Project Report: Query by Gesture - Polypheny

Jannik Jaberg and Jonas Rudin

University of Basel
Seminar: Modern Human Machine Interaction (MHMI)
Autumn Semester 2019

1 Introduction

The goal of this project is to input relational query plans via gesture. For this, we used two existing projects of the University of Basel and connected them. The gesture recognition is done by Deepmime while the query visualization and processing are done by Polypheny. To realize our goal, we implemented a bridge that parses detected gestures to a JSON String which can be interpreted and represented in the Polypheny UI. For this, each gesture is assigned to exactly one query attribute or navigation commands to build the tree. These hard-linked states forfeit the intuitiveness of the interaction.

The project idea itself is innovative and futuristic. The invention of the touch-screen has been the foundation of Human-Maschine-Interaction (HMI) without using a keyboard or a mouse. The next step would be to not even touch anything anymore, which can be realised by voice and gesture control. Alexa made already a big step into this direction. If a reliable gesture control is developed, it will only take a few steps to feel like Tony Stark, while talking to Jarvis and controlling his computer free handed and only by gestures and voice control. In this report, we find out, how many more steps have to be taken to achieve a useful and reliable gesture control.

2 Concepts

As mentioned in the introduction the goal is to connect the two separate systems Deepmime and Polypheny to have an additional HMI for Polypheny, instead of only mouse and keyboard.

2.1 Deepmime

Deepmime is a gesture recognition implementation based on a 3D residual neural network (ResNet). A residual neural network is an artificial neural network (ANN) of a kind that builds on constructs known from pyramidal cells in the cerebral cortex.[1] For the implementation to recognise gestures on the fly, the skip connection is the relevant aspect for choosing a ResNet. The implementation we worked with takes a pre-trained model and classifies the video feed into a probability which labels it could be.

[1] `https://en.wikipedia.org/wiki/Residual_neural_network`

2.2 Polypheny

"The last years have seen a vast diversification on the database market. In contrast to the “one-size-fits-all” paradigm according to which systems have been designed in the past, today’s database management systems (DBMS) are tuned for particular workloads. [...] In such cases, multistores are increasingly gaining popularity. Rather than supporting one single database paradigm and addressing one particular workload, multistores encompass several DBMSs that store data in different schemas and allow to route requests on a per-query level to the most appropriate system.[...]"[2]

Polypheny addresses exactly this problem. For our project, we are not dependant on the polystore aspect of the database but on the relational algebra part. Polypheny supports a graphical user interface (Polypheny-UI) where SQL queries can be built, connected via drag and drop and ran afterwards. There the user must understand SQL or relational algebra to build such a tree, hence our implementation also does not address users which can not build statements.

3 Implementation

3.1 Query by Gesture Architecture

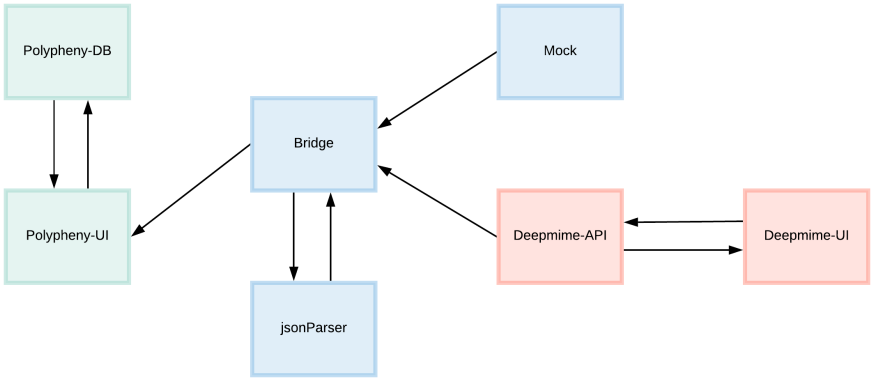

Fig. 1: Overview Architecture Diagram

As seen above in Figure 1 the architecture of the project can be split into three modules: Polypheny (cyan), Query by Gesture Bridge (blue) and Deepmime (red). Deepmime classifies gestures from a live video and sends them to the

[2] Icarus: Towards a Multistore Database System (`https://edoc.unibas.ch/58210/`)

Query by Gesture Bridge. There they are processed and alter the relational query which is stored as a JSON object. This string then gets transferred to the Polypheny module where it gets displayed and in the end, the query can get executed. The modules are connected via a WebSockets.

