Imitation Learning-Based Target Tracking

Using Shutter to track moving targets

Kristina Shia
Yale University
kristina.shia@yale.edu

Anushree Agrawal
Yale University
anushree.agrawal@yale.edu

ABSTRACT

In this project, we use imitation learning to teach the simulated shutter robot how to follow an object of interest that moves across its field of vision over time in a more natural, human-like way. Previously, we have done work on orienting Shutter to exactly center an object in its view; this may not be the most natural way to indicate that the object has attracted the robot's attention, especially in social interaction settings. We use machine learning models to map absolute target position and relative target position coordinates to robot joint state positions and change in joint state positions respectively. Using these models, the robot still behaves in a way that is not human-like and precise.

We then use teleoperation to generate data, in which an expert directly controls the robot with a joystick to follow a target around in a natural way. We train a machine learning model on this data and find that the imitation learning mechanism is more natural than using the previous absolute and relative inverse kinematic methods to train our models. Further work includes conducting user evaluation studies to see if there is a preference in robot target tracking using an imitation learning-based model or an inverse kinematics-based one. Additionally, extending the system to use the real robot in a room to track human movement in its field of vision could be used to test if the imitation learning method is more natural in a real life setting.

ACM Reference Format:
Kristina Shia and Anushree Agrawal. 2018. Imitation Learning-Based Target Tracking: Using Shutter to track moving targets.

1 INTRODUCTION

In our previous work on object tracking with Shutter, we've focused primarily on calculating the desired joint positions based on the transformations between Shutter and the target. In class, we've discussed imitation learning techniques that allow the system to learn policies based on the demonstrations of an expert. In this project we'll work to combine these to build a system that allows Shutter to perceive a moving object and imitate the tracking behavior of real humans.

Rather than using inverse kinematics to solve for the desired joint angles, we can opt to use machine learning techniques to approximate the underlying relationship between the position of the target and the movements of the robot. However, we also considered how we might further improve the system, not just to track a target, but to do so in a human-like manner.

While tracking a target very accurately may be desirable for many use cases, moving a robot to exactly follow a human subject might be less than ideal for an interaction where the robot is trying to seem personable or human-like. When humans observe a moving object, they might occasionally move to keep the object in their field of view. However, they are unlikely to constantly move at the same pace as the object to keep the object perfectly centered in their view. In fact, this behavior may seem unsettling and too precise.

In this project, we worked to understand the challenges and limitations of building a prototype of a robot that will track a target. We examine the potential for applying machine learning techniques to model the desired movements and also mimic the behavior of a human expert.

2 RELATED WORK

Our approach of using machine learning to imitate the behavior of humans is inspired by work to train machine learning models to control robots to identify and interact with humans in social settings, like at a bar [Keizer et al. 2014]. Previously, studies have also attempted to have robots in these social settings act more human-like. Notably, a study used a model that codified a human storyteller's gaze while reading a story, and had a robot follow this handcoded model [Mutlu et al. 2006]. They found that humans were more engaged and remembered the story better when the robot imitated human-like gaze patterns, which indicates that natural gaze patterns is important for successful applications in human computer interaction. We would expect similar results for similar situations, like a bartender or a teacher.

Many frameworks and background knowledge for using imitation learning to teach robots movements comes from Argall et al. 2008. We used their data generation setting of teleoperation for our work, in which a robot is directly controlled by an expert [Argall et al. 2008]. Machine learning for teleoperation has been attempted with various models and settings, including Hidden Markov Models to determine actions for a space station robot [Yang et al. 1994].

Methods for ensuring that expert data is reliable have been studied extensively. Expert use of a joystick controller to control a robot and its camera in teleoperation has been shown to have the best performance, compared to other forms of control like tracking head movements, especially when the user is experienced with using a joystick [Doisy et al. 2017].

Imitation learning has some problems, notably when the robot begins making small errors that eventually leave it in a state that the human expert has not given it in the training data. The DAgger algorithm, an imitation learning algorithm, has been demonstrated to help with this problem, and is an important consideration for our work [Ross et al. 2011].

The perception and image segmentation aspects of our project are mostly based on previous class assignments. Our project uses the Shutter robot which has 5 links and 4 revolute joints. The

, ,
2018.

manipulation of joint angles and use of the camera will be similar to previous assignments. We will focus on moving joint 1 and joint 3 of the robot to adjust yaw and pitch.

