Phillip Nielsen

Optimization: Gradient based Algorithms and Differentiable Programing

Introduction:

For this project a simple 3 arm robot will be controlled with 3 inputs and optimized for accuracy and speed. The state space $X(t)$ will have x(t),y(t) and θ1(t), θ2(t), θ3(t) and the $\dot{\theta}_1$, $\dot{\theta}_2$,$\dot{\theta}_2$ will be the control parameters. Shown below is how the dynamics of the system will change with each time step.

$$x(t + 1) =- L1cos(\theta_1(t)) - L2cos(\theta_2(t)) - L3cos(\theta_3(t))$$
$$y(t + 1) = L1sin(\theta_1(t) + L2sin(\theta_2(t)) + L3sin(\theta_3(t))$$
$$\theta_1(t + 1) = \theta_1(t) + \dot{\theta}_1(t)\Delta t$$
$$\theta_2(t + 1) = \theta_2(t) + \dot{\theta}_2(t)\Delta t$$
$$\theta_3(t + 1) = \theta_3(t) + \dot{\theta}_3(t)\Delta t$$

The L1,L2 and L3 are the lengths of each arm going from center out to the tip which is the location of the x and y points. The closed loop controller will have limits on the rate at which each arm can rotate which for this case is 0.1 radians/sec in the positive and negative direction.

$$u(t) = [\dot{\theta}_1(t), \dot{\theta}_2(t), \dot{\theta}_3(t)] = \pi_w(X(t))$$

The π_w is the neural network which is used to optimize the controls with it being updated after each of the training data sets.

A loss function was implemented to optimize the controls/neural network which is calculated for each time step Δt. This loss function will be minimized when the tip of the arm is at (0.5, 1.5) as shown below.

$$min_w || x(t) - 0.5||^2 + ||y(t) - 1.5||^2$$
$$x(t + 1) =- L1cos(\theta_1(t)) - L2cos(\theta_2(t)) - L3cos(\theta_3(t))$$
$$y(t + 1) = L1sin(\theta_1(t) + L2sin(\theta_2(t)) + L3sin(\theta_3(t))$$
$$\theta_1(t + 1) = \theta_1(t) + \dot{\theta}_1(t)\Delta t$$
$$\theta_2(t + 1) = \theta_2(t) + \dot{\theta}_2(t)\Delta t$$
$$\theta_3(t + 1) = \theta_3(t) + \dot{\theta}_3(t)\Delta t$$
$$u(t) = [\dot{\theta}_1(t), \dot{\theta}_2(t), \dot{\theta}_3(t)] = \pi_w(X(t)), \forall t = 1, \dots, T - 1 \text{ (T is the final time step)}$$
$$- 0.1 \le \dot{\theta}_1(t) \le 0.1 , - 0.1 \le \dot{\theta}_2(t) \le 0.1, - 0.1 \le \dot{\theta}_3(t) \le 0.1$$

Phillip Nielsen

This was done using PyTorch which minimizes the loss function with respect to the control parameters as shown above.

Assumptions
Robotic controls normally take into account the mass moment of inertia, resulting forces, … for this problem it was greatly simplified to 3 arms with no mass and the controller had direct control of the angular velocity. This was also simplified to 2 dimensional space.

In the beginning of the project there was an effort to reduce the assumptions in a way that the controller only controlled the angular acceleration, with the setup it did not converge even with only 2 arms/bars. When simplifying the controls to be in full command of the angular velocity if converged, an extra arm was added for slight more complexity.

Analysis:

In the beginning of the project there was an effort to reduce the assumptions in a way that the controller only controlled the angular acceleration, with the setup it did not converge even with only 2 arms/bars. When simplifying the controls to be in full command of the angular velocity if converged, an extra arm was added for slight more complexity.

