Project 6: Semester Project – Final submission

Name of Project: *Enabling Novice Coaches: A Feedback-based Approach to Robot Skill Repair*

Team Members: *Jack Kawell, Aaquib Tabrez, James Watson, Christine Chang*

Final State of System Statement:

Our system is intended to solicit feedback from a human about the performance of a robot with which they are collaborating, and then utilize that feedback to improve the robot's future performance. We built upon a previous project by first converting portions of the code base into an OOP paradigm. After that, we expanded upon the research thrust of the work by adding more robust sensory input and a verbal feedback interface that allows users to easily adjust the robot's policy with only their voice. Additionally, we integrated an explainable artificial intelligence (xAI) component which allows the robot to explain why it behaved the way it did and thus aid the human to more easily adjust its behavior through simple feedback. We also visualize the movement of the robot in simulation, as well as visually portray the path implemented. Each of these components is described in more depth below.

Targeted query from the demonstrated trajectory to elicit desired feedback for accelerated learning and Parsing User Feedback into Usable Constraints

Targeted queries are generated through recognition of the faulty states based on the affect detection from the human. The intuition behind the skill augmentation is that the robot has already learned a skill - like pick-and-place or object handover to human - using modern learning algorithms such as Reinforcement Learning or Learning from demonstration (LfD), but the learned skill or task does not account for the human factor in the environment. For example, when the robotic arm learns to hand over an object and tries to do a knife handover to a person, it does not have precognition about sharp tools being dangerous, especially when handing them over to people. This is just one example, but in general a robot cannot learn certain social aspects and norms that we have been trained implicitly from birth, but it can acquire them by intelligent queries.

Our algorithm for generating the targeted query can be summarized as follows:

1. Determining the states of interest (S):

   This is done by observing the human during the skill/task execution to determine the states where a change in human affect is noticed (in the knife handover example, people might get scared or angry at the robot for the pointing sharp end towards them)

2. Determining the predicates that satisfy the states of interest:

   Mapping the states to predicates that are getting activated. Predicates are Boolean state classifiers as found in traditional STRIPS-style [1] planning. When they are paired with string templates describing the meaning of true or false classification, they offer a means of identifying specific regions of state space in a representation-agnostic manner. Thus,

predicates provide a meaningful abstraction and are good candidates for relaying comprehensible and interpretable policy updates.

3. Minimize the predicates as minimum set cover problem:

   This is the problem of efficiently describing state regions as a set cover problem, trying to find the smallest logical expression of communicable predicates to succinctly describe target states similar to Hayes and Shah [2]. In the case of cc-LfD [3], which we are using in our system, these are already constraints applied to keyframes that can be directly iterated for the queries.

4. Use the solution from minimum set cover (i.e. predicates) as the binary queries:

   We iterate over each constraint (predicates) and get feedback from the novice coach on whether to update that constraint or keep it the same.

5. Parse the feedback using bag of words or simple string search:

   The robot waits for the verbal feedback to each constraint (we use Google's Speech-to-Text to get the string), then we do a simple bag of words sentence classification. We use this to update the constraints for the skill. Then we repeat 1-5 until no negative emotional affect from the person is observed.