3 METHOD

We've divided our approach to building this system into a few components–perception, generating training data, and training.

3.1 Perception

The input to our system relies on perceiving the location of the target. We took two approaches to representing the target location: absolute and relative positioning.

3.1.1 Absolute Positioning. In our initial iteration, our perception was based on problem set 1. We wanted to obtain the position of the target moving in space as problem set 1 allows us to. In this case, we are simply using the absolute 3D position of the target. We obtain the 3D coordinates from the `target` topic to which `generate_target.py` publishes. We use this 3D coordinate to determine how to orient the robot's joints. In this case, there should be no noise in our observations, since our system knows the absolute coordinates of the target.

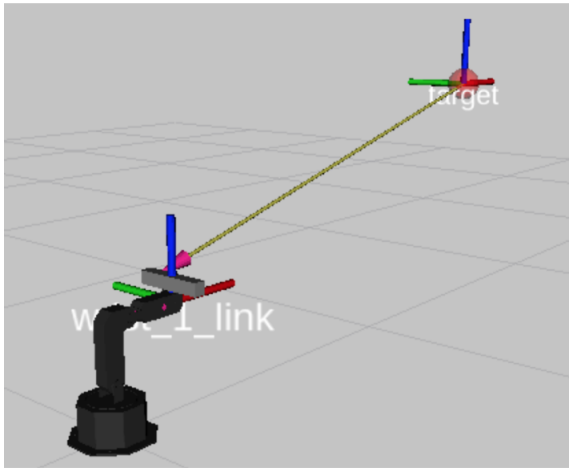

Figure 1. Absolute positioning based on target x, y, z coordinates.

While the lack of noise and comprehensive view of the environment is desirable for tracking the exact position of the target, we chose to move to use relative positions as inputs to allow the robot to operate in a more realistic environment. Outside of simulation, the only way the robot could obtain its 3D coordinates in the real world would be to have the room calibrated with sensors. In contrast, the relative positioning of the target can be obtained from the robot's camera alone, allowing significantly more flexibility in the types of environments in which the robot can operate.

3.1.2 Relative Positioning. In a later iteration, we used the virtual camera to obtain the x and y positions of the target relative to the robot. With this approach, the robot only perceives the target as an x, y coordinate in an image frame. It has no concept of the actual 3D location of the target. Instead, it only knows the position of the target as compared with its camera center. The data is formatted as the distance from the center of the camera along the x and y axis. Our virtual camera image was 640 px by 480 px.

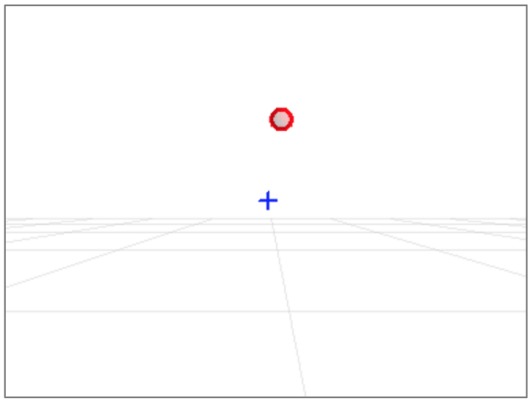

Figure 2. Relative positioning based on distance from center in an image frame.

When we use the relative position from the virtual camera, we will be mapping our observations to changes in joint states, rather than absolute joint states.For example the target being to the left of the camera center should result in a change in joint state that shifts the camera to the left. After this correction is made, the target should again be in the center of the image frame.

3.2 Training Data Generation

3.2.1 Automatically Generated. To generate training data for the non-expert cases (absolute and relative without expert), we ran a simulation of the shutter robot moving in response to the ball moving. We created a script to automatically run different episodes, where each 60 second episode was the ball at a certain x starting, y starting, and z starting position, and a different radius. The x starting position is how close or far the ball is from the robot. The ball would either move in a horizontal or vertical fashion for our training data. We ended up training our absolute and relative without expert models on 12 minutes of data.

