NITHILASARAVANAN KUPPAN
260905444
nithilasaravana.kuppan@mail.mcgill.ca

ECSE 683: Assignment 2
Writing Characters

November 15, 2020

1 INTRODUCTION

The objective of this assignment was to generate a set of trajectories using imitation learning. The task was to reproduce at least 3 characters while staying within the robot's practical limits. From a multitude of possible solutions, this document will illustrate one such solution that utilizes a powerful regression technique to successfully complete the aforementioned task.

For this solution, *PyBullet* was used which is a Python module dedicated towards robotic simulation; *Kuka IIWA* was the chosen robot for this process. The detailed procedure and algorithms will be discussed in the later sections but learning of the parameters was done via **Imitation Learning**, specifically using a regression algorithm called *Random Forests*. Inverse Kinematics control was applied to the end effector to effectively track the path generated by the algorithm.

Since *Kuka IIWA* has 7 joints, the configuration space can be represented as $\mathbb{R}^7$ and the Degree of Freedom as **7**. The task space is produced by the end effector of the Kuka robot which is $\mathbb{R}^3$ with **3** Degrees of Freedom.

The presented solution uses an Inverse Kinematics control which takes in the predicted velocities of the end effector as the input and outputs all the corresponding joint velocities and then these are fed into a *setJointMotorControl2* function *(leveraging velocity control)* that converts it into proper torques that move the different links of the arm according to the velocities.

Since the script does take position as an input and the model outputs predicted velocities, it would be logical to conclude that the **state** is the **position** of the end effector and **action** is the **output velocity** i.e., the *end effector velocity*, which is in turn used to calculate the joint velocities using pseudo inverse of the Jacobian.

The following section provides an incisive look into the method used to solve this problem.

2 METHOD

Essentially, to solve this problem, the two following steps were performed

1. Finding the velocity required to track the path of a character
2. Using this velocity effectively on the robot

2.1 *Imitation Learning as a Regression Problem*

The first step was to collect or obtain trajectory data for all these characters, specifically all the English alphabets. Such a set of characters were borrowed from the source codes of the python library *PbDlib*, which is a package for robot programming by demonstration *(learning from demonstration)*. This data has been stored in the repository under *data/2Dletters* - this folder has all the *.mat* files containing trajectory information *(position and velocity)* required for tracking alphabets.

After leveraging *scipy* to convert *.mat* files into python readable files *(nested lists of positions and their corresponding velocities)* the data was split into train and test. From n sets of trajectories, $n-1$ were used for training and one was used to test / predict the velocities. The presented solution looks at this problem like a regression task - the *position* columns can be taken as **features** and their corresponding velocities as **target variables**.

This regression task was solved using *Random Forests*. *Random forest* is a supervised learning algorithm which uses ensemble learning method *(technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions)* for classification and regression. The algorithm constructs multiple decision trees using the training data and churning out the mean prediction of those trees. Python's *scikit-learn* library provides a set of easy to use functions to code this algorithm and this solution leverages the same. After a considerable amount of experimenting with the model, a set of hyperparameters were decided upon and finalized for this script *(exact values of the parameters are given in the experiments section)*.

2.2 *Predicted Velocity based Robot Tracking*

Before using this predicted velocity as an input for the robot, slight modifications need to be made to make it more conducive for a practical simulation environment -

1. The trajectory data that was used for modelling was collected on a flat 2D surface, from a tablet. And hence, it is important to scale those velocities for them to work efficiently in a 3D environment. Therefore, it is of utmost importance to scale the velocity to successfully cater writing *multiple* characters while not violating the physical limits of the robot - This was achieved simply by scaling the output by a constant
2. Writing characters is a 2D task, but since this solution simulates a 3D robot arm in Python *(specifically, Kuka IIWA robot)* a third dimension must be added *(with a value of 0 since the robot only needs to move in 2 dimensions)*

Once these velocities *(which are technically the end effector velocities)* were obtained, the corresponding joint velocities were calculated using pseudo inverse of their respective Jacobians and were then supplied to the robot. The base code for robot simulation was taken from the *previous assignment submission*, which in turn referred to Prof. Hsiu-Chin Lin's *Course Example Repo*.

3 EXPERIMENTS

3.1 Model Parameters

After multiple rounds of evaluating different types of non linear regression techniques, *Random Forests* was finalized with the following parameters -

Parameter Name	Short Explanation	Parameter Value
n_estimators	*# trees*	1000
criterion	*func. to measure split quality*	*mse*
oob_score	*toggle out-of-bag sampling*	*True*

3.2 Character Selection

As mentioned in the last section, the velocities were scaled down by a factor of 40. This was done to bring the velocities to the robot's limit *(a function was also created to check whether the velocities are in range for every control cycle)* and to enable writing multiple characters, all while being in the range of the robot.

