Ball Casting

Jonas Mantel
TU Darmstadt

Stepan Konrad
TU Darmstadt

Abstract

In this paper, we teach a robot to cast a ball hanging on a string from its end effector into a cup.

To achieve this, we define a parameterized trajectory for the end effector and use the finite difference method to search for suitable parameters for this motion using the simulator.

As a second approach we apply the finite difference method to a forward model created using a Gaussian process regression.

We further analyse and compare the effectiveness, speed and robustness of the two methods and evaluate their respective merits for the task at hand.

1 Introduction

While controlling the precise movement of a robot arm can be a challenging task in its own right, a recurring problem is using the arm's movement to indirectly control an object loosely attached to it.

One example of this kind of task is the control of a mass attached to the arm's end effector by a string. In this problem, difficulties arise from the inability to directly influence the exact position, velocity and acceleration of the mass. Instead, the robot's controller is limited to moving the string's suspension point; owing to the flexibility of the string and the physical laws governing pendular movement, the mass then describes a path that is difficult to predict.

The field of robot learning provides tools to tackle these difficulties. By employing learning algorithms instead of analytical methods, many of the intrinsic problems of fine movement control can be circumvented.

We examine an instance of this problem where a ball hanging by a string from the end effector is cast into a cup placed in front of and below the robot. By casting the ball, the length of the string can extend the range of the arm, enabling the robot to hit a cup position outside of the arm's reach. In addition, since the ball's swing towards the cup is an intended part of the movement, there is no need to wait for the ball to stop swinging as there would be in a straight drop. This reduces the overall duration of the motion. By parameterizing the casting motion and calculating a reward corresponding to the distance of the cup to the ball's landing point, we can employ learning algorithms to solve the task. We analyse two methods that find suitable parameters to perform a successful motion.

In the following we will detail the setup we used, providing a definition of our arm motion with the corresponding parameters. Furthermore, we will give a short overview of the finite difference method for approximating the parameters and the finite difference method enhanced with a forward model created using a Gaussian process regression. Finally, we will detail the results of our analysis and give a conclusion based on these results.

2 Casting problem

2.1 Setup

For our analysis of the problem, we use a simulation software called SL. SL provides a simulation environment including a functional model of a robot arm, built-in motor control servos and a shared memory interface for communication with MATLAB.

The specific simulated robot arm we use is a Barrett WAM. For use with the casting task, the simulation is extended by a string of fixed length attached to the arm's end effector, a ball attached to the string and a cup which can be variably placed at any point in the simulation environment.

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume 22 of JMLR: W&CP 22.

The robot learning algorithms are written in the MATLAB programming language and the end effector trajectories are sent to the simulator using the MATLAB interface provided by SL.

2.2 Problem

The goal of the casting problem is to bring a ball hanging on a string from the end effector into a cup on the floor. It should be noted that the goal is not to simply move the end effector above the cup and lower the ball into its opening; rather, the ball is to be cast towards the cup, swinging on its string, and then dropped down while the string is extended away from the end effector. The casting itself is achieved by a combination of backward and forward movement, letting the ball gather momentum for its swing. To achieve this goal, the robot has to learn working parameters for the motion. This motion is defined as follows:

In our analysis, a casting experiment begins in an initial position (see Figure 1, 1.). The shoulder joint is set at a negative angle with the elbow joint bent at a positive angle, so the forearm is almost in a horizontal position. The ball hanging from the end effector is not directly above the cup; instead, the cup is placed a variable distance in front of and below the ball.

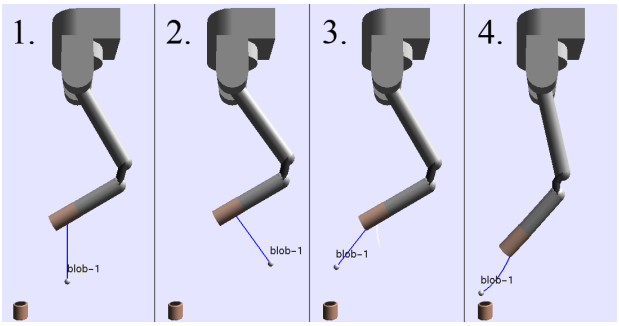

Figure 1: Outline of the ball casting movement, divided into four steps:
1. Initial position
2. Backward movement
3. Forward movement
4. Downward movement

The actual motion starts with a backward movement from the initial position (2.). The end effector then moves back towards the cup until the arm has resumed its initial position (3.). This first basis movement b_1 of the motion, comprised of the backward and forward swing, is defined as:

$$b_1 = \begin{cases} (1 - \cos(2\pi t))\frac{\pi}{12}\theta_1 & \text{if } t \leq 500 \\ 0 & \text{if } t > 500 \end{cases}$$

where θ_1 is a first parameter which controls the amplitude of the movement.