For the absolute case, we generated data that mapped the absolute x, y and z positions of the ball to the absolute values of joint state 1 and joint state 3, using inverse kinematics. For the relative no expert case, we generated data that mapped the relative x and y positions (distance from the center of the image in the x and y directions) of the ball to the change in joint state 1 and joint state 3. The desired joint states of the robot in this case was still determined by the inverse kinematics, but the data that we recorded was based on difference between the last joint state and the new desired joint state for a particular sample.

3.2.2 Expert. In order to mimic how a human might track a moving target, we had a human expert teleoperate the Shutter robot. To obtain examples of an expert operating the robot to track the target, we had a single expert move the robot using a joystick. We added the node `ps2_controller_input.py` which processed and scaled input from a PlayStation controller joystick and converted the inputs into changes in joint state. With this setup, we could move the robot freely using the controller.

We collected data from the expert operating the robot to follow the target when the target in two long episodes: one with the target moving horizontally across the screen and one with the target moving vertically. We also allowed the expert to practice operating the robot in a few episodes before actually recording the data since it was difficult to move the robot smoothly without jerky movements at first. We opted to avoid training on the circle trajectory since operating the joystick to smoothly circular trajectory was difficult for the expert.

The expert also needed to avoid anticipating the movement of the target. Since the expert may observe the pattern of the movement of the target (ex. moving left and right in intervals), the expert may have been able to operate the robot to anticipate the target location. However, the expert specifically had to avoid this behavior. To elaborate, if the target is moving to the left so it appears to the left of the camera center, we would expect the expert to move the robot left. However, if the expert anticipates that the target will continue to move left, the expert may overcompensate and move the robot farther to the left. Then the target will appear to the right of the camera center as the robot continues to move left. This behavior will contradict the expected behavior where a target appearing on the right will result in the robot moving right. As a result, we attempted to limit this anticipatory behavior when recording expert data. However, the expert was otherwise able to operate the robot freely, so the expert could allow the target to move farther from the camera center before "catching up" to it.

We ended up training the expert model on 4000 samples, or about 2 minutes of data, for each of the vertical and horizontal trajectories.

3.3 Training

Our generated data and expert data provided us with samples correlating target position to joint states or joint state changes. We fit a few simple scikit learn models, and then an off-the-shelf scikit learn neural network to our training data. The choices about model selection, outlier removal, and separation between x and y models are discussed later. After training the model, we replaced current implementation of `look_at_target.py`. Using the real-time input data (absolute or relative positioning), we loaded our trained models and tried to predict the ideal position of the robot's `joint_1` and `joint_3`. We could then observe the motion of the robot as the target moved with different trajectories.

4 EXPERIMENTS

To run our experiments, we used the generated training data to train a model. Our model then was used instead of inverse kinematics to figure out where the robot should move. In the absolute position case, we were passing in absolute x, y and z positions to the model and received absolute joint state 1 and 3 positions. In the relative position cases, we passed in the relative x position to one model and received the joint state 1 difference, and the relative y position to another model and received the joint state 3 difference. We used the output of our model to move the robot, and then we passed the new position data to the model again. The results of our experiments are clearly identified in the supplementary videos.

4.1 Absolute Performance

We used a `MLPRegressor` neural network for our absolute positions, and found that the robot followed the target exactly, as expected. The model performed well, with a low loss rate.

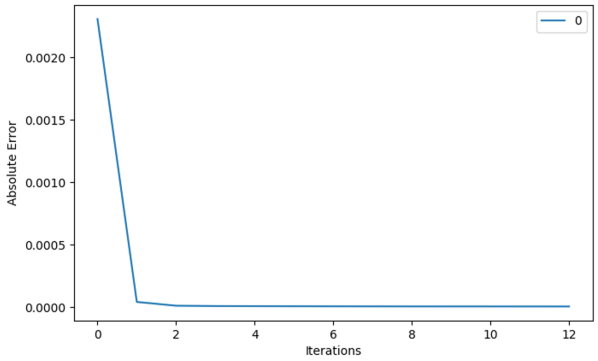

Figure 1. Loss curve for MLPRegressor trained on target x,y,z positions

We could observe the robot moving to track the target in RViz very smoothly and accurately using the joint angles predicted by our model. This served as a good demonstration of our complete end-to-end integration of our system, though the performance is better than we expect to achieve with later iterations, since in this case, there is no noise in our observations.

