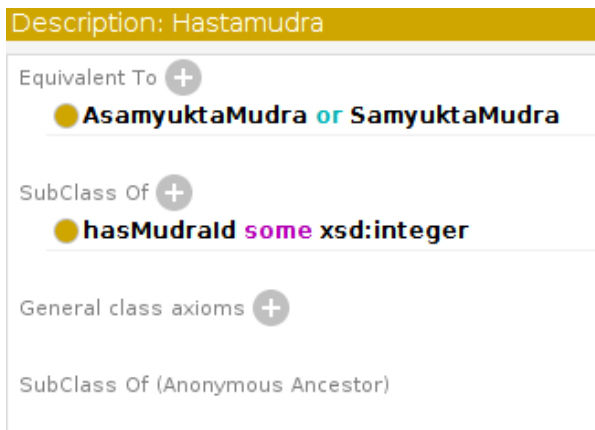


Figure 1(d)

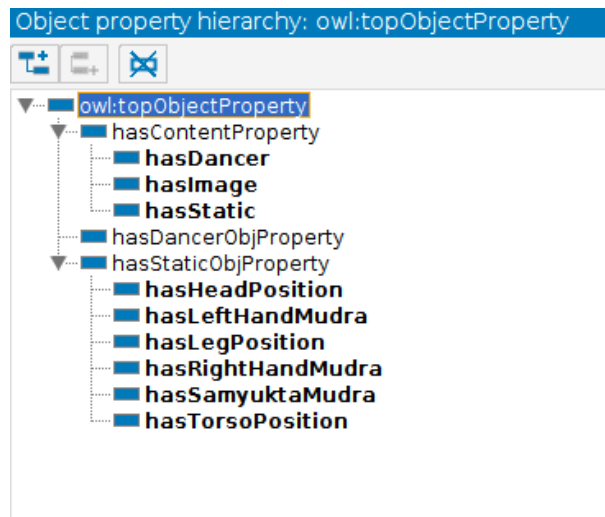


Figure 1(e)

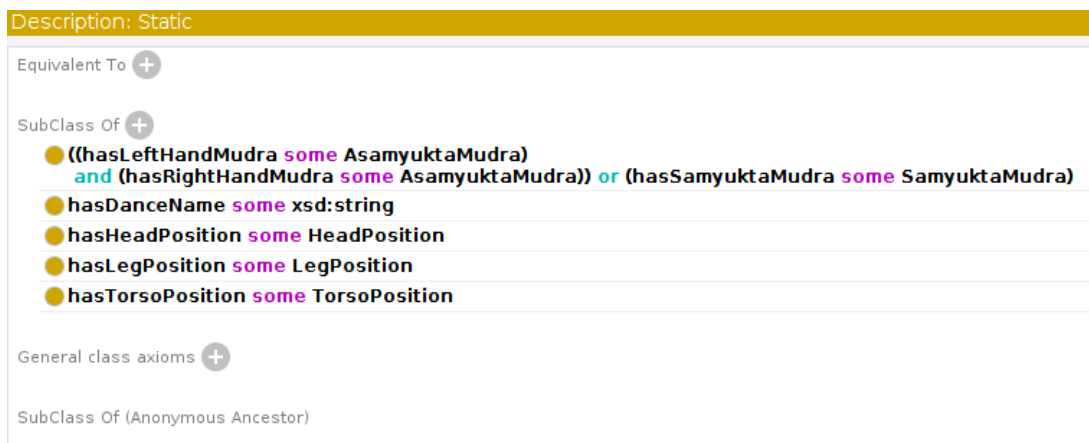


Figure 1(f)

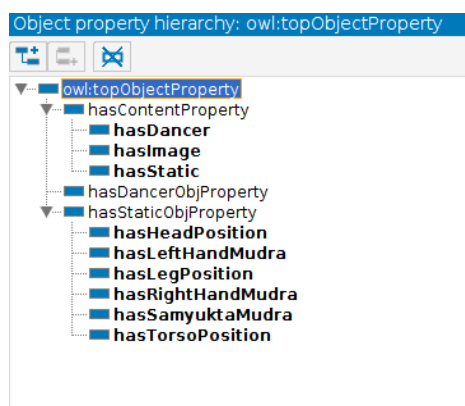


Figure 1(g)

