Learning how to cast a ball into a cup using a robotic arm

Nicole Brunkhorst
TU Darmstadt

Stefan Rado
TU Darmstadt

Abstract

Many real-world problems involve the control of pendulums and other swinging masses such as automatic cranes. Precisely controlling them can be very difficult depending on the environment and technical capabilities.

In this paper, we examine how to use a robotic manipulator to bring a ball hanging on a string from the end effector into a cup on the floor. We present different approaches to control such a task and finally solve the problem by learning a forward model that we use to plan the required motion. Our approach successfully hits the cup in 90% of the cases.

1 Introduction

Controlling a freely swinging mass is a difficult task itself and it gets even harder if it shall be moved to a target position. The problem we want to tackle in this paper is to bring a ball hanging on a string into a cup on the floor using a robot manipulator.

What makes this task difficult is that the only way to move the ball is to move the suspension point of the string. It is very hard to predict the reactions of the ball as a string is very flexible and we have no possibility to measure the ball's current position besides using complex vision-based approaches.

Additionally to the lack of information about the pendulum, controlling a robot manipulator to do precise movements turns out to be a problem that can hardly be solved using analytical methods. Although many robots are build such that it is easier to calculate influences such as gravity and centripetal forces, even small influences such as wear and tear can render analytical models unusable.

That is where the field of robot learning techniques come into play. Instead of finding a control method analytically, it is possible to learn a control policy from training data.

For fast motions, it may be impossible to reevaluate the motion's parameters for each time step as this takes time. It is also a lot simpler to search through a limited policy space before actually starting the motion. There are two ways to proceed: Model-free policy search methods that allow learning a controller without a model of reality and model-based policy search methods that first learn a model and then use it to derive a controller [5, p. 227].

Model-free policy search methods have the advantage that they can directly map from a target state to an action required to get to this target state. They can directly be used for control, but they also have a few downsides: First of all, such an Inverse Model is not necessarily a function – one target state could be reachable by different actions and the model does not have to return the best of the possible actions. Also, learning them requires a lot of data as it needs to be trained separately for each target state. In computer simulation, this might not be a big problem, but when working with a mechanical system, a lot of data inherently means that a lot of experimental time is required, moreover causing wear and tear to the system.

Model-based policy search methods use a Forward Model that can be used to predict how the environment will respond to actions [5, p. 230]. Learning a Forward Model needs less training data, but requires additional effort to find the action leading to a desired target state.

To solve our ball-casting problem, we decided to learn a Forward Model from training samples. We then use this Forward Model to plan the motion parameters required to cast the ball into the cup.

2 Casting

Our approach to model-based machine learning to learn motor skills requires a couple of steps. First, we introduce our set-up used to implement the problem (Section 2.1). We continue with describing the motion the robotic arm has to follow to swing the ball such that it falls into the cup and specify the parameters required to describe this motion (Section 2.2).