The code and results from the single and batch initial conditions can be found in the github repository below:
https://github.com/pnielse4/Optimization.git (Project1 folder)

Single Initial Condition

The initial condition is with the arm lying down in the negative x direction, so the tip location is (3,0) with arm lengths of 1 unit each. The target is to have the tip location reach point (0.5, 1.5) which is when the loss function is minimized. When running the simulation for 50 training iterations it can be seen that it converged on a solution as shown below.

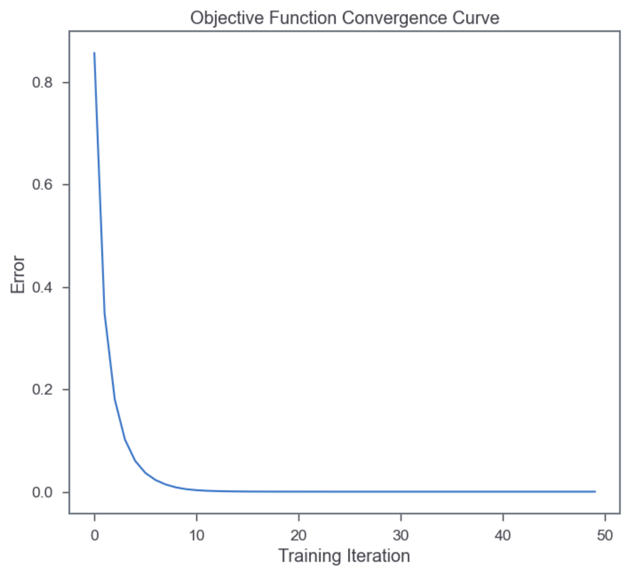

Figure 1: Convergence of single initial condition with 3 arms/bars.

From the figure above it can be seen that it converges on a solution quickly, this is mainly due to the fact that the dynamics of the problem are fairly simple. From this a graph of the tip was traced out and a plot the states (θ_1, θ_2 and θ_3).

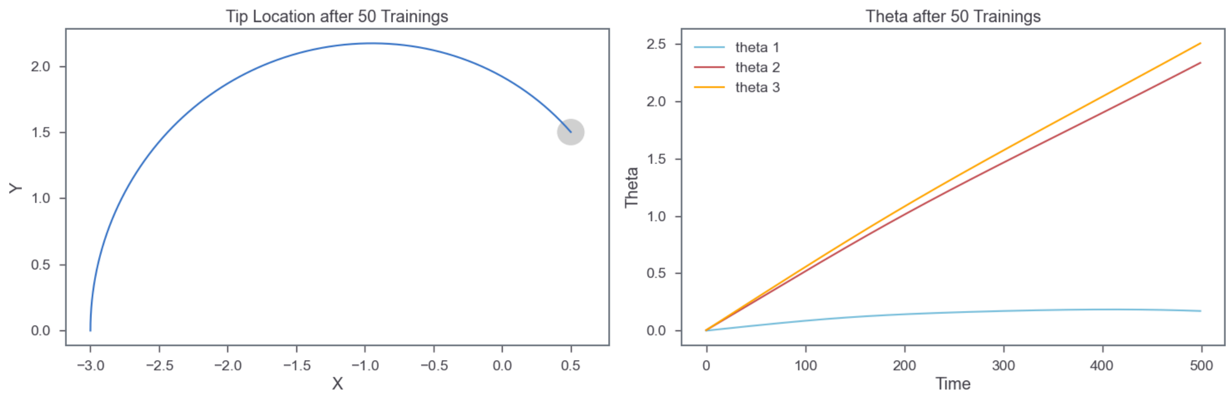

Figure 2: Plotting of tip location and theta states with single initial condition.

The path taken by the tip is a composite arc mainly from theta 2 and 3, which makes since this requires the least amount of input with the most movement towards the target location. A GIF of the animation was also created to show the moment of the arm with better visualization(github).