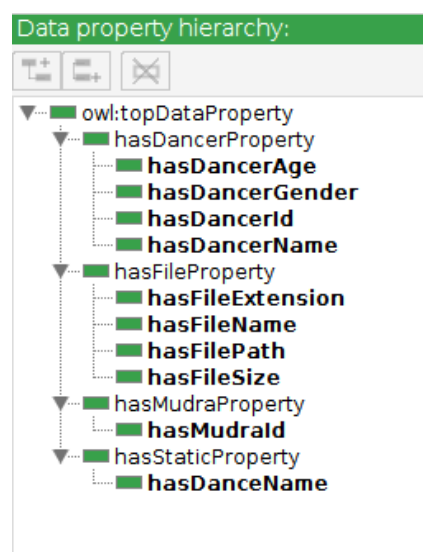


Figure 1(h)

Note that all data properties except hasDancerName and hasDanceName are functional i.e every Dancer/Dance can be related to multiple data members by these properties (i.e. each dancer/dance individual can known and identified by multiple names)

`OntoEditModule.py` (OEM):

Contains functions necessary to load the ontology as Python object, modify and save it. OEM makes use of Owlready2.0 package.

Functions provided by OEM:

```
1. def add_new_dancer_instance(_dancer_name, _dancer_age, _dancer_gender)
    # create an individual of Dancer class in the ontology with properties
```

```
DancerName=[_dancer_name,], DancerAge=_dancer_age, DancerGender=_dancer_gender
    # note that hasDancerName property is not functional which means the same
Dancer individual can have multiple names, and hence is stored as a list
```

```
    # allowing same dancer to have multiple names gives the advantage of identifying
him/her with different names in different images, also ensures that while searching, an
image featuring a particular dancer is retrieved irrespective of what name is used in the
query(of course the name needs to be a valid one)
```

```
2. def update_dancer_info(self, _add_name=None, _update_age=None)
```

```
    # added as a member function of the Dancer class
```

```
    # if _add_name != None, it is appended to the list of names of the dancer individual
```

```
    # if _update_age != None, the age of the individual is modified to the argument
```

```
3. def add_new_image_instance(image_path)
```

```
    # adds new individual of Image class, the data members(full name, image name,
extension, size) are decided from the image_path
```

```
4. def add_new_static_instance(_dance_name, _is_samyukta=False,
    _samyukta=None, _left=None, _right=None)
```

```
    # adds a individual of the static class with DanceName=_dance_name, note that
hasDanceName property is also not functional (same dance can be known by multiple
names)
```

```
    # if _is_samyukta==True, the Static individual is assigned the samyukta mudra with
MudraId=_samyukta
```

```
    # else, it is assigned asamyukta mudras for both left and right hand which have
MudraId _left and _right respectively
```

```
    # note that the ontology has the restriction that a Static individual can have either 1
unambiguous Samyukta mudra OR a pair of 2 Asamyukta mudras for both hands
```

# note that due to lack of domain knowledge we could not account for attributes like HeadPosition, LedPosition, TorsoPosition, though they are there in the ontology and can be extended by adding individuals of these classes. As of now we have created dummy(NULL) individuals of these classes and assigning them whenever new Static individual is created

5. def

```
add_new_content_instance(_image_instance, _dancer_instance, _static_instance)
```

```
# creates the outermost wrapper of an annotation, viz. an individual of the Content class with _image_instance(Image individual), _dancer_instance(Dancer individual), _static_instance(Static individual)
```

OEM also provides a terminal based UI in the form of the function

```
annotate_img_terminalUI()
```

 for image annotation

6. AddImageAnnotationGUI.py:

Provides a GUI for convenient image annotation. Uses Python's Tkinter framework. It contains code to create the widgets for taking different annotations as input and imports the functions from `OntoEditModule`(OEM) to save those annotations in the ontology file.

Run as: Go to ontology\_editing directory and run

```
$ python3 AddImageAnnotationGUI.py
```

## GUI Characteristics

### 1. Dancer Details section

(a) Select Dancer from records:

To choose a dancer from the list of dancers who are already present in the ontology through a dropdown menu. Also possible to enter a new DancerName that will be appended to the list of names of the chosen dancer. (Figure 2(b))

(b) Add new dancer:

Create a new dancer individual (different from any other previously entered dancer in the ontology), with name, age and gender as input. (Figure 2(c))

### 2. Dance Details section:

Enter dance name.