Afterwards, there is a pause which allows the ball's built up momentum to propel it forward, letting it swing beyond the end effector's length. The duration of this waiting time is parametrized by a second parameter θ_2. Finally, the end effector moves downwards to a stretched position, letting the ball drop downwards from its swung-out position (4.). This second basis movement b_2 is defined as:

$$b_2 = \begin{cases} 1.5t & \text{if } t \geq 500 + 100\theta_2 \\ 0 & \text{if } t < 500 + 100\theta_2 \end{cases}$$

The complete motion of the end effector can be modeled by a trajectory τ for the robot arm, consisting of components for every joint. The shoulder joint's component τ_s and the elbow joint's component τ_e are combinations of the two basis movements b_1 and b_2 and are defined as follows:

$$\tau_s = -b_1 - \frac{\pi}{6} - \max\left(\frac{1}{2}b_2, -\frac{\pi}{6}\right)$$
$$\tau_e = \max\left(b_1 + \frac{\pi}{2} + b_2, 0\right)$$

All other components are set to 0.

To use robot learning methods, we also need to receive some feedback from the setup. This comes in the form of a reward J, which is a negative value directly proportional to the quadratic distance of the cup position and the ball position determined when the ball drops below the cup's rim height. The reward is calculated as follows:

$$J = -100(p_{\text{cup}} - p_{\text{ball}})^2$$

3 Learning Methods

Learning the movement parameters is accomplished by using two learning algorithms. Our first approach is to use the finite difference method on the SL simulator. As a second approach we collect output data for a set of input samples, and use this data to acquire a forward model by using a Gaussian process regression on the input and output samples. We then use the finite difference method again, but instead of using it on the SL simulator directly, we apply it to the forward model.

3.1 Finite difference method

The strategy of the finite difference algorithm is to approximate the derivative of the function f, which

maps the motion parameters to the distance between the cup and the point where the ball hits the bottom.

To compute this gradient, an initial parameter θ is randomly perturbed called $\theta + \delta_i$ with a certain standard deviation which is later called perturbation factor η.

$$\Delta = \begin{pmatrix} \delta_1 \\ \vdots \\ \delta_n \end{pmatrix}$$

Then the function is evaluated at the current θ and several perturbations.

$$J_{\text{ref}} = f(\theta)$$
$$J_n = f(\theta + \delta_n)$$

The gradient ∇J, which points into the direction of the biggest improvement of the reward, is then calculated as the following:

$$\nabla f(\theta) = (\Delta^T \Delta)^{-1}(J_n - J_{\text{ref}})$$

θ is updated in the direction of the gradient with a certain learning rate γ.

$$\theta_{i+1} = \theta_i + \gamma \nabla f(\theta)$$

The algorithm iterates until θ converges or the reward is over a specific threshold.

3.2 Gaussian process regression

The Gaussian process is a kernel-based process that can be used to interpolate projected results between or around known data points.

To improve upon the performance of the simple finite difference method, we use Gaussian process regression to create a forward model. A forward model is a model that uses a set of training data to try to predict expected outputs for hitherto unprocessed input values. Since the usage of a forward model is often far faster and less costly than running tests on real world machinery, it can save valuable time, effort and resources. In our analysis, we use a forward model to quickly iterate over large sets of input data, since the slow speed of a real-world robot arm makes running a large batch of trajectories on it time consuming. The forward model can also help to reduce noise introduced by real-world physics and other non-deterministic factors of the mechanical movement, making convergence on a set of parameters easier.

3.2.1 Creating a forward model

To construct the model (see Figure 2), we collected a set of 400 data points from SL simulations, each consisting of one value for each of our motion's parameters and the corresponding drop down point of the ball, giving the maximum distance the ball can still hit the cup for each pair of parameter values. We then used this training set as input for the Gaussian process regression, creating a function capable of predicting ball ranges for new parameter sets to within the certainty indicated by the process's variance output.