4.2 Relative without Expert Performance

We started off training the relative without expert data using the same MLPRegressor Model. We found that the robot was not following the target, and would move around jerkily, ending up in extreme position states. We then switched to using a LinearRegressor, which is explained more in the Discussion section. We used a separate model for figuring out the x coordinate and the y coordinate. We first only ran episodes for the horizontal case (about 12 minutes) and then trained a model to only discover the x position with an 80/20 split training and test data, and then we ran episodes for the vertical case (about 12 minutes) and trained a model to discover only the y position with an 80/20 split as well. We then tested our robot's responses with horizontal, vertical, and new circle ball movements.

With our two model setup, the robot was able to follow the target with not much human-like lag on new circle movements, and horizontal and vertical movements, which is what we expected.

4.3 Relative with Expert Performance

We trained two LinearRegressor models with expert generated data– one for target x positions, and one for target y positions, as in the relative without expert case. We had similar horizontal only and vertical only data that we trained each of our models on. We only had around 4000 samples of data for each case, which is much less than the non-expert case. We then ran our robot using the trained horizontal model for horizontal target movements, vertical model for vertical target movements, and both models for the generalized circle data. In all cases, we noticed that the robot was not following

the target exactly. There was some human-like lag, which agrees with our prediction. The robot did however move in a smooth fashion that indicated that it was following the target, and there was no jerky movement or movement that was completely incorrect. These results are promising, especially since we trained with a lot less data than the non-expert cases.

5 DISCUSSION

5.1 Model Selection

For our relative without expert method, we originally used a MLPRegressor neural net model, like we used with the absolute position data. However, we found that this model was giving us robot movements that were extremely jerky and incorrect. We graphed the correlations between x positions and joint state 1 positions for both the absolute (horizontal, vertical and circle movements) and relative (just horizontal movements) position data. We also graphed the changes in joint state and x positions over time.

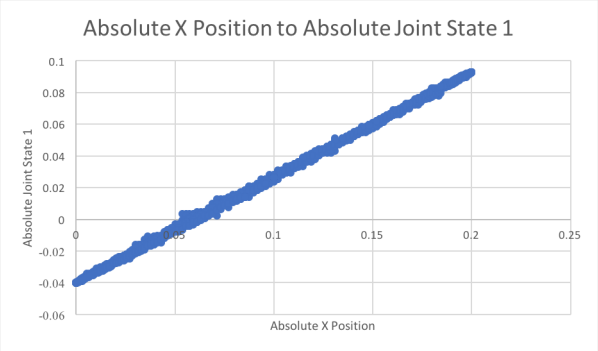

Figure 2. Correlation for absolute method

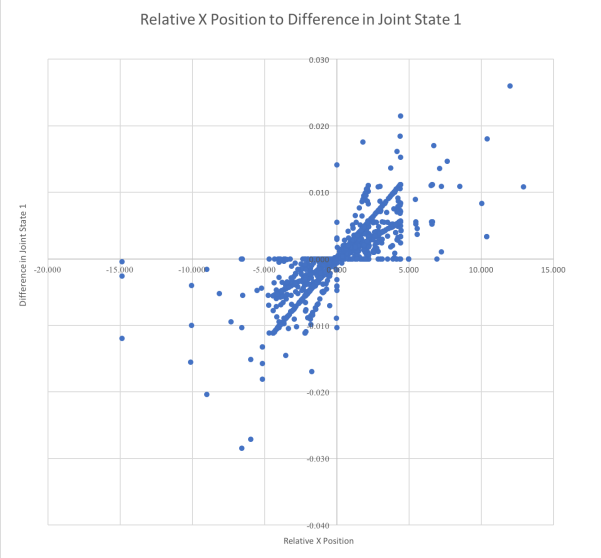

Figure 3. Correlation for relative non-expert method

As Figure 3 shows, the correlation for the relative non-expert data is much more noisy than the correlation for the absolute data in Figure 2, because using relative positions is not as exact as absolute positions. There is still an underlying linear correlation that we want our models to find. The MLPRegressor model was overfitting to this noise, which was the reason for the jerky movements. In the absolute position case, the overfitting did not matter because the correlation was not noisy. Therefore, when we switched to using the LinearRegressor for the relative data, because we were not overfitting to all of the noise, the robot actually followed the target instead of moving erratically.

