Dynamic Optimization Assignment 2

Kazuya Otani

February 24, 2017

1 Optimizing walking

The goal of this section was to implement an optimization procedure to optimize simulation parameters to minimize the cost of a 20 second simulation. All of my code is in learn_cmaes.c, which uses the cmaes-c library I put in the same directory.

1.1 Initial optimizations

We were given a simulator that takes SIMBICON [1] controller parameters, simulates for 20 seconds, and returns a score. To optimize this score, I used the C implementation of CMA-ES. I modified learn.c to incorporate optimization, in which CMA-ES would pass parameters to the simulation function and get back a score. I set the initial value of the parameters to be the default values that we were given. I also scaled the parameters to be on the same order of magnitude (0 to 1), and adjusted the initial standard deviation accordingly, so that CMA-ES could search over the parameter space more uniformly. I found that this allowed it to drive down the cost in less iterations. (Note: I found out later that this could also be done by setting standard deviations for each dimension in CMA-ES parameters. This may be a cleaner way to achieve the same results.)
For the original cost function, the cost using the original parameters was 1075.34. After 50 iterations, I got a controller that achieved a score of 485.866. However, the gait for this controller didn't look very different from the original one. I believe this is because the original controller parameters were good enough that the rest of the changes that the optimization made were almost imperceptible.

1.2 Optimizing for Different Behaviors

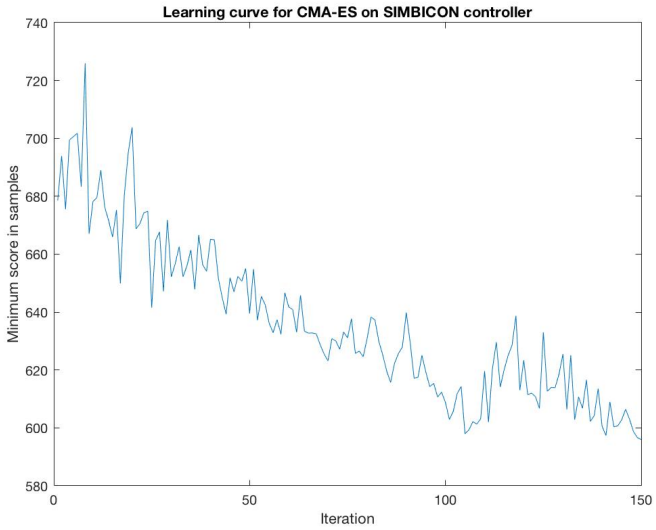
Next I changed the parameters of the objective function to try to get the model to walk faster (2 m/s desired speed, instead of 1 m/s). I found that if I only increased the speed penalty weight, the controller would not converge to a significantly faster gait. This is because the excessive foot force and torso pitch penalties were high enough that as the "robot" explored gaits that walked faster, the foot force penalty terms would reject those gaits in favor of more gentle gaits. Over the 50 iterations that I watched, the optimizer ended up keeping the speed penalty almost constant while driving down the foot force and torso pitch penalties.
To deal with the issue above, I decreased the foot force and torso pitch penalties by an order of magnitude. I also implemented the thresholded quadratic cost mentioned in [2], for the reasons stated in the excerpt below:

> "An obvious way to define an objective function is with a weighted sum of quadratic penalty terms on quantities such as total power consumption and deviation from a target speed. In practice, finding suitable weights for such terms is extremely difficult. For example, if chosen poorly, then even when the target speed is nearly achieved, optimization might continue to ignore the energy term in favor of imperceptible speed refinements. Instead, we employ a weighted combination of objectives that do not penalize small differences from targets. Specifically, we define a thresholded quadratic."

With this new cost function, the total cost and speed penalty for the default controller parameters (p0) were 696.9 and 540.3, respectively. Notice how the speed penalty term is the majority of the total cost. After 150 iterations of CMA-ES, the costs were 595.9 and 479.4. The robot walked visibly faster than the original gait, however, it did not reach 2.0m/s. This could be due to three reasons:

1. I didn't run the optimization algorithm long enough.
2. My initial guess and cost function for the optimization were not formulated well for achieving my goals.
3. There are some inherent limitations in the SIMBICON controller and our simplified model.