Having collected the resulting drop down points p for each combination of input parameters $(\theta_{1_i}, \theta_{1_j})$, the matrix K, the vector k and the scalar c, containing evaluations of a covariance function, can be calculated as follows:

$$[K]_{ij} = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{1}{2}\left(\frac{\theta_{1_i} - \theta_{1_j}}{d_1^2} + \frac{\theta_{2_i} - \theta_{2_j}}{d_2^2}\right)\right)$$
$$[k]_j = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{1}{2}\left(\frac{\theta_{1_i} - \theta_{1_{\text{new}}}}{d_1^2} + \frac{\theta_{2_i} - \theta_{2_{\text{new}}}}{d_2^2}\right)\right)$$
$$c = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{1}{2}\left(\frac{\theta_{1_{\text{new}}} - \theta_{1_{\text{new}}}}{d_1^2} + \frac{\theta_{2_{\text{new}}} - \theta_{2_{\text{new}}}}{d_2^2}\right)\right)$$

The variable d denotes two times the distance between the sample points. As K is independent of θ_{new}, it can be precalculated from a set of sample points and used for for different parameter evaluations. Then the estimated position p_{new} and a corresponding variance of the estimation can be calculated:

$$p_{\text{new}} = k^T (K + \lambda I)^{-1} p$$
$$\sigma^2 = c - k^T (K + \lambda I)^{-1} k$$

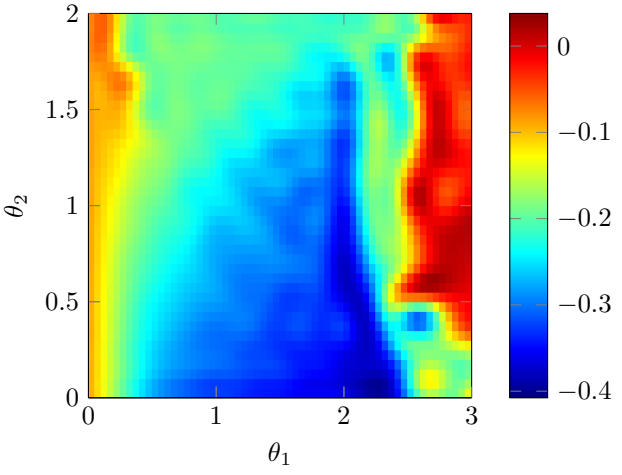

Figure 2: Forward model created of 400 samples using Gaussian process regression:
For each combination of θ_1 and θ_2, the color corresponds to the estimated ball location p_{cup}

To verify the validity of the forward model, we used it to predict the result for simulations using the parameters from the training set, giving us the opportunity to compare the predictions with the actual simulation results we had collected. We observed that the Gaussian process regression model was fitted through the sample points nicely.

3.2.2 Using finite difference with a Gaussian process regression

Using the forward model described in the above, we again use the finite difference method to learn our parameters. This time, however, we apply the method to our model, substituting the model for the direct interaction with the SL simulator. We verify the resulting parameters by running them through a SL simulation; if needed, these can be used as the starting point in another application of the finite difference method on SL, yielding a convergence in fewer iterations than would otherwise be needed.

4 Experiments and evaluation

In this section, we describe the approach of our experiments and the results we conclude. There are several possible variations of the input parameters: For example, it might be relevant to figure out which standard deviation of the random parameter perturbation (in the following called perturbation factor η) in the finite difference method is sensible. Concerning the finite difference method on the forward model, the minimal required resolution of the model is also of interest. If applicable, we test those variations on both methods while leaving the other parameters constant. The default parameters of our tests – unless mentioned otherwise – are the following:

perturbation factor	$\eta = 10^{-2}$
learning rate	$\gamma = 0.05$
cup position	$p_{\text{cup}} = -0.3$
model resolution	$n_{\text{data points}} = 400$
FD parameter samples	$n_{\text{samples}} = 4$
initial parameters	$\theta_{\text{init}} = (0, 0)$
maximum iterations	$i_{\text{max}} = 50$

4.1 Finite difference method

The first algorithm we analysed is the finite difference method. As described above we use the simulator directly to calculate a reward for specific input parameters. Figure 3 shows an exemplary learning curve for the default cup position of -0.3. The graph shows that this method is capable of finding suitable parameters for the defined motion.

