Natural Gesture Data Modeled in Graph Database (Neo4j), contrasted with RDBMS (PostgreSQL)

Gururaj Mulay and Dhruva Patil
Dept. of Computer Science, Colorado State University

I. Introduction

People communicate with each other by the use of words, gestures and expressions.Current state of the art systems are able to process sounds with a high degree of precision. While gesture analysis is fast growing, the domain of natural gestures can be particularly harder as the gestures are subtle with a lot of variation in the intent for a small set of movements.

To address this domain, the Computer Vision Laboratory at CSU aggregated a collection of naturally occurring gestures, known as Elicited Giant Gallery of Naturally Occurring Gestures (EGG-NOG) [1].While the existing database has some limitations in efficient data retrieval, external scripts need to be written to analyze a sub-domain of data fields. This project aims at developing a graph based database in Neo4j for faster and more efficient data retrieval and storage.

A. A Brief About Dataset Collection

The setup for collection of this dataset consists of pairs of participants in front of a table and a screen but in different rooms conversing with each other. One of the participants has a image of a predefined layout of wooden blocks that he must get constructed from the other participant with allowed communication involving natural gestures and if permitted then voice. The former is referred as the signaler and the latter as the actor. Both the participants were recorded using Microsoft Kinect v2. This dataset is a collection of gestures performed by the signaler.

We have 20 pairs of participants each pair constituting a 'session.' In every session, we had up to 10 trials of videos called as 'blocks' of videos for a signaler and then we swapped the pair and carried another set of up to ten trials for the new signaler. Out of these twenty sessions, the pairs were allowed to talk to each other only in 10 sessions. In all, 360 trials were recorded which constituted to about 8 hours of total gesture data. Some trials took as low as 5 seconds and some took as high as about 13 minutes to complete.

B. Dataset Labeling

The gestures in this dataset were labeled in accordance with a language described in paper [1] wherein the gesture was described in terms of body parts such as *head, body, arms, hands, left arms, right arm, left hand, right hand*, the motion of the body part, and the direction of the motion. For labeling we used a software tool 'EASEL' [7] with a separate track for gesture labeling. The syntax for gesture labeling can be found in the paper. With each gesture there could be one or many associated intents that were labeled in another track called intent track.

Figure 1 shows a table that depicts the features associated with the gesture present in the second column of the table. We have labeled 24502 instances of the gestures corresponding to the rows in the table, occurring in 360 trials performed by 40 participants. The first column is the unique *GestureID* assigned to every gesture represented in second column called *Label*. Third column *Project* tells if the participant pair was allowed to use sound during communication. Fourth column is *Session Name* with information about session number, participant number, and the block (video) number for that gesture and fifth column is the location of that video file. Next columns contain the timing information for the gesture including start and end timestamp for the gesture in terms of 100 ns ticks. Columns *Start Frame Timestamp* and *End Frame Timestamp* tells the frame numbers over which the gesture occurred.

Apart from the gesture dataset, we also recorded the depth and skeleton information for the signaler. A skeleton file contains the information about the human joints tracked by the Kinect sensor on per frame basis.

II. Gesture Database as RDBMS in PostgreSQL

As the first step, we modeled the dataset in a relational database in PostgreSQL - an open source relational database system [2]. We used a python package 'psycopg2' to import the CSV files of the tables in the dataset into the PostgreSQL Database. The scripts used for the import can be found in the zip folder submitted along with the report.

