Imitation Learning on Manipulator Arms

AKSHAY KUMAR, DOMINIC CUPO, ONKAR TRIVEDI, AAYUSH SHAH, RISHI KHAJURIWALA

Introduction

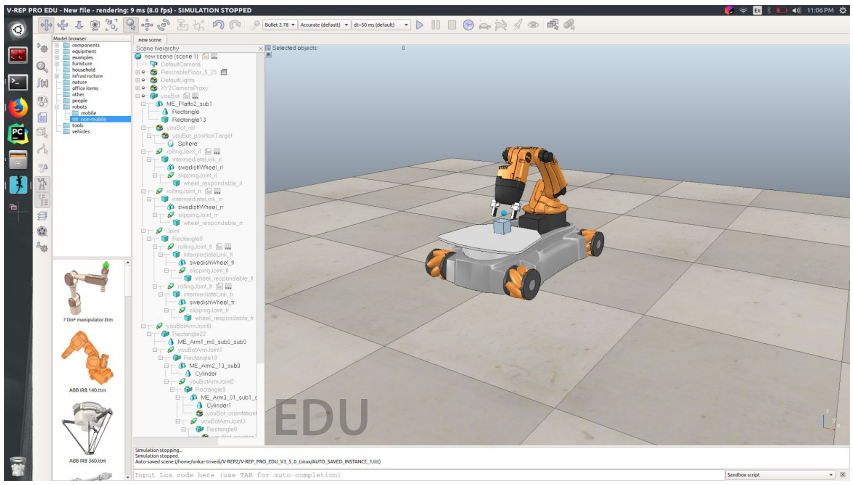
Modern day manufacturing companies are working towards a fully autonomous process with minimal human interaction. A large percentage of robots in these industries are robot manipulators for high volume products. Though full automation of some robots has been reached, programming the robots to perform these tasks require skilled engineers to interpret and codify its actions. Employing these engineers is expensive, and restrictive as you can only afford to hire a few engineers.

Programming by Demonstration is an approach that helps the user program the robot by physically showing the robot what and how the perform a task. The user does this by physically moving the robot through the desired trajectory. Apart from being extremely intuitive, one major advantage of this technique is that the nature of trajectory/curvature can be approximated by a dynamic environment without having to explicitly program the robot for every changing start and end position. You can effectively break down the movements into Dynamic Movement Primitives, allowing combinations of primitives for different movements later.

Background

For this project we focused on a teaching technique called Programming by Demonstration. In essence, this means to teach a robot what to do by showing it the movement, not coding the path. This is a popular method of programming for robots aimed at beginners or for non-technical use. The operator guides the robot through the motions and actions desired, and the robot records the motions and plays them back later. There are two main ways to teach the robot the motions desired. You can manipulate the robot via the attached Teach Pendant or similar

control. Alternatively you can physically move the robot through various positions and record the joint/end effector states continually or at waypoints.



Once you have your paths, you can either play that path back directly, or else break that path down into a series of Dynamic Movement Primitives (DMP). DMPs are a series of very basic movements, such as ‘move up’, ‘move down’, ‘rotate’, etc. The idea is to create a large amount of these blocks of motions, so you can chain them together in different combinations later. This allows you to avoid having to create an entire new path for every new action. This can be especially tedious if the robot’s actions need to have slight variations for different parts or actions. Rather than re-record the entire motion for each variation, you can keep the bulk of the movement the same, and simply change the DMPs for that specific variation.

The AI technique we decided to use is Imitation Learning. This is a form of Reinforcement Learning, also called Learning by Demonstration. This fits almost perfectly with our plan of utilizing Programming by Demonstration for our robot. Imitation Learning seeks to fix an issue with starting a new reinforcement learning session. Reinforcement learning relies on a Q Table, a table that maps certain actions to their rewards. At the beginning, the Q table is empty; the program has no idea what actions return what rewards. While you can let the algorithm iterate until it learns, this takes time, and in certain circumstances can be dangerous. Imitation learning aims to solve this issue by providing information from the start. For example, if you wished to use AI to teach a car how to avoid accidents while driving, you would save a large amount of time by providing the program with all the rules of the road. This way you can spend more time learning how to deal with people changing lanes without a turn signal, and skip trying to figure out what that red octagon means.