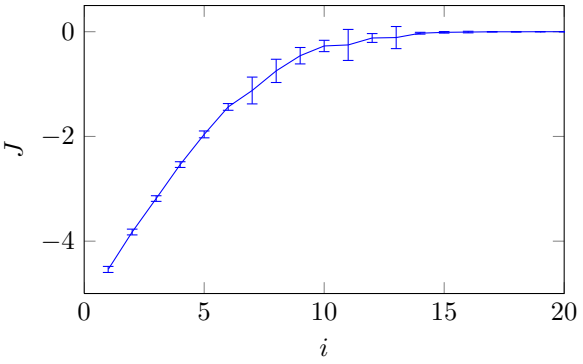

Figure 3: Finite difference method:
Rewards over iteration for $p_{\text{cup}} = -0.3$

We observed that the finite difference method does not converge well if we set the perturbation factor too low, for example $\eta = 10^{-3}$ or lower. To optimize the methods' performances, we ran an additional set of experiments with varying perturbation factors on both methods. The results can be seen in Figure 4. For the finite difference method, we found that decreasing η resulted in better rewards and fewer iterations only until a factor of $\eta \sim 10^{-2}$ is reached.

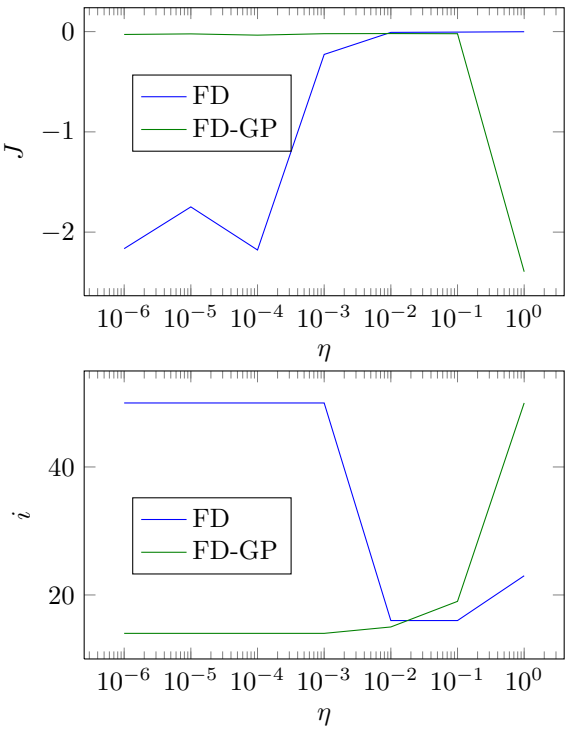

Figure 4: Rewards and iteration count for varying perturbation factor η

As the finite difference algorithm with Gaussian process regression allows for a smaller η, we conclude that the simulation of the casting task in SL has more noise

or discontinuities. Upon η reaching an order of magnitude comparable to that of the noise, a decrease in performance of the method, such as we observed, would be expected.

4.2 Finite difference method with Gaussian process regression

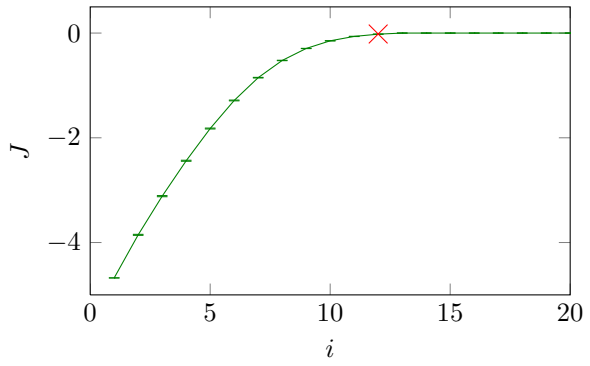

Figure 5: Finite difference with Gaussian process regression:
Rewards over iteration for $p_{\text{cup}} = -0.3$
A final simulation with predicted parameters was done in SL – here marked with a red cross.

Using a gaussian process regression to create a model out of sample points, we analysed the finite difference method on this model as a second approach. We found that the model-based approach was not only superior with regards to the number of iterations needed for success and the speed of each individual iteration, but that it was also more robust with regards to convergence for varying starting parameters. As shown in Figure 5, this method tends to converge in a straightforward manner, with less risk of regressions to worse rewards or fluctuations.

As mentioned in the beginning of the section, we also tried to figure out the minimum count of data points for the forward model estimation. Figure 6 shows the result of this analysis. For our problem, 25 data points are sufficient to create a forward model through Gaussian process regression to achieve suitable motion parameters.