3.2 Deepmime

Deepmime is split into the UI and the API. In the UI you can upload a video or do life gesture recognition. The segments from the videos are sent to the API where they are classified. The classification is then again displayed in the UI. In the API, after the video segment has been classified, the results get analysed. If the same gesture was detected several times in a row and its classification score is higher than a certain threshold it is sent to the server in the Query by Gesture Bridge. The UI is written in Angular and the API in Python.

3.3 Query by Gesture Bridge

As the name already suggests, this module serves as a bridge between the two other modules. The server acts as the WebSocket server between the two other modules. It takes gestures from Deepmime and translates them into a query plan in a JSON string that is then sent to Polypheny. The translation is done in the jsonParser. This class takes a gesture and executes the command the gesture is allocated to. Depending on that command it can add a new operator, change the specification of an operator or delete the last or all inserted operators. It also automatically calculates the position of all the operators that form the query. Additionally, this module also holds the mock file. It mocks the Deepmime module and is also able to connect to the server and send gestures. This whole module is written in Python.

3.4 Polypheny

This module only uses the polypheny-UI part of the whole Polypheny. An overview of the used components and services can be seen in Figure 2. The *right-sidebar.component* takes the connection details (ip:port) for the WebSocket connection and saves them in the *webui-settings.service*. The "connect"/"disconnect" button starts/ends the WebSocket connection to the Query by Gesture Bridge. It does that by communicating with the *relational-algebra.component* over the *rightsidebar-to-relationalalgebra.service*. The *relational-algebra.component* then activates the *web-socket.service* which gets the connection details from *webui-settings.service* and connects/disconnects to the Query by Gesture Bridge. Once the connection is established, the *web-socket.service* listens and sends any incomming JSON objects to the *relational-algebra.component* which then processes it, either by updating the query or deleting it. The Polypheny-UI is also written in Angular. As usual in Angular, the components are built from a typescript, a CSS and an HTML file. The services are only a typescript file.

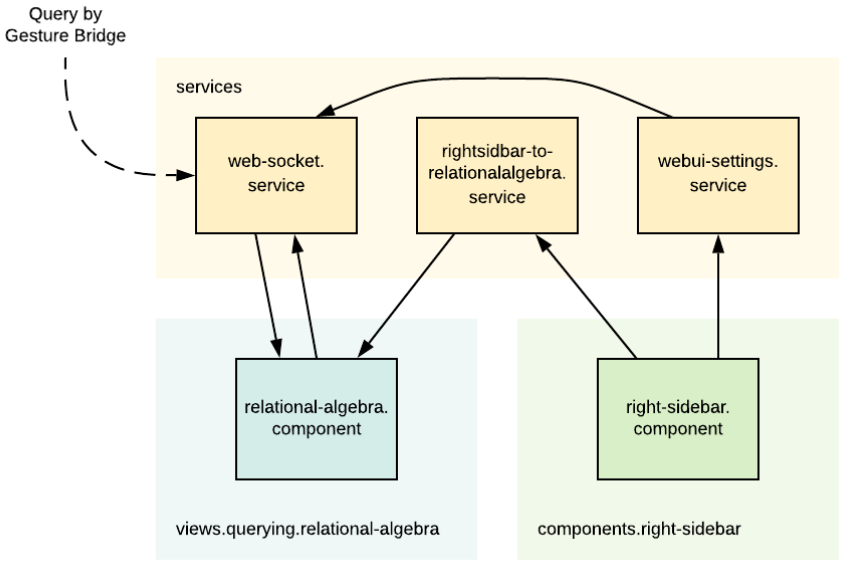

Fig. 2: Components and Services used by Project in Polypheny-UI

4 Usage

To run the program, Deepmime, Polypheny and the server (Query by Gesture Bridge) must be run individually. It is recommended to start the server first. For this you have to run the following comands:

```
pip3 install python-socketio
python3 server.py
```

After that you can run Polypheny and Deepmime. Open the two applications in different browser windows or tabs (eg as seen in Figure 3 with the server running in the lower left-hand corner). Deepmime will automatically connect to the server when you start a live video. In Polypheny you must be in the `Plan Builder` to connect to the server in the settings. There you are also able to change and save the IP and port, see Figure 4 (needs to be changed in server and Deepmime as well).