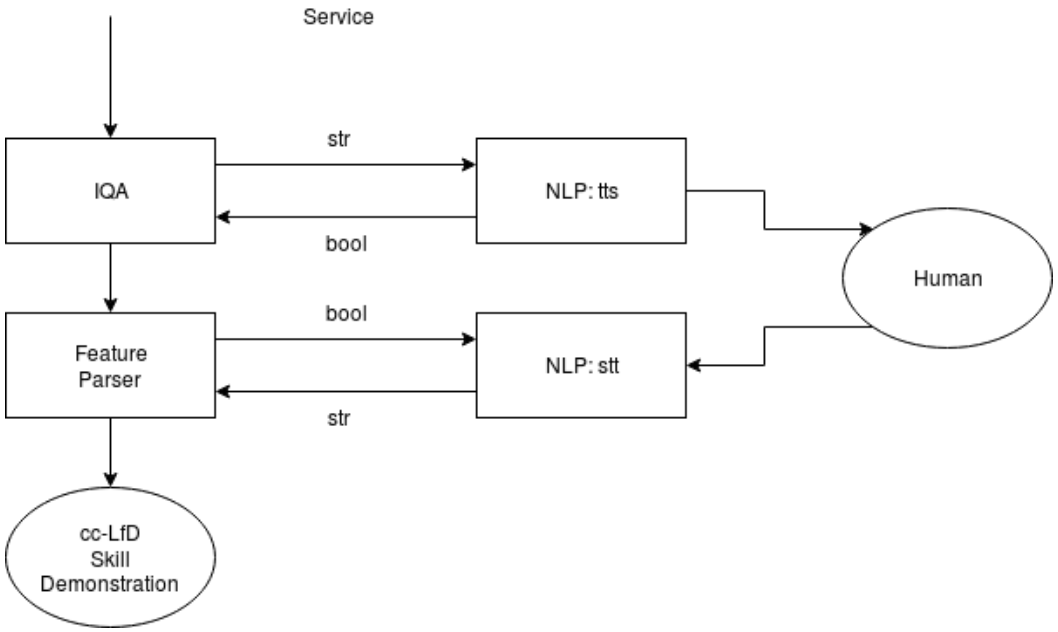

Figure 1. Architecture diagram for the components involved in the Intelligent Query generation (IQA) and parsing using the ROS framework.

Graphical 3D Display of Robot and Constraints

Various aspects of the constraints and the model are now displayed visually for a human user. We utilize RViz, a visualizer for ROS. In addition to the planned path being shown in space, the actual path that the robot takes is now traced as well. This provides a user with information about

the original intent as well as the actual history of the arm movement. An object is shown in the gripper of the robot arm, in this case a mug, to exemplify the real-world use of CC-LfD. In addition to this, the text generated in the speech-to-text translation from the human is shown in the window. This provides the human user with visual feedback about the information being generated and used by the robot. The planned path, traced path, and text are all enabled by subscribing to publishers with that information, coming from the system described in the section above.

References

[1] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. Artificial intelligence, 2(3-4):189–208, 1971.

[2] B. Hayes and J. Shah. Improving robot controller transparency through autonomous policy explanation. In Proceedings of the 12th ACM/IEEE International Conference on Human-Robot Interaction, 2017.

[3] Carl Mueller, Jeff Venicx, and Bradley Hayes. Robust robot learning from demonstration and skill repair using conceptual constraints. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2018), 2018.

Final Class Diagram and Comparison Statement:

- Because the UML diagrams are rather large, they can be seen in full resolution (in both draw.io and PNG format) at this link: https://drive.google.com/open?id=1lFQBOxxwKoSOWwjsWe4xhwcjwxxBwhYv