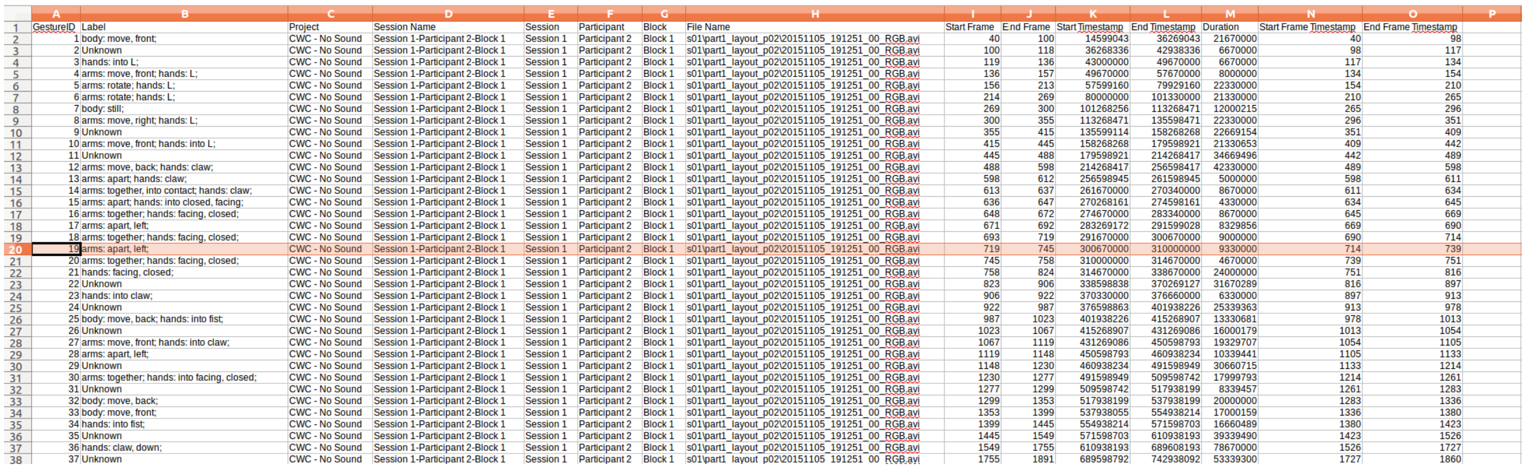
A. Structure

First, we created a database in PostgreSQL named as cwcdb (Communicating with Computer database). We structured the database mainly into 5 tables viz., gesture_table, session_table, participant_count_no_sound, participant_count_with_sound, and main_table. Figure 2 and 3 show the schema for some of these tables.

The *gesture_table* contains information about the individual gestures. It lists the GestureID (PRIMARY KEY), SessionName (FOREIGN KEY on session_table) in which the gesture occurred, start and end timestamp of the geture and the start frame and end frame timestamps for the gesture. The table contains 24502 instances of the gesture along the rows.

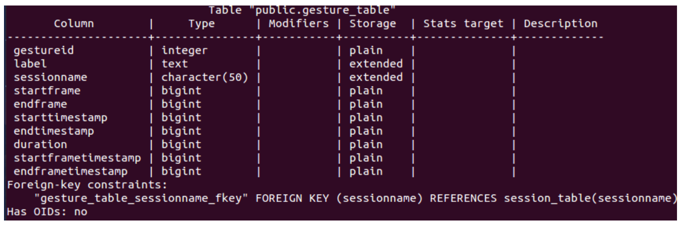
The *session_table* gives the details of each session in terms of the SessionName (PRIMARY KEY), the pair of participant in the session, blocks of videos occurring in that session, directory location of the video file for each SessionName.

The *participant_count_no_sound* table and *participant_count_with_sound* tables gives details on