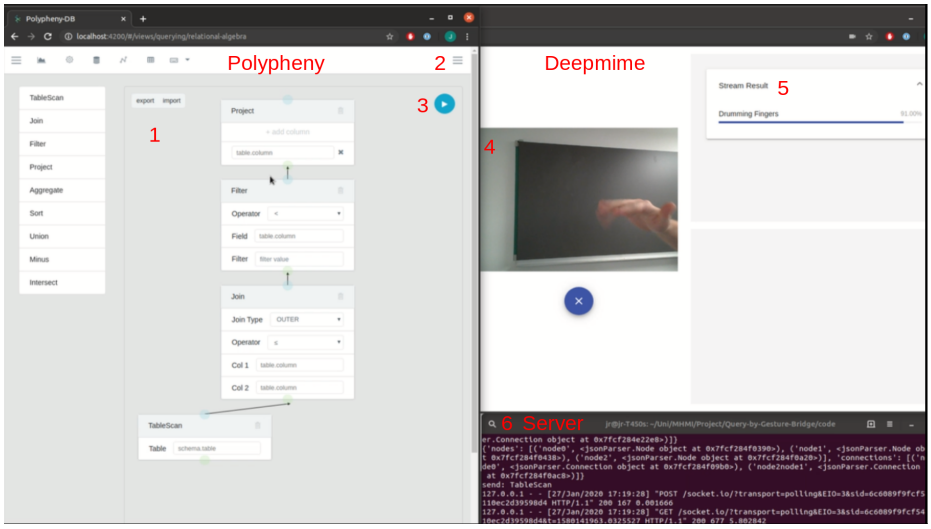

Fig. 3: Example layout of Query by Gesture

1: query plan builder
2: settings
3: run query
4: video input
5: found gesture with classification score
6: terminal which runs the server

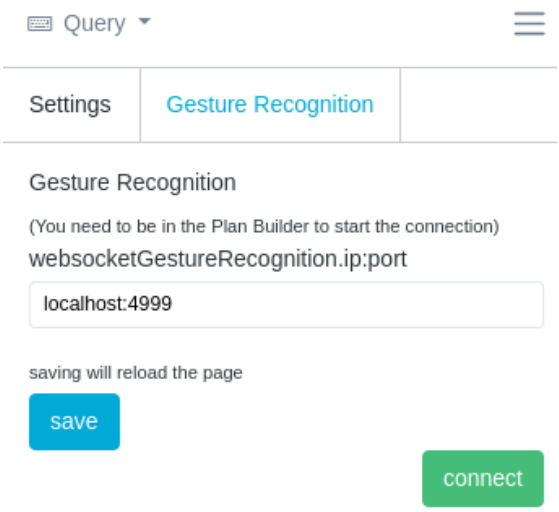

Fig. 4: Settings for Gesture Recognition in Polypheny

When everything is running and connected you can use the following gestures to build a query:

operators

- Drumming Fingers: TableScan
- Zooming In With Two Fingers: Join
- Zooming Out With Two Fingers: Join
- Pushing Hand Away: Project
- Pushing Hand In: Project
- Shaking Hand: Sort
- Stop Sign: Filter

navigation

- Swiping Left: next
- Swiping Right: next
- Thumb Up: confirm
- Thumb Down: cancel
- Swiping Down: undo
- Swiping Up: undo
- Turning Hand Clockwise: delete
- Turning Hand Counterclockwise: 'delete

There are some things you need to know when building your query. The "Join" and "Filter" operators have two respectively one additional fields you have to specify via gesture control. You can do this via "next" (which goes through the possible options) and "confirm" (selects the shown option). Additionally the "undo" will delete the last inserted operator and the "delete" will get rid of the whole query.
If you only want to insert queries without running it in the end it is enough if you run Polypheny-UI with `ng serve`. Furthermore you can mock Deepmime by simply running:

```
python3 mock.py
```

In the terminal you can then type in the commands (e.g. TableScan, confirm, next etc.).