Setup

We first needed to choose a robot to use with our project. As one of our group members was already working with a robot for another class' project, we decided to use that one to skip the time needed to learn how to operate it. The robot used was a Kuka YouBot. This is a 5 Degree of Freedom (DOF) robot, using a 3DOF planar arm on a rotating base, with a rotating wrist joint. As our project involves following a trajectory, we can ignore the wrist joint, as we don't care about the orientation of the end effector, just the position. The robot arm is attached to a wheeled, mobile base. As we only care about the arm trajectory, we treat the robot as static and do not move it around. There is a physical robot in the CIBR lab, at 85 Prescott.

In an attempt to speed development, and so as to not be constrained to the physical robot, we opted to do our development and testing in simulation. As the focus of this project is the AI behind the movement, not the actual use of the robot, this does not affect our project negatively at all. The initial plan was to utilize the Robotic Operating System (ROS) and its simulator, Gazebo. We quickly learned that Kuka stopped supporting ROS a few years ago, and unless we opted to use an older version of ROS, we would have to find another solution. Luckily, they support another simulator called the Virtual Robot Experimentation Platform (VREP). VREP includes a plethora of helpful features, we utilized its realistic physics simulator, forward and inverse kinematics, and a Python remote API. The API is especially helpful as it allows us to continue to work in Python, as we have with all of our previous assignments. In addition, we interfaced a PS2 controller with VREP to control the robot for the demonstration part of the project.

Implementation

The AI technique implemented for the projected wasn't built upon an already existing code or project. We wrote the code from scratch. The V-REP setup was also figured and done from scratch.

The Kuka YouBot module in V-REP has its 5-DOF robotic arm controlled as linkages with set constraints. These linkages are controlled to move towards a desired end-effector cartesian coordinate with the help of a spherical virtual object existing near the middle of the arm's gripper. Controlling this sphere controls the position

and orientation of the end-effector of the robotic arm. In order to solve the IK of the robot, we can trivialize it to the solving of IK for a single sphere in mobile-frame with respect to the robot's base.

The Kuka YouBot is a model in V-rep which uses its own internal child-script, written in Lua programming language. To collect the data while "demonstrating" the robot using the PS2 controller, we needed a working forward and inverse kinematic implementation. Here are two approaches we took:

First, we modified the internal API by modifying the child script to allow it to interface with external ROS commands. Using those ROS commands, we then move the end effector in coordinate space, and record the joint angles for later analysis. In this step we made use of the inverse kinematics of the robot.

Unfortunately, when we wish to play back the motions, we require use of the external API, not the internal. When using the external API, we have to disable our Lua script from before. This disables some of the internal IK functionalities, and the IK-joint mode settings disappear. This is where second becomes useful.

In this approach we use an external API to implement the forward kinematics. This means that we control the robot via an independent Python script providing joint angle data. This data is logged and the manipulator is later trained on the data. While the external API has several advantages, including not having to modify the child script to control the robot, it does have certain disadvantages. We initially attempted to bypass these by using the first approach. The reasoning can be understood by observing how the IK functions internally in V-REP.

Through some trial-and-error, and with the help of the ext_ROS_interface package for VREP, we manage to establish connection of -REP's model with ROS. In other words, we can utilize messages over a ROS node to control the robot while making minor changes in the internal API. This enables the use of full internal functionalities as well as retaining control of the robot to external published messages. The use of this ROS interface also enabled us to easily use an external ps3-controller to control the end-effector position of the robot and generate desired trajectories for training.