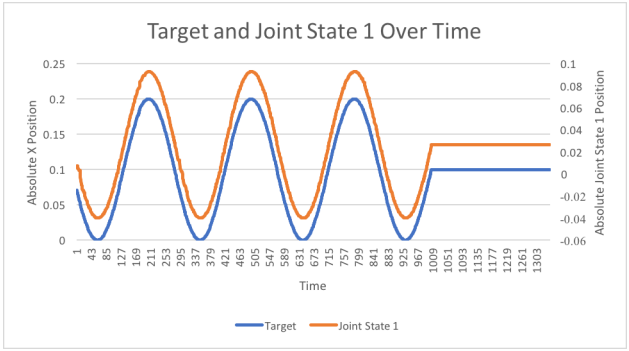

Figure 4. Change over time for absolute method

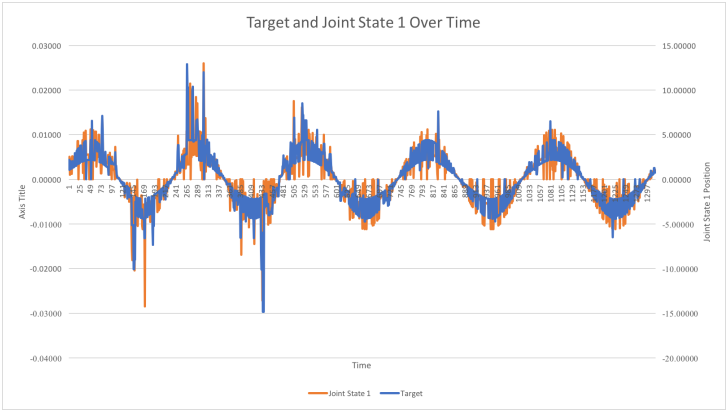

Figure 5. Change over time for relative non-expert method

The change over time in Figure 5 also indicates this phenomena, where the changes in joint state position compared to the changes in the target position are still following the general pattern that was generated, but in the relative case, there are more spikes and more noise throughout.

When we graphed correlation for our expert data, we noticed there was even greater amounts of noise than the non-expert data case.

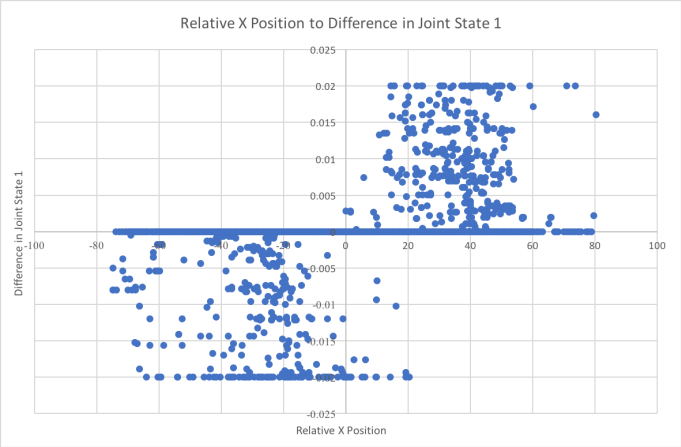

Figure 6. Correlation for relative expert method

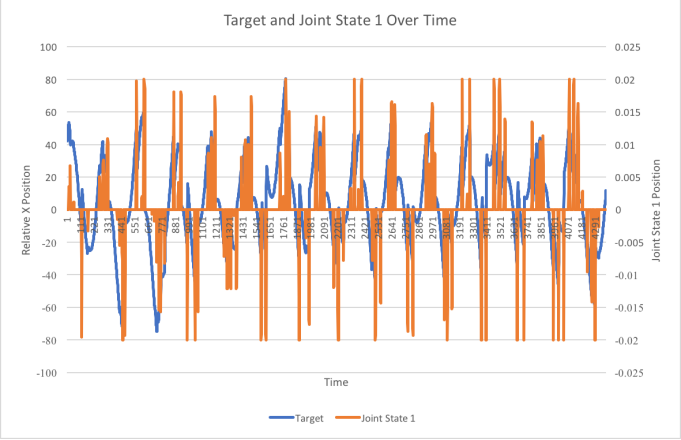

Figure 7. Change over time for relative expert method

In the expert case, Figure 6, with the solid line at the x-axis, shows that there was more of a bias to not move, or stay at 0 change. This is because the expert moves the joystick in smooth movements, with large movement at once, and not small movements at each time point. However, there was still a generally linear correlation in this case, with some outliers. We used the LinearRegressor again to identify this generally linear underlying pattern. The change over time is even less clean here because of how the expert moves the joystick, with general movements that do not necessarily follow the periodicity of the actual ball moving.