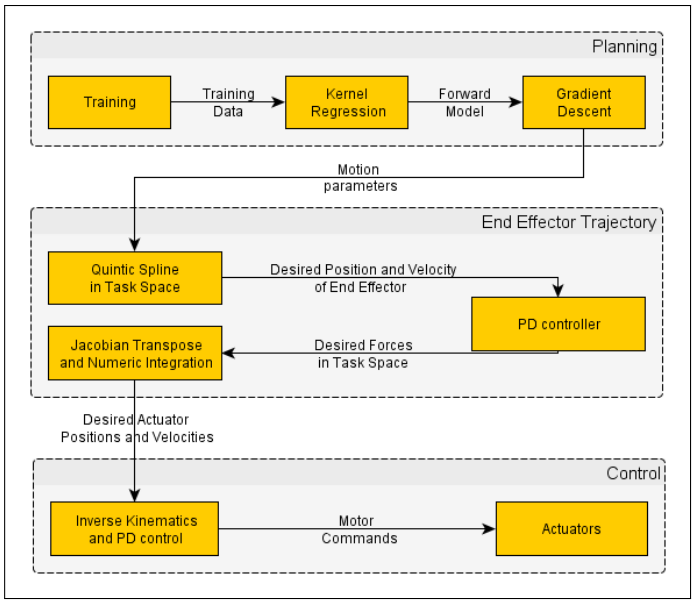

Figure 1: Project structure

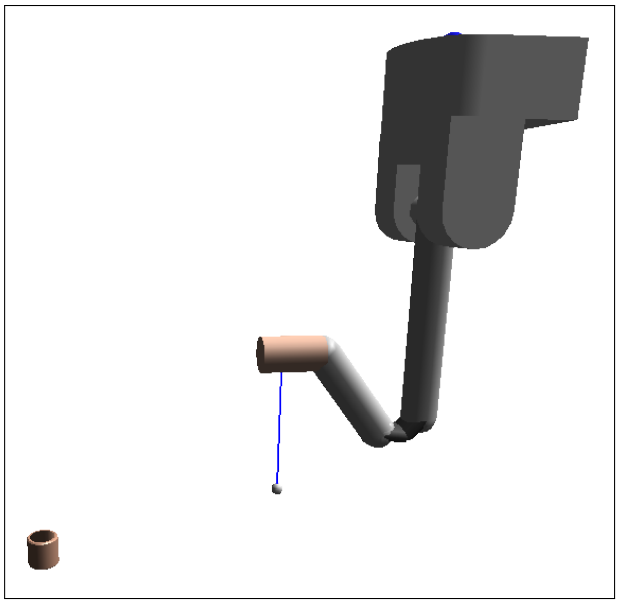

Figure 2: Robot arm set-up

Subsequently, we describe how we implemented the controller for this motion (Section 2.3).

Having a working robot (simulation), we continue with learning a forward model of the problem (Section 2.4). This model can then be used to find the motion parameters required to cast the ball into the cup sitting at a given position (Section 2.5). An overview of our approach can be found in Figure 1.

2.1 Set-up / Environment

The robotic manipulator used in our study is a 7-degree-of-freedom WAM™ Arm by Barrett Technology® Inc. that is mounted onto the ceiling. A string connects the end effector to a small ball (see Figure 2).

To speed up development and training data generation, we used the robot simulator SL [4] instead of real hardware. Nevertheless it should be possible to adapt the robot controller presented in this paper to the real world with only minimal changes. The described learning techniques are independent of the concrete robot manipulator.

For complex mathematical computations, SL can be connected to the numerical computing environment MATLAB. We therefore implemented all expensive matrix calculations required by the learning and planning algorithms in MATLAB while doing the robot simulations in SL.

2.2 Specification of the Motion

To cast the ball into the cup, the robotic arm needs to perform a precise motion. To allow learning of the motion's parameters, the motion also has to have as little noise as possible. See Figure 3 for a visualization of the following explanations.

For simplification, we expect the cup to be on a fixed line on the floor, so only the distance to the base of the robot arm varies while the angle is fixed. Generalization to arbitrary cup positions is straightforward by calculating the angle with which to turn the first degree-of-freedom.

Every casting attempt starts with a defined initial arm posture. This posture is chosen such that the ball is hanging freely from the end effector, its lower edge being above the height of the cup's upper rim. The ball must not be swinging for the attempt to be reproducible. This initial posture is also shown in Figure 2.

To accelerate the ball in the direction of the cup, the end effector needs to perform a horizontal movement towards the cup. The height of the end effector should not change during this first movement, otherwise the ball might hit the cup from the side, being unable to fall inside from the top. The speed of the horizontal movement should be high enough to get the ball swinging if the cup's position is outside of the end effector's range.

After finishing the horizontal movement, the robot needs to wait for the now swinging ball to catch up.