While testing out trajectory data for different characters, it was seen that characters that can be written in a single stroke *(or characters that only have one possible direction to proceed in, at every point)* work flawlessly while complex characters that cannot be written in one stroke perform bad. Two metrics were calculated on the trajectory data for all the characters that were tested - *mean squared error* & *R2 score*. An *R2 score* of 1 would mean that the predicted velocity perfectly overlays the test velocity and a negative score would mean that the predicted values are useless.

It was seen that single stroke characters like C, D, G, L, N, O, etc all have a good R2 score of 0.75 or above. The non linear regression algorithm successfully predicted the velocities corresponding to the input positions, engendering near perfect paths for the robot to follow. Even some complex characters like M, A, W had a decent score of 0.5 or above and the robot was able to *imperfectly* reciprocate the intended shape. However, multi stroke characters like H, X, Y, K all had very less scores, possibly even negative scores and the robot could not follow the output velocities to track these characters.

Following much testing and experimenting, the letters in the word **'GROWL'** was decided to be target letters for the robot to *write*. The following images are screenshots of the metrics generated from the model *(these values were captured from STDOUT console)*.

```
Train MSE for G -> 2.05
Train R2 for G - > 0.99
Test MSE for G -> 6.98
Test R2 for G -> 0.88
```

```
Train MSE for O -> 1.84
Train R2 for O - > 0.99
Test MSE for O -> 5.21
Test R2 for O -> 0.9
```

```
Train MSE for R -> 2.35
Train R2 for R - > 0.99
Test MSE for R -> 9.42
Test R2 for R -> 0.84
```

```
Train MSE for W -> 3.13
Train R2 for W - > 0.97
Test MSE for W -> 12.99
Test R2 for W -> 0.55
```

```
Train MSE for L -> 1.06
Train R2 for L - > 0.99
Test MSE for L -> 4.36
Test R2 for L -> 0.86
```

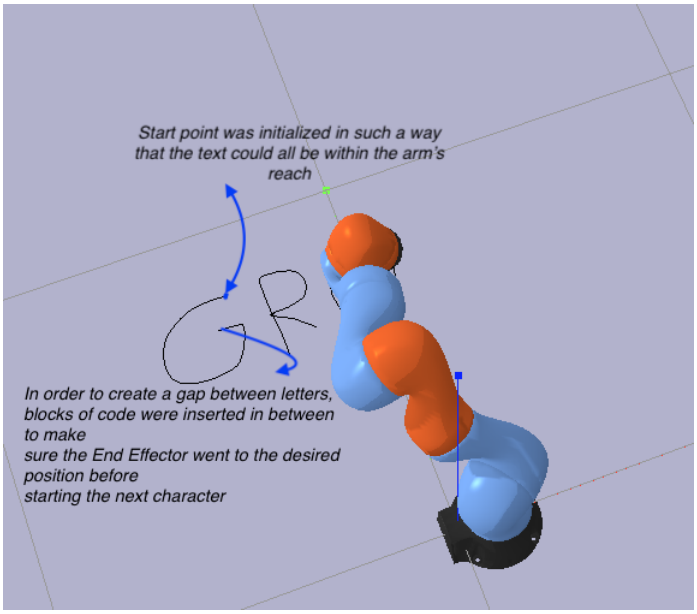
3.3 Velocity Limit Checker

A function called *velocityInRange()* was used to check whether the velocities being fed every control cycle are within the range for the robot. Since the predicted end effector velocities are assured to be in range *(because of scaling them down earlier)*, the joint velocities calculated by inverse kinematics were fed into this function. A dot (.) was output to STDOUT if it passed the limit check *(the limit in the urdf file was* ± 10 *but to be even safer, the function uses* ± 8*)* and if the check fails, an error message would be printed to STDOUT and the script would terminate. The following is the screenshot of the STDOUT console that shows this function in action