This low number also means that using a forward model can be very efficient, since the only evaluations that have to be done on the actual robot are the ones needed to create the model. All subsequent usage is independent of the hardware.

In contrast, the unaided finite difference method requires one evaluation for every sample and one for the reference per iteration; As we used 4 samples and typically reached between 10 and 30 iterations for one run, the forward model amortises its sampling cost in only

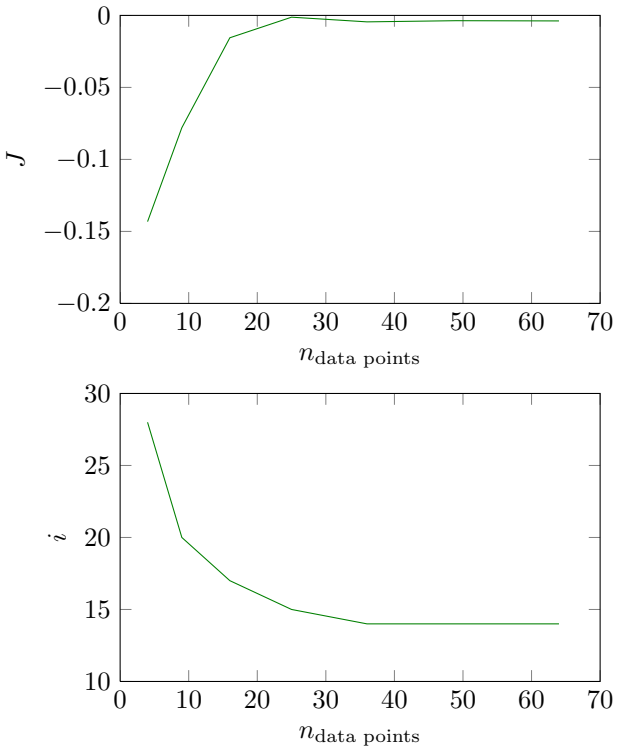

Figure 6: Finite difference with Gaussian process regression:
Rewards and required iterations over data points count

one run. Even with the greater sample set of 400 data points, 4 to 5 runs are enough to have the forward model require less evaluations on the robot than the unaided finite difference method.

5 Conclusion

In this paper, we compared two methods for learning parameters for a casting problem.

We have shown that, while the finite difference method applied directly to the robot will reach success in many cases, creation of a forward model and application of the method to the model instead of the robot will yield success for a broader range of initial parameters.

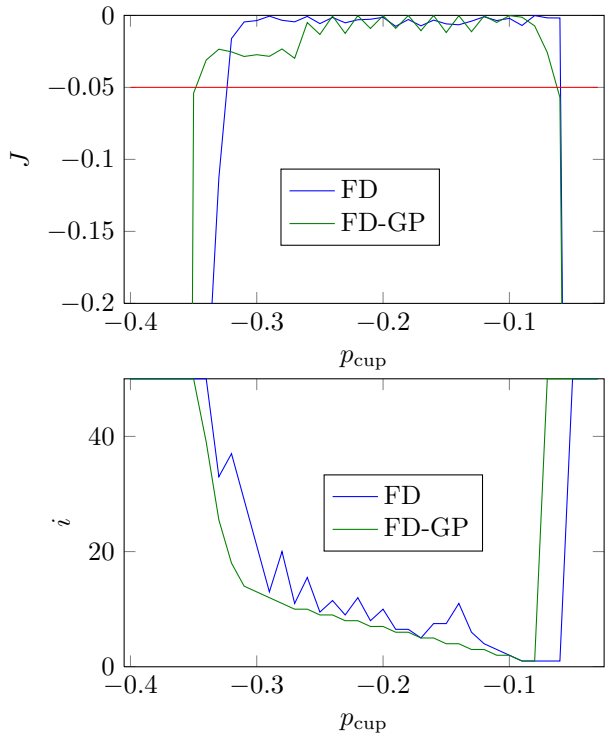

Figure 7: Rewards and iteration count for varying cup positions p_{cup}. The ball is in the cup if the reward is over a certain threshold (red line)

Both methods perform very well in achieving the goal for different cup positions, as can be seen in Figure 7. In contrast to the finite difference method on the forward model, however, the direct finite difference algorithm requires more iterations and more evaluations on the robot arm overall.