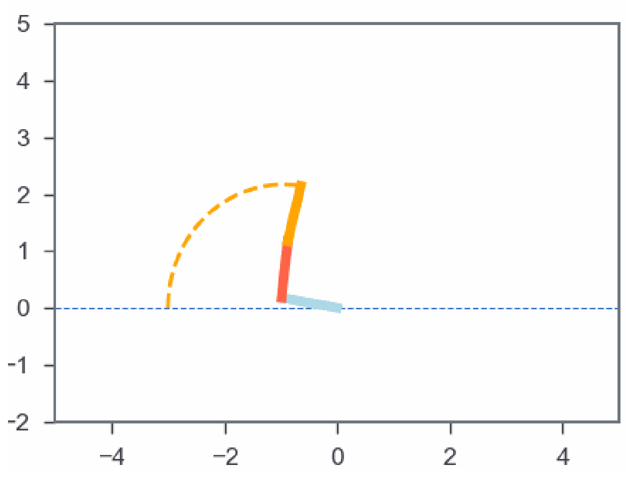
Figure 3: Preview of single initialization, GIF can be found in github.

Batch Initial Conditions

Using a batch of initial conditions can help train the neural network for a more robust controller. A batch of 10 randomly generated initial conditions were used to train the controller. The target location remained the same for all the initial conditions. The convergence of this simulation after 50 training iterations are as shown.

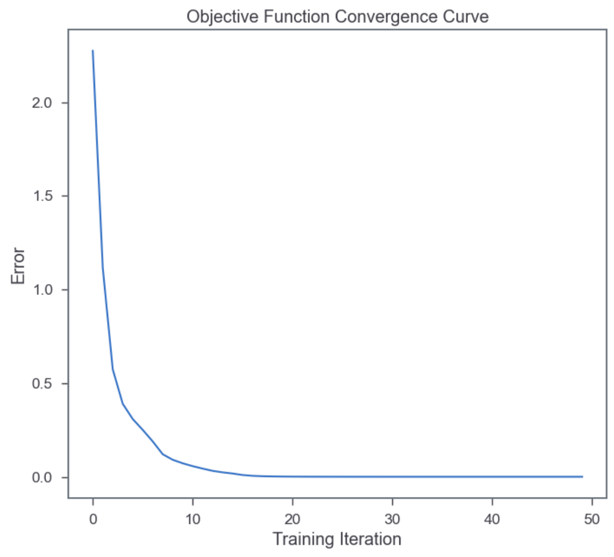

Figure 4: Convergence curve of batch initial conditions.

It can be seen that it takes slightly longer for the batch to converge compared to the single initial condition, which makes sense since it has to be optimized for a wider range of initial states. A plot was created to show how the states converged on the target location.

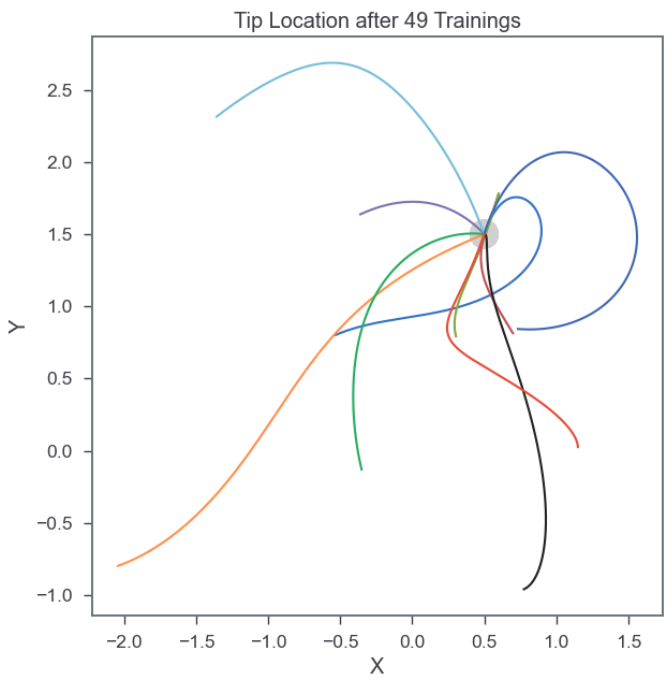

Figure 5: Tip location traces of all 10 random initial states.