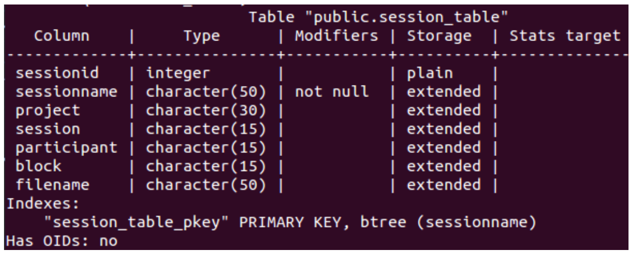
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	GestureID	Label	Project	Session Name	Session	Participant	Block	File Name	Start Frame	End Frame	Start Timestamp	End Timestamp	Duration	Start Frame Timestamp	End Frame Timestamp	
2	1	body: move, front;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	40	100	14599043	36269043	21670000	40	98	
3	2	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	100	118	36268336	42938336	6670000	98	117	
4	3	hands: into L;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	119	136	43000000	49670000	6670000	117	134	
5	4	arms: move, front; hands: L;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	136	157	49670000	57670000	8000000	134	154	
6	5	arms: rotate; hands: L;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	156	213	57599160	79929160	22330000	154	210	
7	6	arms: rotate; hands: L;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	214	269	80000000	101330000	21330000	210	265	
8	7	body: still;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	269	300	101268256	113268471	12000215	265	296	
9	8	arms: move, right; hands: L;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	300	355	113268471	135598471	22330000	296	351	
10	9	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	355	415	135599114	158268268	22669154	351	409	
11	10	arms: move, front; hands: into L;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	415	445	158268268	179598921	21330653	409	442	
12	11	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	445	488	179598921	214268417	34669496	442	489	
13	12	arms: move, back; hands: claw;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	488	598	214268417	256598417	42330000	489	598	
14	13	arms: apart; hands: claw;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	598	612	256598945	261598945	5000000	598	611	
15	14	arms: together, into contact; hands: claw;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	613	637	261670000	270340000	8670000	611	634	
16	15	arms: apart; hands: into closed, facing;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	636	647	270268161	274598161	4330000	634	645	
17	16	arms: together; hands: facing, closed;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	648	672	274670000	283340000	8670000	645	669	
18	17	arms: apart, left;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	671	692	283269172	291599028	8329856	669	690	
19	18	arms: together; hands: facing, closed;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	693	719	291670000	300670000	9000000	690	714	
20	19	arms: apart, left;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	719	745	300670000	310000000	9330000	714	739	
21	20	arms: together; hands: facing, closed;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	745	758	310000000	314670000	4670000	739	751	
22	21	hands: facing, closed;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	758	824	314670000	338670000	24000000	751	816	
23	22	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	823	906	338598838	370269127	31670289	816	897	
24	23	hands: into claw;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	906	922	370330000	376660000	6330000	897	913	
25	24	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	922	987	376598863	401938226	25339363	913	978	
26	25	body: move, back; hands: into fist;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	987	1023	401938226	415268907	13330681	978	1013	
27	26	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1023	1067	415268907	431269086	16000179	1013	1054	
28	27	arms: move, front; hands: into claw;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1067	1119	431269086	450598793	19329707	1054	1105	
29	28	arms: apart, left;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1119	1148	450598793	460938234	10339441	1105	1133	
30	29	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1148	1230	460938234	491598949	30660715	1133	1214	
31	30	arms: together; hands: into facing, closed;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1230	1277	491598949	509598742	17999793	1214	1261	
32	31	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1277	1299	509598742	517938199	8339457	1261	1283	
33	32	body: move, back;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1299	1353	517938199	537938199	20000000	1283	1336	
34	33	body: move, front;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1353	1399	537938055	554938214	17000159	1336	1380	
35	34	hands: into fist;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1399	1445	554938214	571598703	16660489	1380	1423	
36	35	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1445	1549	571598703	610938193	39339490	1423	1526	
37	36	hands: claw, down;	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1549	1755	610938193	689608193	78670000	1526	1727	
38	37	Unknown	CWC - No Sound	Session 1-Participant 2-Block 1	Session 1	Participant 2	Block 1	s01\part1_layout_p02\20151105_191251_00_RGB.avi	1755	1891	689598792	742938092	53339300	1727	1860	

Fig. 1: Gesture Dataset Features



```
                              Table "public.gesture_table"
       Column        |     Type      | Modifiers | Storage  | Stats target | Description
---------------------+---------------+-----------+----------+--------------+------------
 gestureid           | integer       |           | plain    |              |
 label               | text          |           | extended |              |
 sessionname         | character(50) |           | extended |              |
 startframe          | bigint        |           | plain    |              |
 endframe            | bigint        |           | plain    |              |
 starttimestamp      | bigint        |           | plain    |              |
 endtimestamp        | bigint        |           | plain    |              |
 duration            | bigint        |           | plain    |              |
 startframetimestamp | bigint        |           | plain    |              |
 endframetimestamp   | bigint        |           | plain    |              |
Foreign-key constraints:
    "gesture_table_sessionname_fkey" FOREIGN KEY (sessionname) REFERENCES session_table(sessionname)
Has OIDs: no
```

Fig. 2: gesture_table Schema in PostgreSQL



```
                      Table "public.session_table"
   Column    |     Type      | Modifiers | Storage  | Stats target
-------------+---------------+-----------+----------+--------------
 sessionid   | integer       |           | plain    |
 sessionname | character(50) | not null  | extended |
 project     | character(30) |           | extended |
 session     | character(15) |           | extended |
 participant | character(15) |           | extended |
 block       | character(15) |           | extended |
 filename    | character(50) |           | extended |
Indexes:
    "session_table_pkey" PRIMARY KEY, btree (sessionname)
Has OIDs: no
```

Fig. 3: session_table Schema in PostgreSQL

how many participants performed that particular gesture in corresponding with/no sound session. It also provides the count of individual instances of each of the gestures in label column. The *main_table* is a comprehensive table that contains all the attributes from the tables mentioned above.

B. Observations on RDBMS

Consider the following simple query which aims to find the columns where the condition in WHERE clause are met which asks for a specific set of gestures from the records of gestures present in the *main_table*.