For hastamudra annotation, it is allowed to enter either one Samyukta hastamudra (Figure 2(e)) from the dropdown or two asamyukta mudras for left and right hand respectively(Figure 2(d)).

Figure 2(a)

Figure 2(b)


Figure 2(c)

Figure 2(d)

Figure 2(e)

Dance Image Annotator

Image Path
/home/rounak/Documents/onto/environment/Ontology-editing-and-Query-processing/data/Hand-20220127T140026z
Browse...



Dancer details

Select a dancer from records

Select dancer
ID:1
Names:['Ravi Gupta', 'Rahul Kumar']
Age:25
Genger:M

Add/Update Dancer Name
Kishore Paul

Add new dancer

Dance details

Dance name
Bharatnatyam

Asamyukta Mudra

Select left-hand Mudra
ID:0
alapadma

Select right-hand Mudra
ID:20
katakamukha

Samyukta Mudra

Submit

Figure 2(f): Typical Use Case

#### MudraMatch.py:

The list of valid Hastamudras in *bharatnatyam* is finite, and hence can be listed down exhaustively. The ontology is loaded with 28 distinct Asamyukta mudras and 23 distinct Samyukta mudras. `MudraMatch.py` provides the function which decides, upon taking a free-form piece of text (a word or a phrase), which of the hastamudras in the ontology is closest match for the input. (Example: if input is 'shonkho mudra' hastamudra keywords with best syntactic match would be among 'shirsha', 'shikhara' mudras)

It maintains 2 lists(one for samyukta and another for asamyukta mudra) in the form of a dictionary where the (key,value) pair is of the form: ( (predefined hastamudra name,ID), a list of possible variations / misspellings of the name of the mudra).

```
asam={
    ('pataka',32): ['pataka','pathaka','potaka','potoka','ptka','ptk'],
    ('tripataka',46): ['tripataka','tripathaka','thripathaka','tripotaka',
'triplotoka','trptka','trptk'],
    .
    .
    .
    ('tamrachura',45): ['tamrachura','tamrochura','tamrochuro','tmrchr'],
```

```

('trishula',47): ['trishula','thrishula','thrisula','trishul','trshl'],
}

sam={
    ('anjali',1): ['anjali','anjhali','onjoli','anjoli','anjhl'],
    ('kapota',16): ['kapota','khapota','khaphota','kapotha','kpt'],
    .
    .
    .
    ('khatawa',22): ['khatawa','khathawa','kthtw'],
    ('verunda',49): ['verunda','vrnd'],
}

```

#### 1. `match(keyword)` function:

We go through each of the mudras (once through `asam` and then `sam`), for each mudra we find the similarity<sup>†</sup> of the keyword with each of the entries of the list(of possible variation/misspellings) and take the maximum which we consider as the overall Similarity of keyword with that particular mudra. We return the ID of mudra(s) with maximum overall Similarity. The function returns two lists

```
match_a=[x for overall Similarity(x)==m and x is AsamyuktaMudra],
```

```
match_s= [x for overall Similarity(x)==m and x is SamyuktaMudra]
```

where `m` is the maximum overall similarity over all hastamudras(`asamyukta` and `samyukta` combined).

(<sup>†</sup> measure of similarity: we use Levenstein(/edit) distance to measure the similarity between two words, lower the edit distance higher the similarity).

#### `OntoQueryModule.py` (OQM):

Contains functions for generating SPARQL queries given the requirements of the search.

We define two kinds of queries here:

1. **Tight queries:** In this type of query the requirements of search are very clearly specified. The requirements are in the form of a dictionary which explicitly specifies the different fields like name, age, gender of dancer, dance name and hastamudra (If any of these fields are not necessary conditions for filtering, that is assigned `None`).