Example videos:

- Deepmime: `https://youtu.be/2SVj17GXv7s`
- Mock: `https://youtu.be/DYrqiwcTzNs`

5 Conclusion

5.1 Future Work

Definitely feedback in any of the three modules has to be implemented. This could be done via text in the Deepmime-UI or even better with highlighting in

the query plan. The reason why we did not implement such a simple-looking task was the structure of the Polypheny-UI. Since it was still in development minor changes pushed us back several steps. Also, the behaviour of angular was sometimes really unexpected and we had border artefacts were there should not be any.

A better way to implement feedback to our hard-linked implementation would have been a new popup window in the Polypheny-UI. There should appear a camera feed and a working space with each operator which is worked on. Afterwards, the tree can be set up in the next step and changes can be made in this popup window. Still, there is the same problem as in our implementation - gestures can still cancel out themselves[3].

Also running the whole system is quite laborious. The three components have to be started separately. Polypheny is forgiving, but if something in Deepmime is a bit off, the whole Deepmime-API has to be reloaded, which takes at least a minute. It would be pleasing if after Polypheny is started, the other components can be run via UI input in Polypheny. This would fit the task above where the gesture recognition is implemented into a popup window.

Further, might be difficultly complicated, we thought of a drag and drop style interaction to solve this problem, where the mouse is replaced by the gestures of thumb and index finger. It should behave like a touchscreen. Precisely, the camera feed and the Polypheny-UI are overlaid so the users see where the fingers are and the operations can be picked by closing thumb and index finger and dragging it into the working field. Unfortunately, we could not realise this, because as expected, it was too complicated. This method would have been much more intuitive and has a better human-machine interaction.

Speaking of intuitiveness, a voice-controlled query plan seems to be the next easier way to have a mouse- and keyboard-less experience. But since most columns in a database have not "normal" everyday names, probably also an Alexa based system would not bring the joy we expect a user to have with a new HMI.

5.2 Lessons Learned

Across the board, we are not satisfied with what we handed in. Since we had so much trouble with other things which kept us from improving or even deliver features. It is no excuse, but the overhead we had to deal in this project took away valuable time, which we would have needed. Also connecting the two systems was not as easy as we thought. Additionally, the unforgiving and non-creative manner of the project did not help us develop features for our bridge. It resulted in boredom and frustration. Nevertheless we learned many things in this project:

[3] `https://youtu.be/ESOhRCdkor0`

Angular is a really powerful typescript framework, if and only if you understand it. For our project, we would have needed much more understanding or experience. It was just not possible to learn Angular to the point of understanding which we needed, in comparison to what other tasks we had during the semester. Therefore, we were limited in doing changes in the Polypheny-UI. Nevertheless, the power that Angular provides should not be underestimated.

We had many **Ideas** for the whole project, some of them mentioned in the future work. Unfortunately, all of these additional ideas could not be implemented since we faced so many side problems. It is important to be aware that not every project is fun and you can bring your creativity into it. Sometimes an idea of something really interesting turns out as a complete logical algorithmic boredom without a real sense of achievement.

Planning is always a big factor in project work. As discussed all over this report our planning was off and problems we had never faced. Apart from Deepmime not working on our computers and the setup problems with Polypheny, at the beginning of implementing the communication between the two modules, the socket implementation was more difficult than expected. After the first establishment of the communication, Cross-Origin Resource Sharing (CORS) problems occurred on one of the machines which we could not fix until the end of the project. Then we firstly could start resolving the actual problem. At that moment we were already really frustrated. We can not fix such an issue with another approach to the project but should bury our heads in the sand.

The **Relevance** of the examined topic could be huge in future technology development. Seen in one of our papers[4] we presented to the other seminar participants. The big feedback of the crowd with ideas, questions and interest in general was astonishing. Unfortunately, the progress in this field is still in its starting hole and our two-man show only scraped on the surface of the iceberg. Therefore **many steps** have to be taken to achieve a **reliable gesture control**. May it be in developing a more robust gesture recognition or have a forgiving, navigatable algorithm to control an UI with gestures.

[4] `https://www.mdpi.com/1424-8220/19/1/59/htm`