It can clearly be seen that all the points converge at the target location and more interestingly the initial states further away tend to have a more straight line approach than those closer to the target. This may just be an outcome of not having a deep enough neural network due to the controller having to satisfy all the initial conditions. The related theta states are shown below.

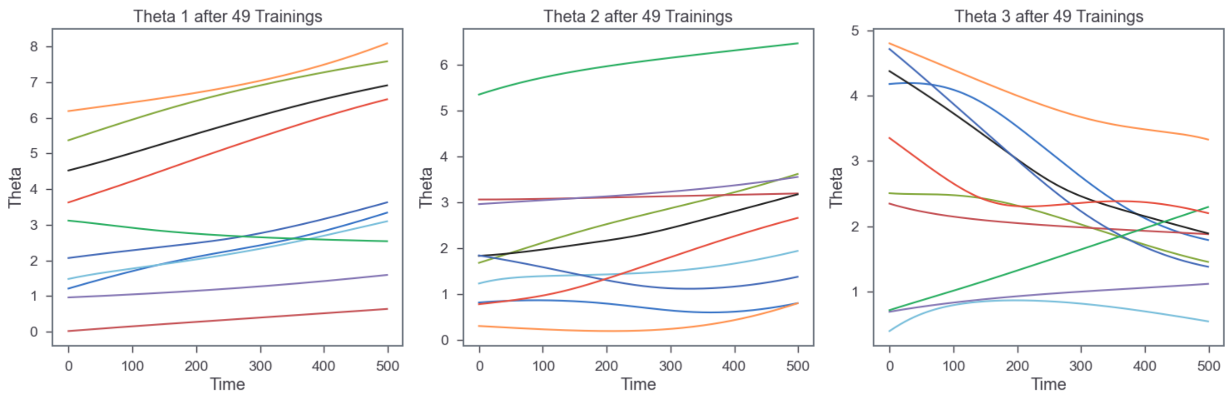

Figure 6: Theta states of all 10 random initial states.

From the plots above it seems that more control input is given to the last arm/bar (theta 3) which makes sense since it is trying to minimize the distance at the tip. An animation of the batch initial condition was also made and is located in the github repository.

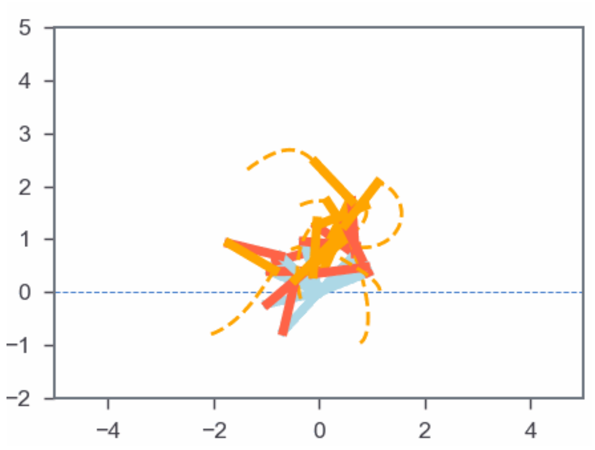

Figure 7: Preview of batch initialization, GIF can be found in github.

Lessons Learned

The biggest lesson learned is to always start with a simple problem and then build from that, since starting with a complex problem might sound good but it saves a lot of time to start small since then a better understanding of the mechanics of the problem can be had. Then things can be added on and a more clear approach can be had.

Majority of the issues with coding were getting the state matrix to update correctly with the controller input and then how to make the plots when a higher dimension matrix was created for the batch initial conditions.

Overall I really enjoyed the project and learned a lot in the process, and also how complex it can get when looking deeper into it.