A typical requirements dictionary is like:

```

requirements={
    'dancer_name': 'rahul' ,
    'dancer_id': None ,
    'dancer_age': None ,
    'dancer_gender': 'M' ,

```



```

    'dance_name': None ,
    'sam_mudra': None ,
    'l_mudra': 4 , # mudra id
    'r_mudra': None
}

```

(all images satisfying the non-None conditions should be retrieved by the query, note that any dictionary which mentions contradictory requirements like mentioning both sam\_mudra and l\_mudra(or/and r\_mudra) is considered invalid)

2. **Open queries:** In this type of query, the requirements are in the form of a natural language text. Due to the difficulty in accurately being able to determine the requirements from free-form text, there is a relaxation provided while generating the SPARQL query. Therefore the task is to determine which keyword mentions which requirement (in terms of dancer details like gender, hastamudra etc) as much accurately as possible. This comes under Named Entity Recognition which is essentially a NLP problem.

Example: Consider the search string-

```

'Show me a picture of male performing ordhapataka on left hand
and alapada on right hand with vira rasa'

```

We should be able to detect the keyword 'male' as gender specifying keyword and 'ordhapataka', 'alapada' as hastamudra keywords (Note that there may be random spelling mistakes in the search string, there is no hastamudra named 'ordhapataka' or 'alapada' in the ontology, so after detecting that these keywords represent hastamudra, we should be able to determine which valid hastamudra(s) is the most likely match).

Functions provided by OQM:

1. `def generate_tight_query(requirements)`  
# takes a requirements dictionary as input and generates a SPARQL query string
2. `def generate_requirements(free_form_text)`  
# recognizes named entities from natural language text, finds closest match of hastamudra keywords recognized in the text with list of hastamudras defined in the ontology and returns the requirements in the form of `DancerName`, `DancerGender`, `AsamMudList`(list of closest asamyukta mudras), `SamMudList`(list of closest samyukta mudras)  
# uses NER pipeline provided by the Python library `spaCy` and trained with natural language queries related to hastamudra for named entity recognition (more details in next

section: Processing natural language queries) and `MudraMatch.py` (MM) for finding closest match

3. `def generate_open_query(free_form_text)`

# calls `generate_requirements(free_form_text)` for determining the requirements, then constructs the SPARQL query

# note that presence of both samyukta mudra and asamyukta mudra in the requirements won't be considered contractionary as it would have been for a tight query. Here the hastamudra requirements are conjuncted by OR (| |) i.e any image which contains any of the detected mudras (samyukta or asamyukta mudra) in any of the hands will be retrieved. This relaxation has been done considering the limited accuracy of our NER model (due to limited quantity and variety of dataset).

# two different SPARQL queries are generated: one filtering by the OR-ed list of closest samyukta mudra and the other of asamyukta mudra.

`QueryGUI.py`:

Provides a GUI for convenient searching. Uses Python's Tkinter framework. It contains code to create the widgets for taking input and imports the functions from `OntoQueryModule(OQM)` for generating SPARQL queries.

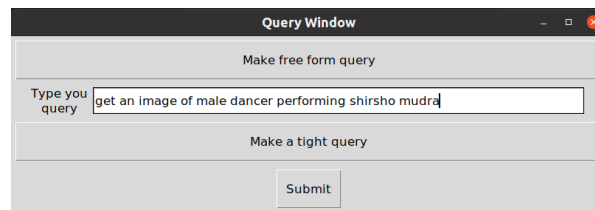
Allows making a tight query (fill in the entries for generating the requirements dictionary) Or an open query by a single piece of text

Run as: Go to `ontology_editing` directory and run  
\$ `python3 QueryGUI.py`

When the user makes an open query, (s)he is given the option to manually annotate the text he entered for the query. It has nothing to do with how the results of this particular query are executed. But a user-entered search text is a potentially useful data sample and if (s)he chooses to help up by annotating it, the search text along with the annotations are safely stored within a binary file called `unlearnnt_bin` in the NLP directory. Samples present in this file represent data that is available but hasn't yet been used to train our NER model. The count of such samples is stored in a hidden file `.unlearnnt_samples_cnt.txt` within the same directory. NLP directory also contains an executable python script `train_NLP_model.py` which can be used to train the model with these unlearnnt samples followed by which `unlearnnt_bin` is cleaned and unlearnnt samples count is reduced to 0.

Example use cases:

## 1. Performing open query



The 'Query Window' interface features a title bar with standard window controls. Below the title bar, there are two buttons: 'Make free form query' and 'Make a tight query'. A text input field labeled 'Type you query' contains the text 'get an image of male dancer performing shirsho mudra'. At the bottom of the window is a 'Submit' button.