Base Technique:

A standard comparison with the AI technique is a simple method. We first gather several demonstrations worth of data. Naturally, these are prone to errors in movement. In order to alleviate these, we average out the data to create an ideal series of joint angles. This is a fixed trajectory that represents the user's intent for the particular task, but does not support any kind of scaling or variations. It is non-adaptive and merely represents a new trajectory that is the best among several error-prone trajectories. The results for this non-AI technique are detailed

in the results section of the paper. However, it is quickly noticeable that there is a constant offset between the ideal trajectory and the averaged trajectory. The video in appendix A labelled 'Non AI Baseline' shows the results of implementation of this new trajectory on the robot in simulation environment.

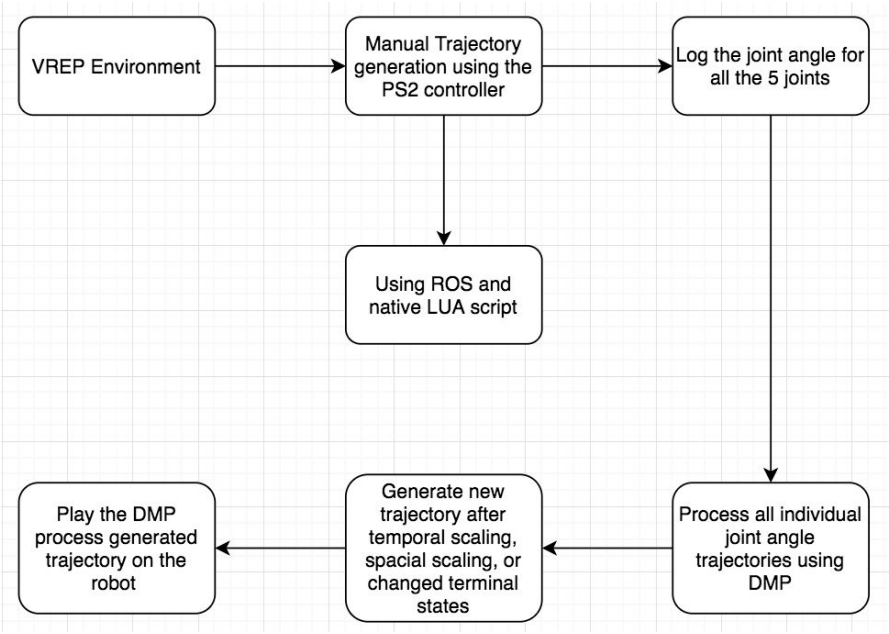
Artificial Intelligence:

We used Dynamic Movement Primitives as the AI technique for learning by demonstration. Dynamic movement primitives (DMPs) are a method of trajectory control/ planning. Imagine that you have two systems: An imaginary system where you plan trajectories, and a real system where you carry them out. When you use a DMP, what you're doing is planning a trajectory for your real system to follow. A DMP has its own set of dynamics, and by setting up your DMP properly you can get the control signal for your actual system to follow. If our DMP system is planning a path for the hand to follow, then what gets sent to the real system is the set of forces that need to be applied to the hand. It's up to the real system to take these hand forces and apply them, by converting them down to joint torques or muscle activations. IT could use something like the operation space control framework or similar control system.

Dynamic motion primitives (DMPs) are a method for trajectory control used in robotic applications. This technique utilizes a number of gaussians at various time-steps called "basis functions" and trains them on an input trajectory by factoring them as "basic movement primitives". The basis functions then, "learn" the weights of the system, modifying themselves to the input trajectory in order to convert a simple linear point-attractor system to a non-linear function by append these basis functions.