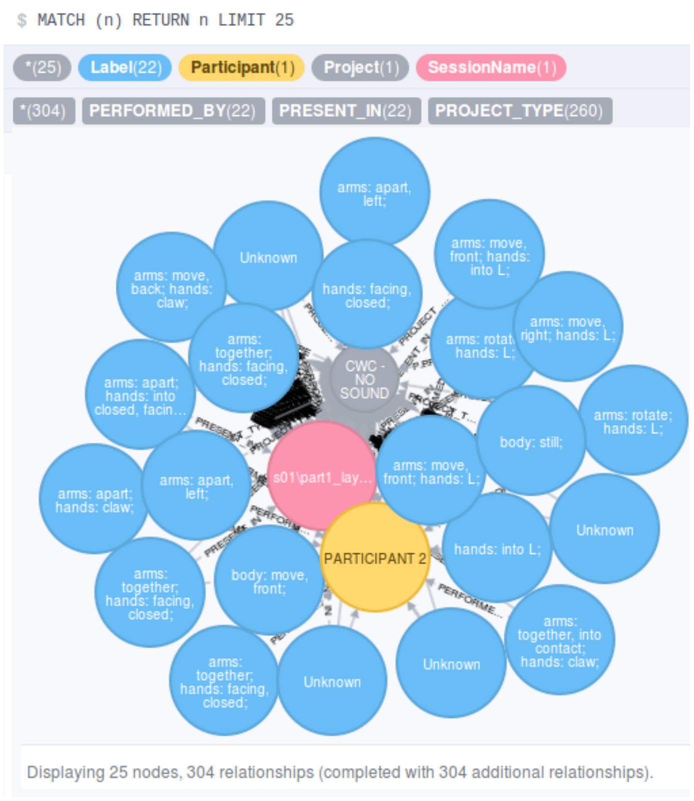
```
SELECT Label, Session, FileName, StartFrame,
EndFrame, Participant
FROM main_table
WHERE Label IN ('body: move, front;', 'head:
nod;', 'RA: move, up;') AND Session IN
('Session 2', 'Session 3') AND Participant
IN ('Participant 4', 'Participant 5');
```

Based on this query we made the observation that as the size of the table increases indicated by increase in the number of records, the time require to process the query increases linearly as a function of number of records. We used 6 different sub-tables cropped from the *main_table* for this observation. We varied the number of records from 3350 in *main_3sessions* table to 24502 in the *main_table*. Figure 12 shows how the query time goes up linearly as the number of records increase.

C. Limitations of RDBMS for our Dataset

Based on the observations, we found that modeling the gesture dataset as an RDBMS will linearly increase the query time as we keep adding additional data into the database. In future, we expect to conduct the similar experiments as described in Introduction on new pairs of participants. Also, along with the gestures we have another set of labels called as 'Intents' that tell what was the intent behind every gesture in that particular context. This gesture to intent relationship is many to many. It is difficult to model this relationship just using two separate relational tables. Therefore, we concluded that the we should find an alternative approach to model the given dataset [5].

Another limitation that we found was that RDBMS was unable to represent the 'relations' that are present in the dataset. For instance, a gesture (say *body: move, front;*) is 'performed by' a multiple participants. Here, we can observe that there is unidirectional relation existing between a gesture and a participant. RDBMS is unable to model this kind of relation and therefore it requires the operations like JOIN on two different tables - *gesture_table* and *session_table*. Moreover, the attributes in the *gesture_table* such as Start-Timestamp, EndTimestamp, StartFrameTimestamp, and End-FrameTimestamp can be represented as the properties of the gesture itself rather than having a separate column for each attribute. A graph database is capable of modeling the relations between gesture and participant, gesture and session, and so



```
$ MATCH (n) RETURN n LIMIT 25
```

*(25)
Label(22)
Participant(1)
Project(1)
SessionName(1)
*(304)
PERFORMED_BY(22)
PRESENT_IN(22)
PROJECT_TYPE(260)
arms: apart, left;
Unknown
arms: move, front; hands: into L;
arms: move, back; hands: claw;
hands: facing, closed;
arms: move, right; hands: L;
arms: together; hands: facing, closed;
CWC - NO SOUND
arms: rotat hands: L;
arms: apart; hands: into closed, facin…
body: still;
arms: rotate; hands: L;
arms: apart; hands: claw;
arms: apart, left;
s01\part1_lay…
arms: move, front; hands: L;
Unknown
PARTICIPANT 2
hands: into L;
arms: together; hands: facing, closed;
body: move, front;
arms: together, into contact; hands: claw;
Unknown
Unknown
arms: together; hands: facing, closed;
Unknown

Displaying 25 nodes, 304 relationships (completed with 304 additional relationships).

Fig. 4: Graph Database showing 25 Nodes

on. Therefore, we decided to explore Neo4j - a native graph database - for our dataset that contains relations and could be scaled in future by preserving these relations.