Figure 2(a): Search string entered

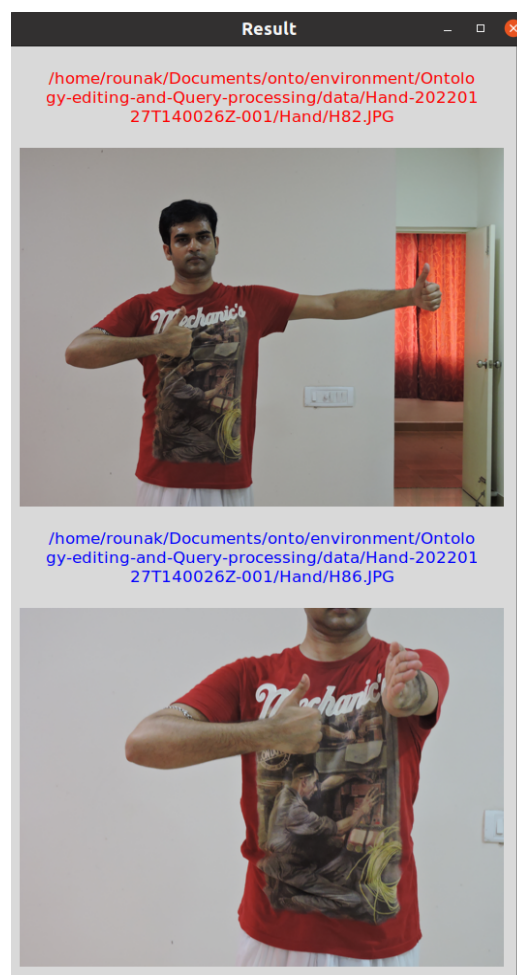


Figure 2(b): Results window

Terminal output:

```
$ python3 QueryGUI.py
* Owlready2 * Warning: optimized Cython parser module
'owlready2_optimized' is not available, defaulting to slower Python
implementation
Performing an open query ...
Asamyukta mudra specific SPARQL query:
PREFIX dio:
<http://www.semanticweb.org/rounak/ontologies/2022/5/dance_image_ontology#>
SELECT DISTINCT ?image_path
WHERE{
    ?content a dio:Content ;
        dio:hasDancer ?dancer ;
        dio:hasStatic ?static ;
        dio:hasImage ?image .
    ?dancer dio:hasDancerName ?dancer_name ;
        dio:hasDancerAge ?dancer_age ;
        dio:hasDancerId ?dancer_id ;
        dio:hasDancerGender ?dancer_gender .
    ?static dio:hasDanceName ?dance_name .
        FILTER regex(?dancer_gender,'M','i')
    ?static dio:hasLeftHandMudra/dio:hasMudraId ?l_mudra .
    ?static dio:hasRightHandMudra/dio:hasMudraId ?r_mudra .
        FILTER ( ?l_mudra=39 || ?r_mudra=39 )
    ?image dio:hasFilePath ?image_path
}
Results: ['H82.JPG', 'H86.JPG', 'H93.JPG']
Samyukta mudra specific SPARQL query:
PREFIX dio:
<http://www.semanticweb.org/rounak/ontologies/2022/5/dance_image_ontology#>
SELECT DISTINCT ?image_path
WHERE{
    ?content a dio:Content ;
        dio:hasDancer ?dancer ;
        dio:hasStatic ?static ;
        dio:hasImage ?image .
    ?dancer dio:hasDancerName ?dancer_name ;
```

```

    dio:hasDancerAge ?dancer_age ;
    dio:hasDancerId ?dancer_id ;
    dio:hasDancerGender ?dancer_gender .
?static dio:hasDanceName ?dance_name .
    FILTER regex(?dancer_gender,'M','i')
    ?static dio:hasSamyuktaMudra/dio:hasMudraId ?sam_mudra .
    FILTER ( ?sam_mudra=37 )
    ?image dio:hasFilePath ?image_path
}
Results: []
Will you help us by annotating your free-form query, this will help us
give better results next time?(y/n): y
Your query: get an image of male dancer performing shirsho mudra
Annotate the keywords:
- 0 if NA
- 1 if PERSON
- 2 if GENDER
- 3 if MUDRA
- 4 if FAC
+ (0, 3, 'get'): 0
+ (7, 12, 'image'): 0
+ (16, 20, 'male'): 2
+ (21, 27, 'dancer'): 0
+ (28, 38, 'performing'): 0
+ (39, 46, 'shirsho'): 3
+ (47, 52, 'mudra'): 0
Adding data sample: ('get an image of male dancer performing shirsho
mudra', {'entities': [(16, 20, 'GENDER'), (39, 46, 'MUDRA')]}))