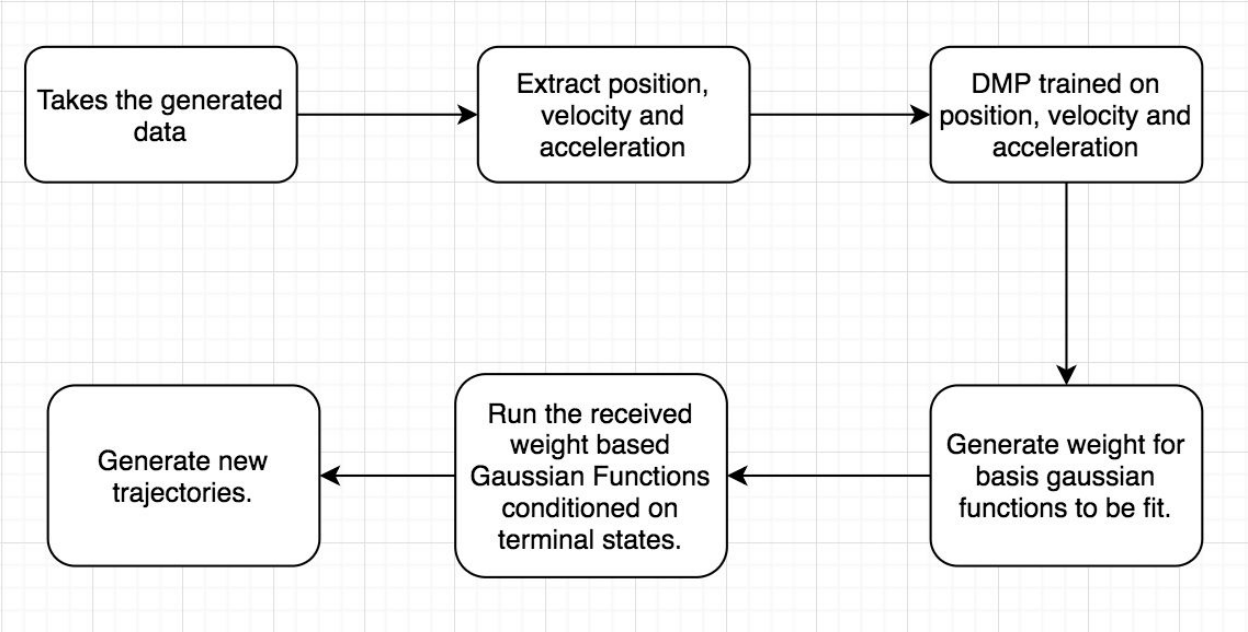
This enables us to train a system to recreate a motion with similar trajectory as the input system, while enabling us to change the end-effector position at will by changing the final DMPs invoked. We can then enable temporal and spatial scaling of the motion as per our desired trajectory, without needing multiple demonstrations of the motion. This learning technique is efficient and enables us to learn a desired motion in one-shot by breaking it into its motion primitives and training an algorithm on them.

The following flowcharts will give an good overview of the entire implementation detail of the Dynamic Movement Primitive.



The implementation of Dynamic Movement Primitives involves processing the obtained data that carries varying joint angular positions for all the individual joints of the manipulator over the complete demonstrated path. It begins with parsing several data points (each data point is a vector containing the angular position of the joints) to segregate each joint independently. After, the DMP training module runs over each of individual joint data and computes weights for the basis Gaussian functions and stores in a sample XML file. Further, depending upon the user-defined variations in the testing conditions, the DMP runner uses the weights to fit a new trajectory that has curvature/nature of motion similar to the demonstrated motion. Changing the terminal states for the joint angles or scaling the motion over a larger/shorter time period are a few user induced variations among others that would be handled by the AI technique.