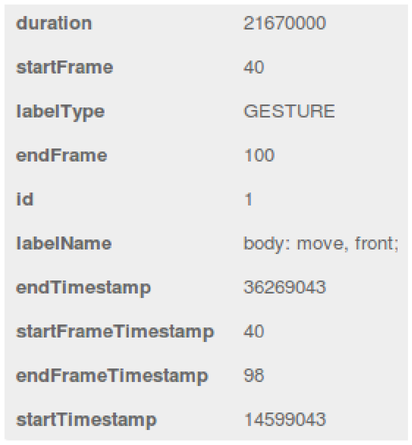
III. Gesture Database as a Graph in Neo4j [4]

The primary motivation behind exploring graph databases was to be able to characterize the relations between gestures, sessions, and participants. We decided to use Neo4j since it is freely available graph database under community edition and it is the leading graph database. Neo4j uses a powerful query language known as *cypher* which allows to express complex relationships and patterns existing between nodes that represent data points.

A. *Structure of Gesture Database*

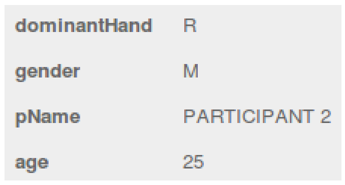
For our gesture database there are various schema in which the graph database could be modeled. We tried a few different schema before settling to the one described in details below. Figure below, depicts the graph-based structure we used for the gesture database.

*1) **Nodes**:* Each of the gesture in the table shown in figure 1 is assigned a node in the graph database. Since the attributes such as StartTimestamp, EndTimestamp present in the table are the characteristics of the gesture itself, they are assigned as the properties for the gesture. The schema for the gesture nodes is given in the figure below. Next, we have a node for each instance of Session Name from the gesture table. It has properties that contain information about the type of project



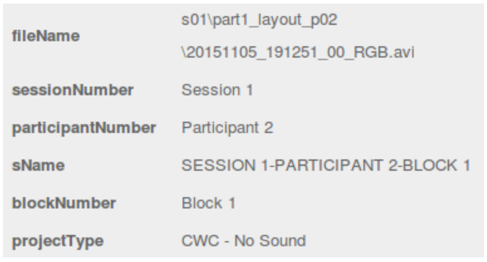
duration	21670000
startFrame	40
labelType	GESTURE
endFrame	100
id	1
labelName	body: move, front;
endTimestamp	36269043
startFrameTimestamp	40
endFrameTimestamp	98
startTimestamp	14599043

Fig. 5: Properties of Gesture Node (blue node)



dominantHand	R
gender	M
pName	PARTICIPANT 2
age	25

Fig. 6: Properties of Participant Node (yellow node)



fileName	s01\part1_layout_p02 \20151105_191251_00_RGB.avi
sessionNumber	Session 1
participantNumber	Participant 2
sName	SESSION 1-PARTICIPANT 2-BLOCK 1
blockNumber	Block 1
projectType	CWC - No Sound

Fig. 7: Properties of a Gesture Node (pink node)

(with/no sound), location path for the video trial file, etc. Then we have a node for the each type of project viz., No Sound and With Sound. Finally, we have a node for every participant containing details about the participant including age, gender, dominant hand, etc.

While constructing the graph database we need to ensure that uniqueness constraints are enforced on to a node property of every node that ensures that only one node with that given property is present in the graph database. If a new data point with same property name arrives, then we simply merge it into the pre-existing node. The uniqueness constraints for gesture and session name nodes are given below.

```
CREATE CONSTRAINT ON (l:Label) ASSERT l.id
IS UNIQUE;
CREATE CONSTRAINT ON (sn:SessionName)
ASSERT sn.sName IS UNIQUE;
```

We create a new node using following command with an example for gesture node where a line is a line read in LOAD CSV command of *py2neo* package. And set the properties for