```

(Note that in (0,3,'get'), 0 and 3 are start and end+1 indices of the word 'get' in the query string and similarly for other keywords)

## 2. Performing tight query

Query Window

Make free form query

Make a tight query

Dancer Name

Dancer ID

Dancer Age

Gender

Dance name

Asamyukta Mudra

Select left-hand Mudra

Select right-hand Mudra

Samyukta Mudra

Figure 3(a): Requirements entered

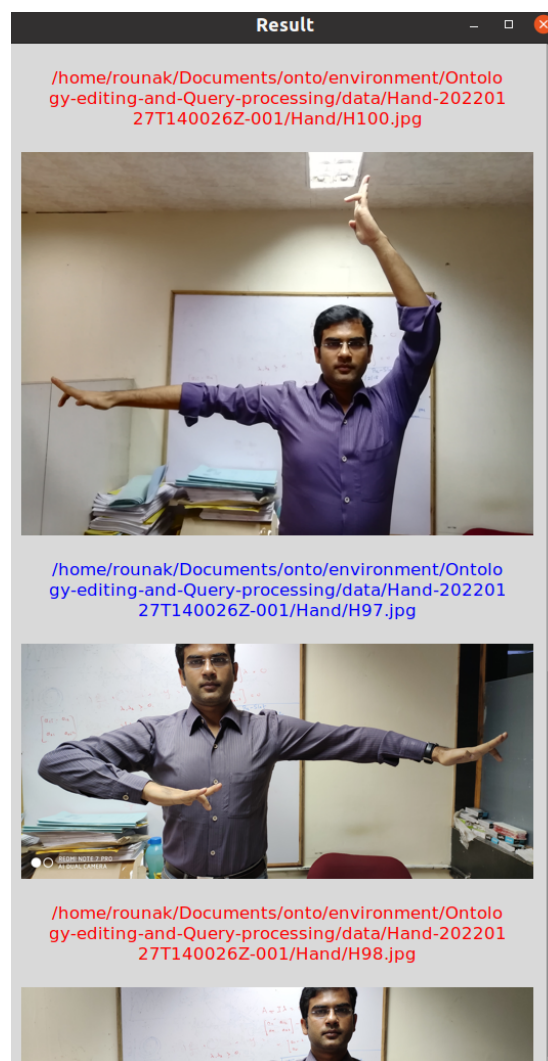


Figure 3(b): Results window

Terminal output:

```
$ python3 QueryGUI.py
* Owlready2 * Warning: optimized Cython parser module
'owlready2_optimized' is not available, defaulting to slower Python
implementation
Performing a tight query ...
Requirements dict.: {'dancer_name': 'Rahul', 'dancer_id': None,
'dancer_age': None, 'dancer_gender': 'M', 'dance_name': None,
'sam_mudra': None, 'l_mudra': None, 'r_mudra': 4}
SPARQL query:
PREFIX dio:
<http://www.semanticweb.org/rounak/ontologies/2022/5/dance_image_ontol
ogy#>
SELECT DISTINCT ?image_path
WHERE{
    ?content a dio:Content ;
        dio:hasDancer ?dancer ;
        dio:hasStatic ?static ;
        dio:hasImage ?image .
    ?dancer dio:hasDancerName ?dancer_name ;
        dio:hasDancerAge ?dancer_age ;
        dio:hasDancerId ?dancer_id ;
        dio:hasDancerGender ?dancer_gender .
    ?static dio:hasDanceName ?dance_name .
        FILTER regex(?dancer_name,'Rahul','i')
        FILTER regex(?dancer_gender,'M','i')
        ?static dio:hasRightHandMudra/dio:hasMudraId ?r_mudra .
        FILTER (?r_mudra = 4)
    ?image dio:hasFilePath ?image_path
}
Results: ['100.jpg', 'H97.jpg', 'H98.jpg', 'H99.jpg']
```