The following flow chart explains what is going on behind the scenes in the Dynamic Movement Primitive algorithm:



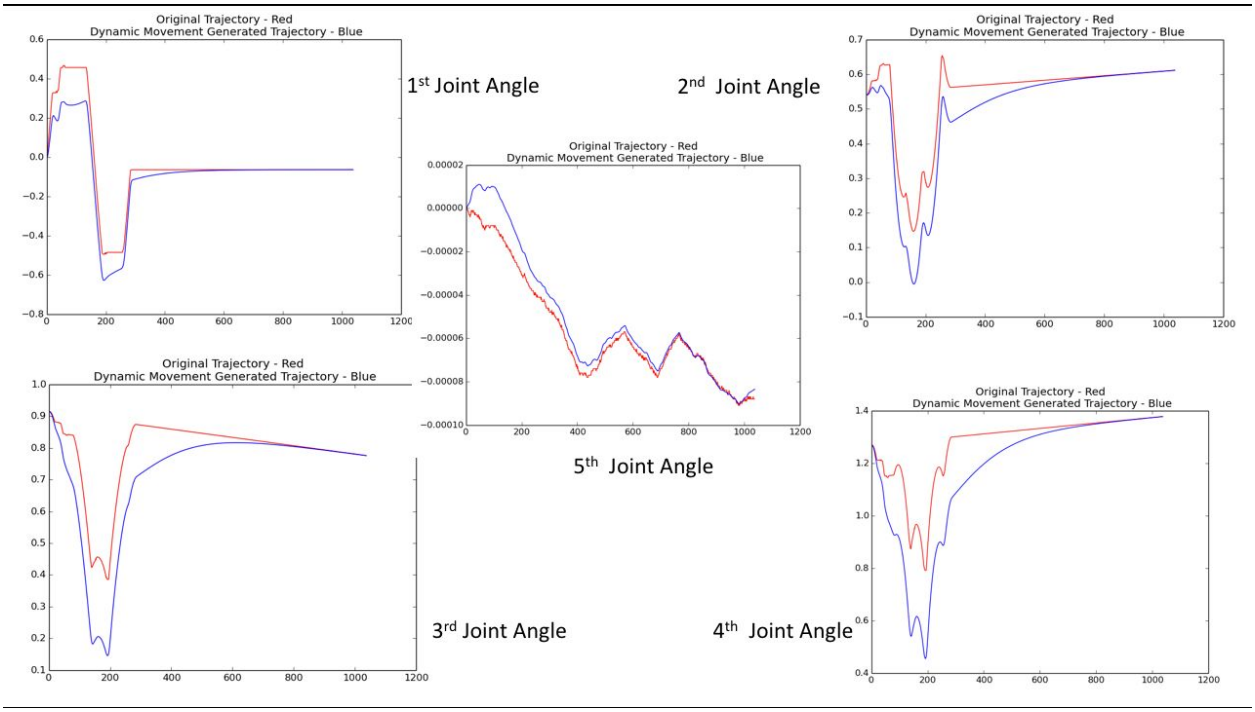
The figure shows a snippet of the DMP weights, variances and Gaussian means generated for the various Gaussian functions during training and shall be used to generate appropriate trajectories.


```
-<DMPs>
  -<Weights>
     <w>-8.812409</w>
     <w>-8.157689</w>
     <w>-6.407151</w>
     <w>-6.800521</w>
     <w>-7.196297</w>
     <w>-7.741294</w>
     <w>-8.009500</w>
     <w>-8.313758</w>
     <w>-8.666910</w>
  -<inv_sq_var>
     <D>4576623.981156</D>
     <D>547723.731401</D>
     <D>213977.391607</D>
     <D>118598.127187</D>
     <D>77966.645107</D>
     <D>56728.120804</D>
     <D>44149.381069</D>
     <D>36047.815556</D>
     <D>30509.029222</D>
  -<gauss_means>
     <c>1.000000</c>
     <c>0.999150</c>
     <c>0.996693</c>
     <c>0.992763</c>
     <c>0.987483</c>
     <c>0.980972</c>
     <c>0.973338</c>
     <c>0.964685</c>
     <c>0.955109</c>
```