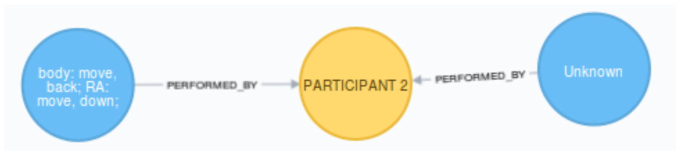


Fig. 8: PERFORMED_BY Relationship

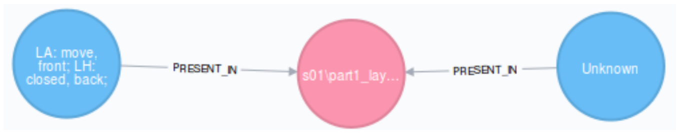


Fig. 9: PRESENT_IN Relationship

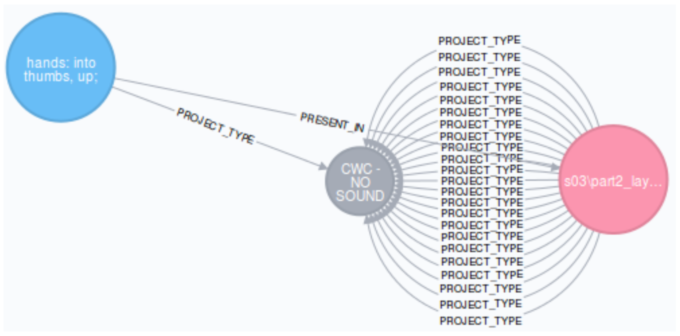


Fig. 10: PROJECT_TYPE Relationship

the node using the SET command as shown below.

```
CREATE (l:Label{id:TOINT(line.GestureID)})
SET l.labelName = line.Label
```

2) ***Relationships:*** We have three main relationships that are visible in our dataset. First is the PERFORMED_BY relationship that goes from a gesture node to a participant node. Second is the PRESENT_IN relationship flowing from a gesture node to a session name node with a meaning as described by the relation name itself. Third is the PROJECT_TYPE relationship telling what project does the gesture or the session belong.

Following example shows how to create a relationship called PRESENT_IN from label to session name.

```
CREATE (l)-[pi:PRESENT_IN]->(sessionName)
```

B. *Testing on Graph Database*

We varied the number of nodes and relationships to create the graph database. Table in figure 11 shows the variation of number of nodes, relationships, properties, and labels as we increase the size of database.

1) ***Time Comparison: Neo4j Vs. PostgreSQL:*** In order to test the query execution time on Neo4j and compare it with that of PostgreSQL, we used the following query which is a cypher equivalent of a PostgreSQL query retrieving the exact same nodes as the records in the PostgreSQL.

Number of Sessions	Nodes	Relations	Properties	Labels	GraphDB Generation Time
3	3406	13400	60356	3406	2308 ms
7	10088	39808	179272	10088	3705 ms
10	15413	60884	274170	15413	4125 ms
13	18649	73480	330939	18649	5507 ms
16	21752	85660	314392	21752	6517 ms
20	24904	98008	441438	24904	7660 ms

Fig. 11: Neo4j: Variable Size of Database

```
MATCH (p:Participant)<-[r:PERFORMED_BY]-(n:
Label)-[:PRESENT_IN]->(SessionName)
WHERE n.labelName IN ['body: move, front;'
, 'head: nod;', 'RA: move, up;'] AND
SessionName.sessionNumber IN ["Session 2",
"Session 3"] AND p.pName IN ["PARTICIPANT 4",
"PARTICIPANT 5"]
RETURN n.labelName, SessionName.sessionNumber,
SessionName.fileName, n.startFrame, n.endFrame,
p.pName;
```

This query finds the set of gestures conditioned by WHERE clause and connected to the nodes Session 2 and 3. We observed that, the query execution time for neo4j is independent of the number of nodes present in the database. The reason is that, Neo4j scans through the gesture nodes connected only to the session nodes Session 2 and 3. So even if the number of nodes in the database increase as indicated by increase in session nodes (Session 4 to 20) and gesture nodes, Neo4j will not scan the session nodes other than Session 2 and 3. As a result the scanning time will always be constant irrespective of the size of the database. Graph in figure 12 depicts that the query time is constant (approximately 120 ms).

Contrary to Neo4j, PostgreSQL takes approximately 3.8 ms to process the equivalent query with the same outputs when the number of records are 3350. However, as the number of records in the database increase to 24502, the query processing time goes up to 22.08 ms. This is a linear increase as a function of number of records. In future, we are expecting to have more gesture data records (upto 150K). So we can conclude that the query execution time will approximately 132 ms after the linear increase. In this case when dataset increases linearly, Neo4j will serve as a faster and intuitive option for our dataset.

We also observed one peculiar characteristic in Neo4j database. When we start the Neo4j database with 24904 nodes at http://localhost:7474/, Neo4j takes around 700 to 900 ms to process and execute the first query. Thereafter, the execution time reduces exponentially to 140 ms for second instance of the same query, 115 ms for third query, 90 ms for fourth query and so on to settle in the range of 50 to 70 ms. We attribute this behaviour to caching of the data or graph structure at the