5.2 Outlier Removal

When examining the training data, we noticed that some of the x, y inputs exceeded the frame of the image. Since the virtual camera image is 640 px by 480 px, we expect our input data to range from -320 to +320 for x and -240 to +240 for y. However, when the robot initially starts up, the target is not initially in its field of view. As a result, the initial input values exceed this range. This would not be possible in the case of perceiving from actual camera input, but these values still occur because the virtual camera is based on transformations. The samples collected when the robot starts up aren't particularly relevant for our target tracking application, so we remove these outliers. The issue of the target starting outside of the field of view is solved by locking onto the target for the initial position. This is discussed in the Limitations section below.

In the case of the expert data, we only begin recording data after the expert has started to track the target, so no outlier removal is performed on expert training data.

5.3 Separate X and Y Models

We used a separate x model that took in relative x positions that outputted joint state 1 changes, and a y model that took in relative y positions that outputted joint state 3 positions. For the circle data, we used both models. This is because our LinearRegressor model was not sophisticated enough to realize that in horizontal training data, y positions were constant, and in vertical training data, x positions were constant. When we used one model trained on both horizontal and vertical data, the outputs would cause the robot to swing and end up in states it had not seen before. Then, when we used a model trained on x and y data in only the horizontal case to provide both joint state 1 and joint state 3 positions for the horizontal case, the LinearRegressor would attempt to find patterns in the y position and change that too, which would cause the robot to move jerkily into extreme states. Then, when we used a model trained on only x data to provide joint state 1 and a model trained on only y data to provide joint state 3 positions, we ended up seeing the result we expected.

5.4 Limitations

5.4.1 Target Moving Off-Screen. Our current models do not handle when the ball is no longer in the virtual camera view of the robot. To cover this case, we would need to train on more data that has this happen. The expert in this case would respond by moving the robot in the direction that it was moving before the ball had gone off-screen.

5.4.2 Locking on Target. When we began our robot simulation, we always started by using the absolute model to allow our robot to lock onto the target. This is because the first movement that the robot makes is a "wake-up" movement, which is exaggerated and not in line with the rest of the movements that the robot makes. In cases where this absolute model or absolute positioning is not available, we would not be able to do this original lock-on, and we would have problems where the robot would end up in a state that it has not seen before. To fix against this, we could use something like DAgger which is able to recover from states that have not been seen before, or we could train on more data with this wake-up state. [Ross et al. 2011]

5.4.3 Separate X and Y Models. Having separate models requires that joint states that control these positions are independent. In another robot case where these are not independently controlled, the separate model case would not work. Additionally, while having separate x and y models is more data efficient, it does require precise horizontal and vertical movements, which would be difficult to emulate in a real life situation. If z position was also going to be changed, this would require a third model and a third iteration of training and testing a model.

6 FUTURE WORK

6.1 Perception

Our relative positioning model currently receives input from the virtual camera. The virtual camera relies on the transformation between the robot and the target to project the target into the image. As future work, we could work on segmenting the target (in this case the red ball) from images seen by the camera of the robot. We would focus on identifying a 2D coordinate of the target in the camera image. This would become our input data. We predict that the performance of the model will be similar. There may be more noise introduced from the image segmentation, but the x, y coordinates of the center of the target in the image should still hold a similar correlation with the desired joint state changes. Initially we will focus on the 2D coordinate, but we could also consider if obtaining depth from the image would improve overall performance of the tracking, especially with more complicated movement trajectories.