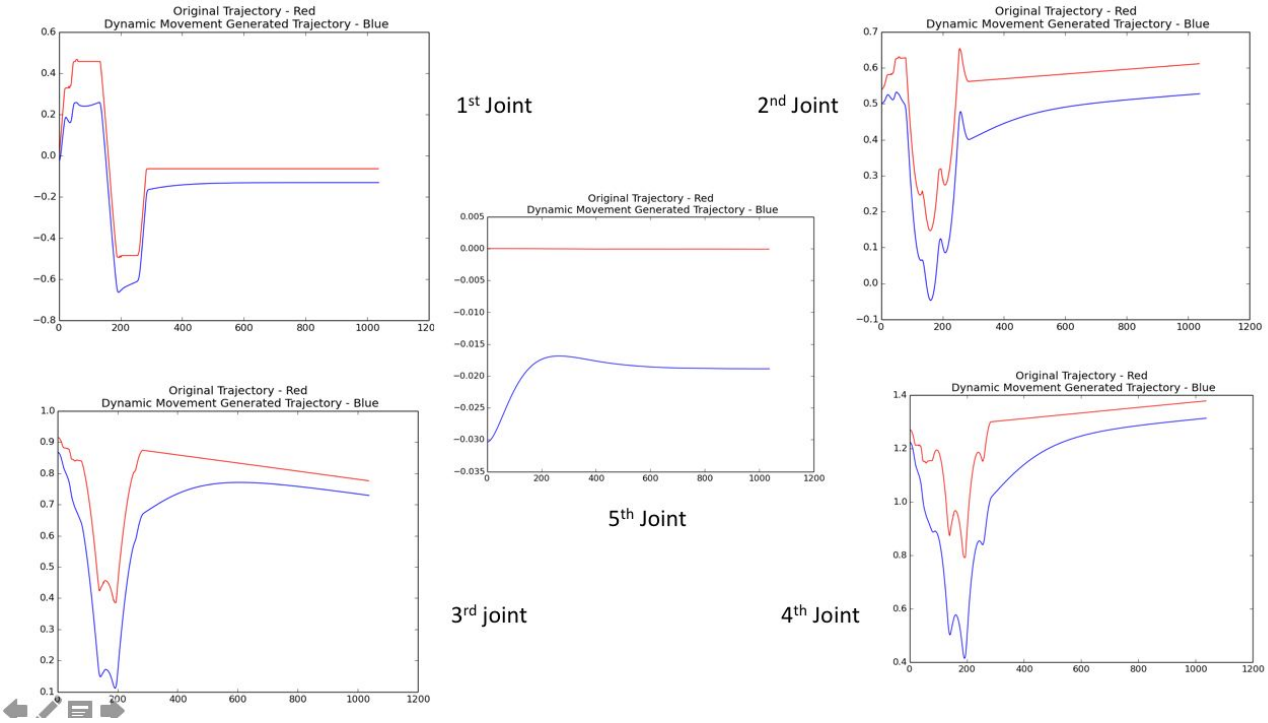

Results and Discussion

Here we present the results of the implemented AI and check for the robustness of the system with different initial setting or scaling. Videos for demonstrating each of the cases are attached as a zip file, or viewable through the links in Appendix A.