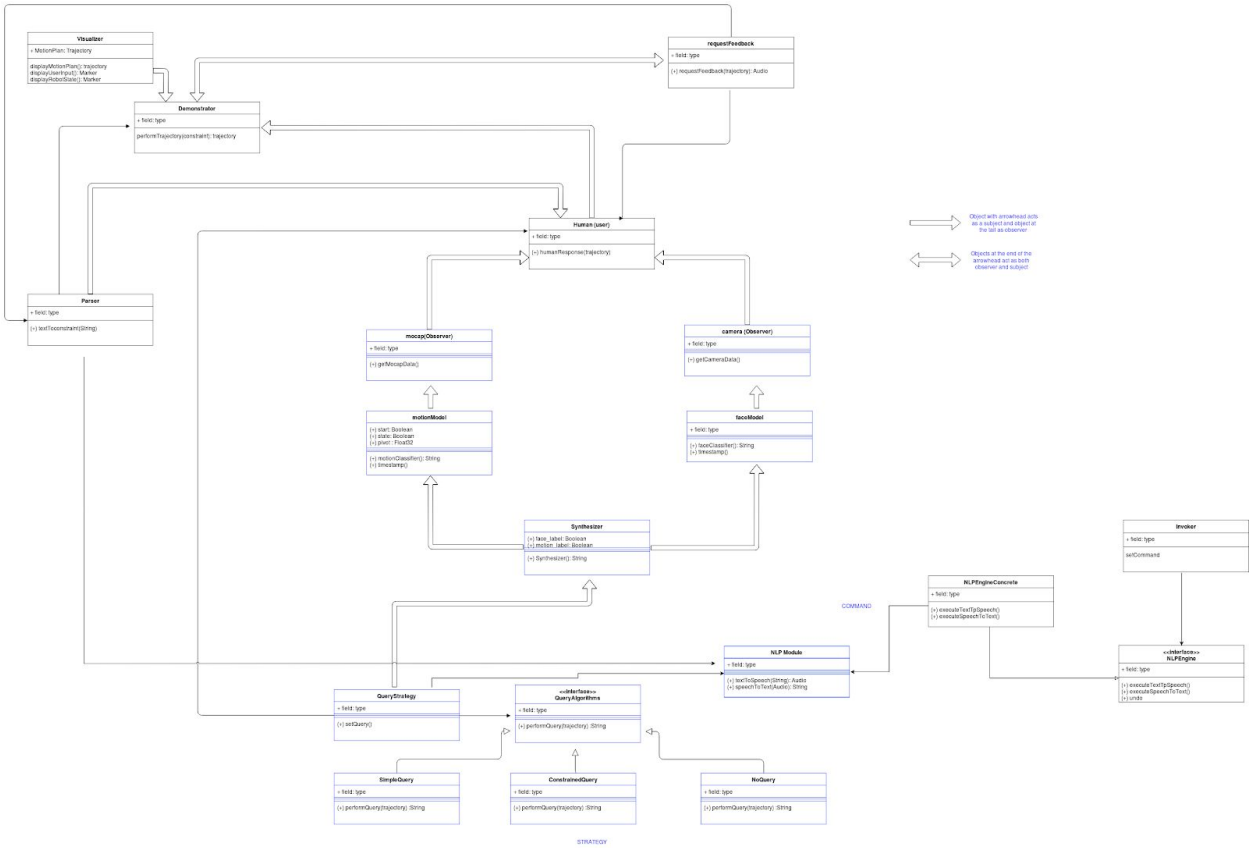

Figure 2. *Original* UML Class Diagram of System

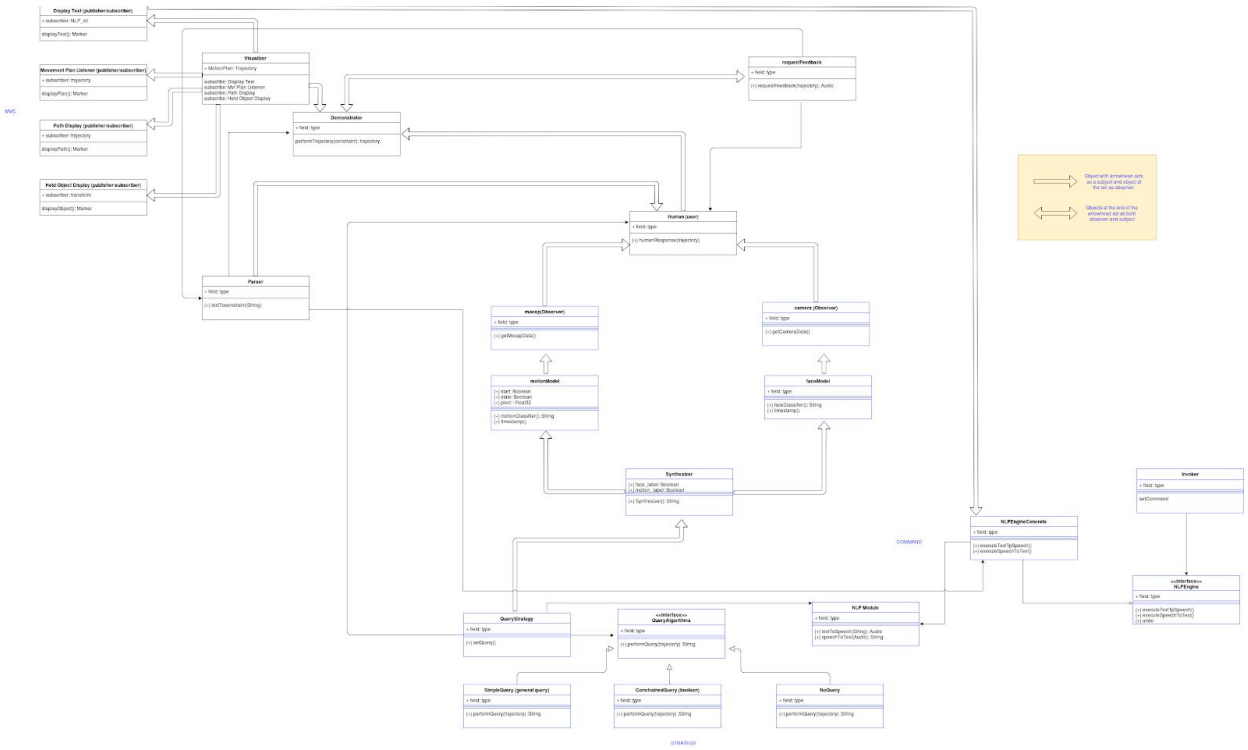

Figure 3. *Final* UML Class Diagram of System

- Patterns that were implemented are labeled in the diagram, and include Command, Model-View-Controller, and Strategy.
- The main change from the original diagram is the full addition of the visualizer elements that provide the user with a graphical visualization of what is happening with the robot and why.

Third-Party code vs. Original code Statement:

Below we enumerate the specific aspects of our code that are **third-party** or generated by people **outside** of our team.

- **CC-LfD**: A basic framework for robotic learning of correct trajectories based on demonstrations. We built on this work by giving the system the capability to map positive/negative user input to correct/incorrect portions, respectively, of an attempted trajectory.
  - https://github.com/cairo-robotics/cairo-lfd
- **Robot Operating System (ROS)**: Framework for building robot perception, planning, and control components in a modular and shareable fashion.
  - http://wiki.ros.org/
- **PyKDL**: ROS package for computing the forward kinematics (joint angles to end position) using a geometric description of a robot arm as input.

    - http://docs.ros.org/diamondback/api/kdl/html/python/index.html
- **Google Speech-to-Text**: Cloud service for converting speech audio into text data.
    - https://cloud.google.com/speech-to-text/
- **RViz**: A 3D visualization tool for ROS.
    - http://wiki.ros.org/rviz

Statement on the OOAD process for your overall Semester Project:

One design issue that we encountered was **coupling** between certain portions of the code. While in general the project is relatively loosely coupled and we were able to work independently on our contributions, at one point we were waiting on a third party to complete their work so that we could