## 2. Processing natural language query

```
./NLP
├── custom_nlp_for_dance
├── train_NLP_model.py
├── generate_query_dataset.py
├── dataset_bin
└── unlearnnt_bin
```

The `spaCy` library provides features for Named Entity Recognition. However the pre-trained pipelines aren't fully suitable for our problem, since all the bharatnatyam and hastamudra specific terminology are not standard English words in general. However the NER pipeline can be trained for custom purposes with sufficient annotated dataset. Since no such dataset is readily available, we have tried to automate the process of generating sample queries of some specific structures along with annotations.

`custom_nlp_for_dance`:

`spaCy` NLP object trained with data samples generated by `generate_query_dataset.py` saved into memory, no need to train it every time it is required, can be directly loaded from here.

Show me a picture which `man GENDER` performing `mrigashirsha MUDRA` on left hand and `kartari MUDRA` on right hand with `hasya FAC` rasa

Figure 4: A visualization of the results of Named Entity Recognition using custom `spaCy` NER pipeline

`generate_query_dataset.py`:(GQD)

Functions provided by GQD:

1. `def save_dataset(n)`

# Generates a set of n sample queries broadly in the below format:

# calls `active_query()` with probability 0.5 that generates query of the form:

(verbs like 'retrieve'/'get'/'show')-(dancer name/gender specific noun or

pronoun/gender neutral noun)-(action words like

'performing','doing','acting')-(asamyukta mudra performed in one hand/both

hands/samyukta mudra/no mudra information at all)-(keyword mentioning facial expression)

# calls `passive_query()` rest of the time that generates query of the form:

(asamyukta mudra performed in one hand/both hands/samyukta mudra/no mudra information at all)-(action phrase like 'performed by','acted by')-(dancer name/gender



specific noun or pronoun/gender neutral noun)-(keyword mentioning facial expression)

# choses gender specifying keyword with prob 0.75, gender neutral keywords rest of the time

# choses asamyukta mudra with prob 0.5, samyukta mudra rest of the time

# saves the list of generated samples in a binary file 'dataset\_bin'

# generated dataset is of the form:

```
[  
  
("fetch me an image of woman performing katakamukha on left hand and  
trishula on right hand with veera rasa", {"entities": [(21, 26,  
'GENDER'), (38, 49, 'MUDRA'), (67, 75, 'MUDRA'), (95, 100, 'FAC')]}),  
  
("retrieve an image where female performing samdamsa mudra on left  
hand and mushti mudra on right hand with advuta rasa", {"entities":  
[(24, 30, 'GENDER'), (42, 50, 'MUDRA'), (74, 80, 'MUDRA'), (106, 112,  
'FAC')]}),  
  
("samdamsa on left hand demonstrated by pandit ", {"entities": [(0, 8,  
'MUDRA')]}),  
...  
]
```

Run as: Go to NLP directory and run

```
$ python3 generate_query_dataset.py
```

This file is not meant to be run directly, anyway for test purposes running the above command generates 100 new data samples and saves them.

`train_NLP_model.py(train):`

executable python script for training and saving `spaCy` nlp object. Input is train (list of training examples)

command line options:

`-q, --query`

print the number of data samples not yet used for training the model

```
-r, --random_learn sample_cnt
    randomly generate sample_cnt number of data samples through
    generate_query_dataset.py and train the nlp model
-l, --learn_rem_samples
    prepares training data by mixing unlearned data samples with equal
    number of randomly generated data samples(generated using
    generate_query_dataset.py) and trains the model with that, the samples
    in unlearned_samples.py are removed and the counter in
    .unlearned_samples_cnt.txt is made 0
```

Run as: Go to NLP directory and run

```
$ ./train_NLP_model.py -q
```

for count of unlearned samples

```
$ ./train_NLP_model.py -r n
```

train with n randomly generated data samples

```
$ ./train_NLP_model.py -l
```

train with unlearned samples mixed with random samples

## • Some sample results

1.