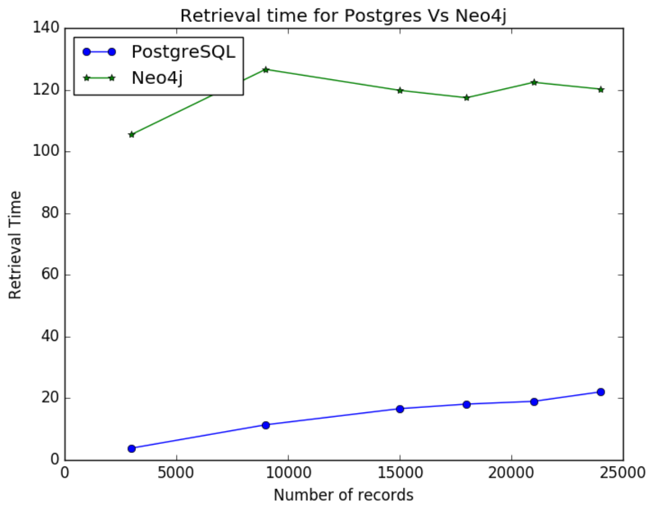


Fig. 12: Variation of Query Processing Time against Number of Records/Nodes

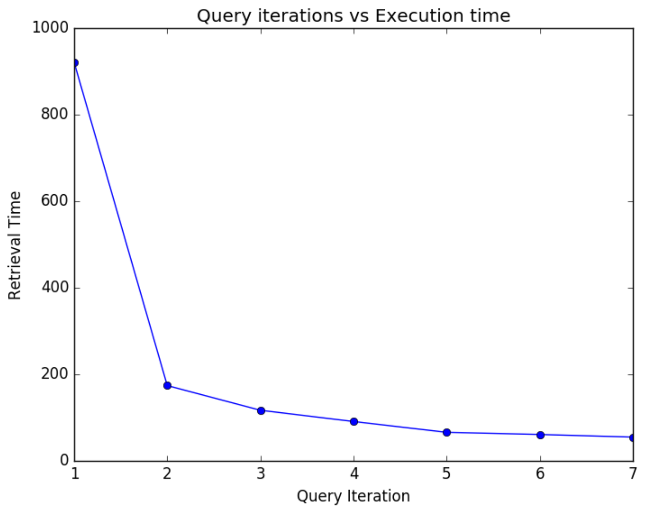


Fig. 13: Variation of Query Processing Time against Query Iteration

cold start. This reduces the process time for the further queries dramatically. Graph in the figure 13 shows this behaviour.

C. Advantages of Graph Database over RDBMS

• **Avoiding costly JOIN operations:** In RDBMS, if the data is distributed in multiple tables then, the JOIN operation is used at the query-time in order to get the results involving attributes from multiple tables. In such operations the pair of primary key and foreign key is matched to form a join. The join operation is expensive in terms of the computation time and memory consumption. As the size of tables in the database increases the cost of join increases in exponential manner [4] [5]. For example, consider a query similar to the discussed above with a difference that we will have *gesture_table* and *session_table* with a JOIN operation performed on SessionName attribute (primary-foreign key relation). This query takes 27.328 ms to execute on entire 24502 record in *gesture_table* which is more than it usually takes if we query on *main_table* without involving any JOIN operations.

```
SELECT Label, session_table.Session, session
_table.FileName, StartFrame, EndFrame,
session_table.Participant FROM gesture_table
FULL JOIN session_table ON gesture_table.
SessionName = session_table.SessionName
WHERE gesture_table.Label IN ('body: move,
front;', 'head: nod;', 'RA: move, up;') AND
session_table.Session IN ('Session 2',
'Session 3') AND session_table.Participant
IN ('Participant 4', 'Participant 5');
```

Conversely, in graph database each node representing a gesture is directly connected to the session and participant nodes under the relationship given as

```
(p:Participant)<-[r:PERFORMED_BY]-(n:Label)
-[:PRESENT_IN]->(SessionName)
```

The gesture node has a list of relationships that is used to have a direct access to the session and participant nodes which have relation with the given gesture node. Thus every session or participant that performed the gesture is one hop away from the gesture node. Therefore, this eliminated the costly JOIN operations as performed in case of the RDBMS.

• **Ability to represent intuitive relationships:** One advantage of using a graph model is that it allows to represent the data in natural and intuitive way as it is present in the real world. Graph model has ability to define the relationship between two data nodes, and these relationships are treated just as the data nodes in that we could assign properties to them as well. Thus, graphs have advantage over the RDBMS in that we do not have to force the data which involves visible relations into a tabular form where it looses these relations or it becomes difficult to extract these relations.

• **Ability to assign properties to nodes and relationships:** Graph database gives ability to assign properties to a relationship. Consequently, we can devise queries that operate on the properties of the relations. This materialization of relationships reduces the penalties for complex queries at runtime. By identifying the attributes that can be assigned to the relationships between nodes rather than creating nodes itself, helps in compacting the structure of the database.