continue with our own testing and implementation. The Concept-Constrained Learning from Demonstration (CCLfD) external system with which our project interfaces is currently a work-in-progress and is in flux. So when we were at the point to test this interface, we did hit a temporary block. A takeaway from this situation is to try to have a usable version of third-party software always ready for testing, even if it has been deprecated or is out-of-date.

Another design element that was evident during the course of our work was **readability** of the code. Because we were starting from an existing program, all members of the team except for one had to get up to speed on the code base in order to contribute to it meaningfully. Fortunately, the code was very readable and understandable, which helped everyone to contribute to a **cohesive** project. What we learned from this is to provide solid documentation throughout the development process and to continue to contribute code that is easily readable, so that future contributors are afforded the same ease of use.

The final design process issue that we will discuss is our use of Docker to aid development. This process required us to maintain two parallel tracks of **version control** to manage the project. Docker is an application for running a contained computing environment, called a *container*, on a host machine. Unlike a virtual environment, an entire operating system (OS) is not run on the host OS. Instead, Docker leverages host OS resources to reduce replicated function, while presenting the user with the illusion of a completely separate system. The main advantage of using Docker is that all dependencies can be deployed with the image, and the difficulties of setup only have to be endured once. This does have a drawback in that changes to the image must be manually pulled and then rebuilt by all users of the image in order to main homogeneity across the environments.