Free form(open) query	'Dancer performing kartaka mudra on left hand'
Detected hastamudra	'kartari', 'pataka'
Corresponding tight query	{'dancer_name': None, 'dancer_id': None, 'dancer_age': None, 'dancer_gender': None, 'dance_name': None, 'sam_mudra': None, 'l_mudra': 18, 'r_mudra': None} (assuming intended mudra is 'kartari' mudra)
Free form(open) query	'/Hand-20220127T140026Z-001/Hand/H9.JPG',

results	'/Hand-20220127T140026Z-001/Hand/H99.jpg', '/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Black_ground/Capture.PNG', '/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Black_ground/fds.PNG', '/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Colourfull_background/FSD.PNG',
Tight query results (intended)	'/Hand-20220127T140026Z-001/Hand/H9.JPG', '/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Black_ground/Capture.PNG'

## 2.

Free form(open) query	'Show a picture of performer doing shirsho mudra'
Detected hastamudra	'shikhara', 'shankha'
Corresponding tight query	{'dancer_name': None, 'dancer_id': None, 'dancer_age': None, 'dancer_gender': None, 'dance_name': None, 'sam_mudra': None, 'l_mudra': 39, 'r_mudra': None} (assuming intended mudra is 'shikhara' mudra)
Free form(open) query results	'/Hand-20220127T140026Z-001/Hand/H82.JPG', '/Hand-20220127T140026Z-001/Hand/H86.JPG', '/Hand-20220127T140026Z-001/Hand/H93.JPG', '/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Colourfull_background/FSD.PNG'
Tight query results (intended)	'/Hand-20220127T140026Z-001/Hand/H93.JPG', '/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Colourfull_background/FSD.PNG'

## 3.

Free form(open) query	'alamadma mudra demonstrated by female dancer with shringer rasa'
Detected hastamudra	'alapadma'
Corresponding tight query	{'dancer_name': None, 'dancer_id': None, 'dancer_age': None, 'dancer_gender': 'F', 'dance_name': None, 'sam_mudra': None, 'l_mudra': None, 'r_mudra': 0} + {'dancer_name': None, 'dancer_id': None, 'dancer_age': None, 'dancer_gender': 'F', 'dance_name': None, 'sam_mudra': None, 'l_mudra': 0, 'r_mudra': None}

Free form(open) query results	'/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Black_ground/FAEG.PNG'
Tight query results (intended)	'/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Black_ground/FAEG.PNG'

#### 4.

Free form(open) query	'Moyura mudra demonstrated by pandit'
Detected hastamudra	'mayura'
Corresponding tight query	{'dancer_name': None, 'dancer_id': None, 'dancer_age': None, 'dancer_gender': None, 'dance_name': None, 'sam_mudra': None, 'l_mudra': 25, 'r_mudra': None} + {'dancer_name': None, 'dancer_id': None, 'dancer_age': None, 'dancer_gender': None, 'dance_name': None, 'sam_mudra': None, 'l_mudra': None, 'r_mudra': 25}
Free form(open) query results	'/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Black_ground/Captureh.PNG', '/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Black_ground/Capturehhj.PNG'
Tight query results (intended)	'/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Black_ground/Captureh.PNG', '/Singlehanded-20220508T135215Z-001/Singlehanded/HM_Black_ground/Capturehhj.PNG'

#### 5.

Free form(open) query	'boraho mudra performed by male dancer'
Detected hastamudra	'baraha'
Corresponding tight query	{'dancer_name': None, 'dancer_id': None, 'dancer_age': None, 'dancer_gender': 'M', 'dance_name': None, 'sam_mudra': 5, 'l_mudra': None, 'r_mudra': None}
Free form(open) query results	'/Hand-20220127T140026Z-001/Hand/H4.JPG'
Tight query results (intended)	'/Hand-20220127T140026Z-001/Hand/H4.JPG'

## ● Limitations and scopes of extension

1. The ontology requires each image to be annotated manually which can be time consuming and repetitive at times. This process can be made automated by application of computer vision and machine learning.
2. Non-robustness of the Named Entity Recognition model used for processing natural language queries owing to reasons including limitation in quantity and variety of dataset used to train the model.
3. Extension of the entire system including ontology for maintaining different other aspects of *bharatnatyam* like facial expressions, leg and foot postures(*mandala*), body movements(*chari*) etc.
4. Extension of the system to manage more complicated movements and scenarios leading to a possible Ontology-based Video retrieval system.