To test reason 1, I made a learning curve of the optimization over a longer number of iterations than usual. Below is a learning curve of CMA-ES over 150 iterations, showing the minimum score in its samples at each iteration. There seems to be a steady decrease in the cost function over the iterations, and the cost function seems to be nearing a local minima in this plot.



Reason 3 also seems reasonable, since the SIMBICON controller was specifically designed for walking gaits. Some its features, such as its finite state machine, do not apply to faster gaits such as running and galloping. The desired speed that I set (2.0m/s) is quite fast; it is the preferred transition speed from walking to running for humans. One reason animals change gaits is because of energy efficiency, so with a walking gait the model will have to use much more energy (torque, foot force) as it gets closer to 2.0m/s. The optimization will likely push the controller's parameters away from this space, back into the slower walking region.
I also tried an optimization with 20% higher torso mass. The gait that emerged was a slower one, due to the shifted optimal tradeoff between the speed penalty and torque/foot force penalties.
The gait animations are shown on the website (https://kazuotani14.github.io/school/dynopt/hw2/hw2.html).

2 Optimizing for Robust Gaits

The goal of this part was to optimize for a robust gait that can handle large random white noise torso perturbations.
For this section, I built a layer on top of the code I wrote for the previous section.
For my first implementation, I perturbed the torso with a random force between $-F_{max}$ and F_{max} Newtons with a probability of 0.025% at each time step. Given our 20 second simulation and 0.001 second timesteps, this results in an expected 5 pushes over the 20 second walk (if it doesn't crash before that). For each parameter set, I repeated the simulation 5 times and averaged their scores to get an overall score for the parameters. I chose to run it 5 times based on Figure 4 in [3] which showed that the success rate of the controller did not improve drastically past 5 samples.

I first chose to optimize the controller for occasional torso perturbations of maximum 50N impulses. (For reference, total body mass for the model is 75kg. 10 Newtons = 1 kgf.) The gait that emerged from the optimization was a strange shuffling movement, which was asymmetric and had a phase where both feet were sliding. The shuffling feet were penalized in the cost function, but the optimizer chose to drive down the other penalties instead. The controller seems to be robust to impulse perturbations up to 200 Newtons. However, past 250 Newtons the controller fails very brittly, as shown in the animations on the website.

The cost for a 20 second period of the perturbation-trained controller in perturbation-free conditions was 383.2, as opposed to 378.4 for the controller that was trained in perturbation-free conditions with the same CMA-ES parameters. This was not as much of a difference as I had expected. I thought that the controller trained with perturbations would have to trade off more of its optimality in the perturbation-free environment. This may have been affected by my choice of cost function as well.
Next, I tried optimizing the controller for a constant white noise torso perturbation. The torso was pushed with a uniformly distributed random force between -30N and 30N at every time step. The gait that emerged from this optimization was a careful one, with slower speed and shorter step length. The controller was able to reliably handle white noise of magnitude of up to 45N. With stronger forces, the feet were not able to maintain static friction with the ground and the robot fell, as shown in the animation on the website. This perturbation environment seemed to better demonstrate the strengths and versatility of the SIMBICON controller and CMA-ES optimizer; in the first few iterations, all of the samples (controller parameter hypotheses) crashed quickly. However, the optimization gradually drove the cost function down and found a set of parameters that worked within 50 iterations.
After doing all of my tests with the random perturbation method I realized that, as noted in [3], randomly picking perturbations for each parameter set makes it hard to reliably compare them, since we don't know whether performance was improved by the parameter differences or easier perturbations. A better way to optimize for uncertain inputs and environments would be to follow the authors' methods and generate a set of sequences of pushes at the beginning of the optimization, and use the same set of perturbations in calculating the score for all of the parameters.

References

[1] Yin, KangKang, Kevin Loken, and Michiel van de Panne. "Simbicon: Simple biped locomotion control." ACM Transactions on Graphics (TOG). Vol. 26. No. 3. ACM, 2007.

[2] Wang, Jack M., David J. Fleet, and Aaron Hertzmann. "Optimizing walking controllers." ACM Transactions on Graphics (TOG) 28.5 (2009): 168.

[3] Wang, Jack M., David J. Fleet, and Aaron Hertzmann. "Optimizing walking controllers for uncertain inputs and environments." ACM Transactions on Graphics (TOG). Vol. 29. No. 4. ACM, 2010.