IV. Implementation of Video File Retrieval

A. *Connecting to Neo4j via py2neo [6][7]*

After creating the graph database as explained in the sessions above, we need to have ability to connect to the database using a high level language wrapper. We decided to use Python for connecting to Neo4j given that py2neo package has a good technical support. The code snippet explains how to specify a url under localhost. The next step is to provide authentication credentials and create an instance of a database in Python. Then we write a cypher query in triple quotes and execute it with a simple 'run' command to get the results in 'data' variable.

```
url = "http://localhost:7474/db/data/"
authenticate("localhost:7474", "neo4j", "cwc")
graph = Graph(url)
query = """ * """
data = graph.run(query)
```

B. *Interface in Python*

As a pilot study, we developed an interface for the user to select the gestures, which will be retrieved from the database after following the process mentioned above (Figure 14). It consists of the target directory to store the videos, and a list of the gestures to select. The 'Get Documents' button will do the database connections, fire the respective queries, and get the appropriate file names for the marked gestures. Each gesture sample will be stored in its parent folder. Figure 15 shows the videos retrieved for 'arms: apart, left;' in its own folder.

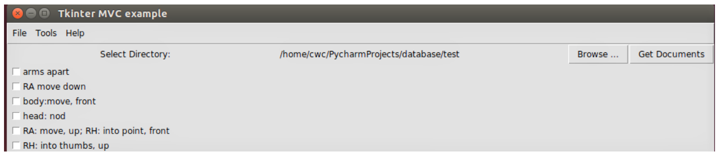


Fig. 14: Interface for retrieval

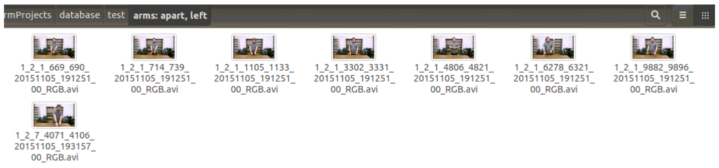

rmProjects
database
test
arms: apart, left
1_2_1_669_690_20151105_191251_00_RGB.avi
1_2_1_714_739_20151105_191251_00_RGB.avi
1_2_1_1105_1133_20151105_191251_00_RGB.avi
1_2_1_3302_3331_20151105_191251_00_RGB.avi
1_2_1_4806_4821_20151105_191251_00_RGB.avi
1_2_1_6278_6321_20151105_191251_00_RGB.avi
1_2_1_9882_9896_20151105_191251_00_RGB.avi
1_2_7_4071_4106_20151105_193157_00_RGB.avi

Fig. 15: Videos Retrieved in folder

V. Conclusion

The goal of this project was to explore and find a suitable and scalable database model for the gesture database. First, we modeled the gesture dataset into a relational database using PostgreSQL. Testing with some standard queries on RDBMS for the execution time revealed that as we increase the size of the table by increasing the gesture records, the execution time increases linearly. Therefore, in future as we expect to scale up our data, the execution time is expected to increase as well. Thus, we explored the graph database model for the gesture dataset.

Any dataset have multiple possible ways in which it could be structured in the graph database. We implemented three different models to fit our data in Neo4j. Finally, we settled on the most intuitive structure as we would naturally visualize for the dataset as described in the sections above. Graph database provided the ability to design the relationships between various nodes and querying the database became natural. Finally, we observed that addition of nodes in existing graph do not disturb the structure or the syntax of the queries used earlier. Also, the query execution time remains approximately constant as we scale up the database by adding more gesture nodes.

VI. Future Work

Along the lines of experimenting with different graph database structures for our dataset, we aim to restructure the database by assigning the time frame properties to the relationships between the gesture node and the session node. This structure will help us reduce the instances of gesture nodes which, we believe, will reduce query execution time. Further, we will be scaling up the database with addition of new gesture instances. We also want to incorporate the many-to-many relationships between gestures and intents after addition of intent nodes in this study.

Acknowledgment

We would like to acknowledge and thank the faculty and students from Computer Vision Laboratory at CSU for the gesture dataset. The dataset collection and labeling was a lengthy process wherein they have a major contribution.

References

[1] EGGNOG: A continuous, multi-modal data set of naturally occurring gestures with ground truth labels, B. Draper, et al., 2016

[2] https://www.postgresql.org/

[3] https://pypi.python.org/pypi/psycopg2

[4] Graph Databases, Robinson, I. and Webber, J., et al., ISBN 9781449356262, O'Reilly, 2013.

[5] https://neo4j.com/

[6] http://py2neo.org/v3/

[7] http://nicolewhite.github.io/neo4j-jupyter/hello-world.html