Eventually, this system could ideally be translated from simulation to the real robot. We could use real image data from a red block moving in space like problem set 2, which will replace the simulated red ball as the target. In this case, the noise from the input data will be more substantial since the target will be significantly more difficult to differentiate from the background in real world images, rather than the simulated space. Future work could explore using a moving person as the target, walking back and forth in front of the robot. The coordinates of person could be obtained from a real video feed using an external library. In this application, the robot's human-like behavior would be most useful.

6.2 Expert Data Collection

In this case, our expert teleoperates the robot via a Playstation gamepad. However, the way the expert controls the robot via a joystick may not entirely capture the way a human would observe a moving target. The joystick teleoperation also requires the expert to make precise movements. In the future, we could consider collecting the expert data by tracking the movement of a human expert. We could potentially use a virtual reality headset to display a target moving in 3D space and then record the corresponding movements of the expert. Doisy et al. 2017 mentions that current head tracking mechanisms have poorer performance for accurate robot control than joysticks when controlled by experts, but if this can be improved then head tracking, and potentially even gaze tracking, would be more precise for modeling human behavior.

6.3 User Evaluations

Since the goal of this project was to produce human-like target tracking behavior in a robotic system with the purpose of making the human-robot interaction more comfortable, we would ideally be able to evaluate the complete system with human participants. If we implemented the full system on a the real robot, we could have human participants move around a room and have the robot move to track their motion. We could have some interactions where the robot is moving using a model trained with relative positioning and no expert. In this case, we'd expect the robot to follow the participant precisely. Then, we could have other interactions where the robot is moving based on predictions from a model trained on expert data. We'd expect the robot to have more human-like movement patterns.

We could evaluate the robot's performance in terms of keeping the person in its frame of view. More importantly, after the participants engage with the robot, we could have them fill out a questionnaire to determine if they could tell the difference between the robots movement pattern, if they felt more comfortable with one pattern or the other, and if they had a preference between the movement patterns. This would allow us evaluate both the performance of the expert trained model and the validity of our intuition that a human-trained model would yield behavior that makes human-robot interactions more comfortable.

7 CONCLUSION

Through building this prototype, we found that imitation learning was an effective technique to use for training a robot's target tracking behavior. Imitation learning was data efficient, requiring relatively little data from an expert. Furthermore, the resulting behavior of the robot exhibited attributes that were reminiscent of the human expert's operation patterns. Rather than constantly keeping the target centered in its view, the robot was able to loosely follow the target in a relatively natural way.

We encountered interesting challenges with outliers in the dataset and edge cases with starting positions that are representative of some of the problems that would be need to be overcome when building a similar system for real-world applications. Though the system performs fairly well under simulated and controlled conditions, noisy perception and uncertainty in the target position would need to be accounted for to build a robust system. In addition, some problems, such as initializing the position of the robot when the target is out it's field of view will need to be solved in the real-world.

Overall imitation learning seems like a practical approach for solving human-computer interaction problems to build more human-like robotic systems.

8 REFERENCES

(1) Keizer, S. et al. 2014. Machine Learning for Social Multiparty Human–Robot Interaction. ACM Transactions on Interactive Intelligent Systems. 4, 3 (Nov. 2014), 1-32. DOI: https://doi.org/10.1145/2600021.

(2) Ross, S., Gordon, G., and Bagnell, A. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. Journal of Machine Learning Research, (Mar. 2011), 15:627 - 635.

(3) Argall, B.D. et al. 2009. A survey of robot learning from demonstration. Robotics and Autonomous Systems. 57, 5 (May 2009), 469 - 483. DOI: https://doi.org/10.1016/j.robot.2008.10.024.

(4) Doisy, G. et al. 2017. Comparison of three different techniques for camera and motion control of a teleoperated robot. Applied Ergonomics. 58, (Jan. 2017), 527-534. DOI: https://doi.org/10.1016/j.apergo.2016.05.001.

(5) Jie Yang et al. 1994. Hidden Markov model approach to skill learning and its application to telerobotics. IEEE Transactions on Robotics and Automation. 10, 5 (1994), 621-631. DOI: https://doi.org/10.1109/70.326567.

(6) Mutlu, B. et al. 2006. A Storytelling Robot: Modeling and Evaluation of Human-like Gaze Behavior. 2006 6th IEEE-RAS International Conference on Humanoid Robots (Dec. 2006).