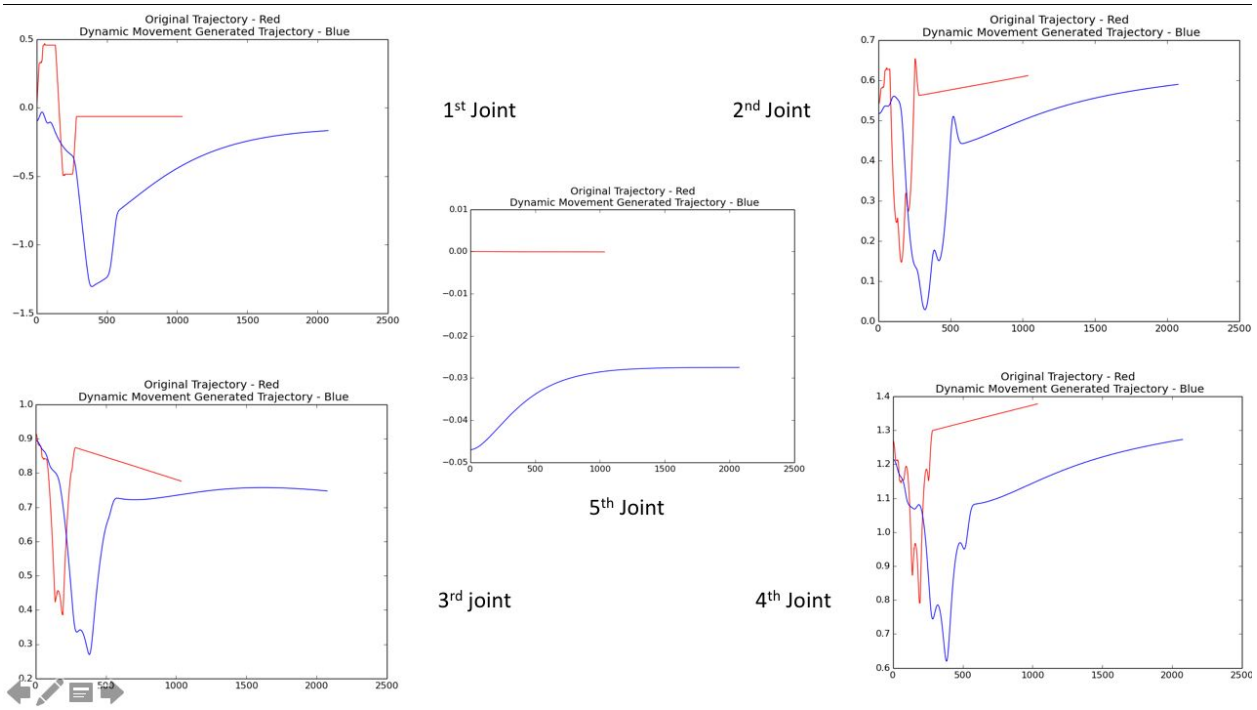
The first case is for non scaling, where the robot attempts to replicate the initial pattern exactly. The plots below show the error over time between the joint set point and actual angle. The plot in red is the expected trajectory that is the one logged while teaching the robot. The plot in blue is what the robot follows after having learned for the generated data.



The second case involves changing the end goal. An offset in the values of the joint angles is added to the learned system. The graph shows that even after the step input, the robot is able to adjust its position to match the new goal, even after an offset is given the general trajectory is followed by the learned system.



The final case involves changing the goal, as well as changing the scale. The graphs below show the system performance over time as before. We chose these variations to show that we can have the robot perform a certain trajectory in the required time, without having to explicitly define a new trajectory each time. The robot learns from it's given trajectory, and our implementation allows for it to adapt to different situations.



Future Goals

The first is to make DMP look at slightly more abstract data. As it is now, it's limited to only joint states. By opening it up to Cartesian states, we can add in additional control. Also, adding in some finer control of the joint limits and workspace zone. Next is to add in the ability for the You-Bot to drive while we move the arm. This will allow us to use DMPs with the mobile base, further expanding our capabilities. Adding in some more advances primitive calculations would allow us to have the robot work in conjunction with humans, learning from their inputs in a cooperative role, rather than a defining role. Finally, the implementation of obstacle avoidance is crucial for the overall use of the robot, and allow us further avenues to apply AI.

Conclusion

In the end, our robot was able to utilize Dynamic Movement Principles on order to replicate given and edited trajectories. We were able to use AI techniques to derive the building blocks of motions in order to later

recombine to suit our test. We were able to successfully implement a programing by demonstration platform for our chosen robot.

Appendix A - Videos

Non-AI:

Non-AI Baseline

AI Cases:

Square In Other Plane

Square Slow Temporal Scaling

Square In Different Plane Temporal Scaling