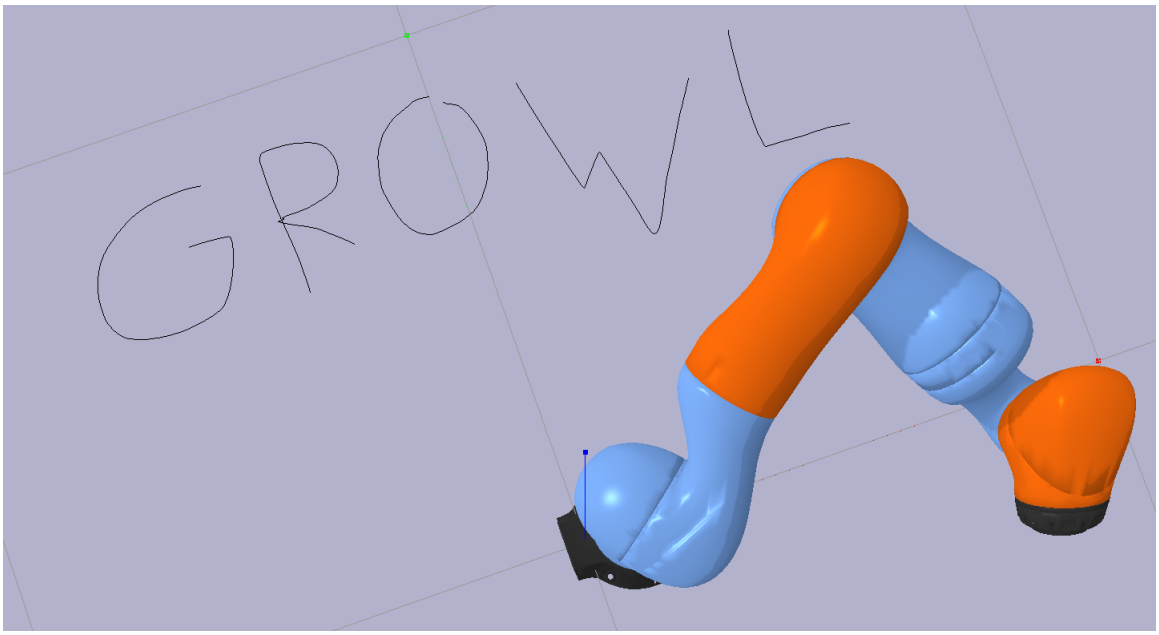
```
..........................................................
..................................
Drawn G
..........................................................
..................................
Drawn R
..........................................................
..................................
Drawn O
..........................................................
..................................
Drawn W
..........................................................
..................................
Drawn L
```

3.4 Character to Character Movement

Since the selected task involves writing the word **'GROWL'**, the arm needs to be initially positioned in a way that the whole word fits into the range of the robot - for this reason, the starting point was moved a bit farther left to [-0.3, 0.7, 0.3]. Additionally, it is important to allow some gap between each of these letters - this was achieved by setting an appropriate velocity and feeding it in between the block of code writing each of the designated letters. One such instance has been demonstrated below -



A screenshot of the final finished simulation has been pasted below. Please refer to *Demonstration_Assign2.mp4* in the repository for the complete video. It can be clearly seen that the letter 'W' is not perfect *(has an R2 score of 0.55)* but its still legible enough to be called a 'W'.



4 LIMITATIONS

This code runs well for the set of letters described in the previous sections - any letter that can be written in a single stroke and at every point has only one direction to move in. However, the script does have some limitations

1. As described above, the script fails to track trajectories corresponding to certain complex alphabets such as K, X, etc. This is because the imitation learning algorithm used, i.e., *Random Forest* was unable to predict the velocities at certain input positions where multiple strokes cross over. This maybe solved using a much better learning algorithm tailor-made for such a use-case or to break multi stroke letters into different stroke based instances which can then be used while training the model

2. The code does not *write* letters dynamically. There are separate blocks of code that write different alphabets; in order to write another letter or change the letter, the whole block needs to be modified. One way to solve this would be create a single nested list and keep appending the different velocities corresponding to the different letters *(along with the required gaps)* and then uses that as an input to the robot

3. The simulation does not employ any form of manual disturbance rejection. If the arm is suddenly moved to a new position forcibly, the end effector will keep writing instead of pausing till the original state is restored. This can be solved by employing an intelligent logic that turns *user debug line* off whenever a disturbance is detected

5 REFERENCES

1. Data from *Pignat, E. and Calinon, S. (2017). Learning adaptive dressing assistance from human demonstration. Robotics and Autonomous Systems 93, 61-75.*

2. Actual Dataset from *https://gitlab.idiap.ch/rli/pbdlib-python/-/tree/master/pbdlib/data/2Dletters*

3. Scikit-learn library *https://scikit-learn.org/stable/index.html*

4. Base code for PyBullet IK Sim: *https://github.com/McGIll-COMP-766-ECSE-683-Fall-2020/python-examples*