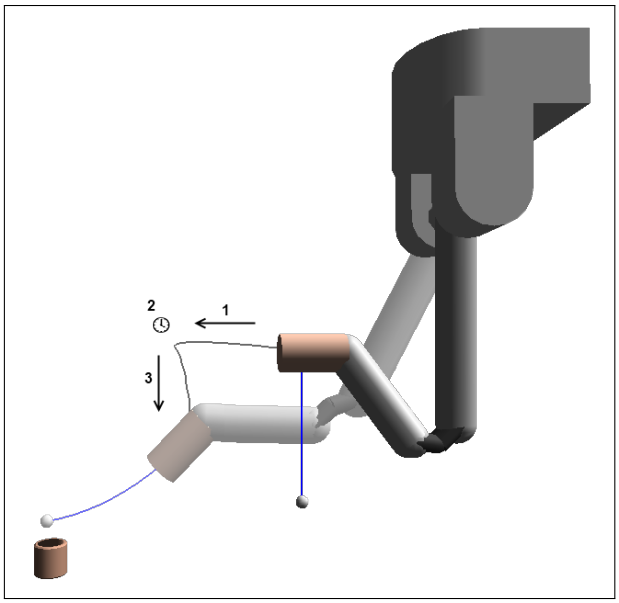

Figure 3: Ball casting motion

The exact waiting time determines how far the ball moves forward. The end effector must then perform a downwards movement that is fast enough to end the string's restriction on the ball so that it falls down into the cup. After the ball falls into the cup or on the floor, the robot arm returns to its initial posture.

The whole motion can be parametrized to hit different targets using the following three parameters:

1. The horizontal movement's distance.
2. The horizontal movement's speed.
3. The waiting time.

2.3 Implementation of the Robot Controller

We implemented a closed-loop controller to execute the motion specified above. Using a simple state-machine, we go through the three stages of the motion – the horizontal movement, the waiting time and the downwards movement.

As joint motors can only influence accelerations and jumps in accelerations may be dangerous at high speeds, we need smooth movements with defined start and end accelerations. To achieve this, we calculate the coefficients $\{a_i : 0 \leq i \leq 5\}$ of quintic splines

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

for motion stage 1 and 3, using the arm's start and target positions and zero initial and terminal velocities and accelerations as the splines' boundary conditions

$$\begin{aligned} x(t_0) &= \text{initial position} \\ \dot{x}(t_0) &= 0 \\ \ddot{x}(t_0) &= 0 \\ x(t_f) &= \text{final position} \\ \dot{x}(t_f) &= 0 \\ \ddot{x}(t_f) &= 0, \end{aligned}$$

where t_0 and t_f are the start and end times of the movements. We can then use these splines to calculate the current desired position $x(t)$ and velocity $\dot{x}(t)$ of the end effector for each time step t.

To bring the end effector to the state given by the spline evaluation, we use a PD controller to calculate the error between its current and the desired position and between the current and desired velocity

$$u_x = K_p(x_{des} - x) + K_d(\dot{x}_{des} - x).$$

The resulting control forces in cartesian space must then be translated into joint space using the transposed Jacobian

$$u_q = J(q)^T u_x.$$

2.4 Learning the Model

To learn a forward model of our robot, we first generate training data which we can then combine using Kernel Ridge Regression. Generating the training data is done by running simulations with random parameters. Each parameter – horizontal distance, horizontal speed and waiting time – gets drawn from uniform distributions between reasonable limits. We obtain the location where the ball crosses the virtual plane lying on top of the cup.

As generating samples is cheap using simulation, we also generate smaller validation and test data sets to allow tuning and evaluation of the model.

Using the training data $\{(\mathrm{x}_n, \mathrm{y}_n) : n \in \{1, ..., N\}\}$ we can now predict the ball's impact location $m(x)$ for new motion parameters x using Kernel Ridge Regression [1, chapter 6]

$$m(x) = \mathbf{k}(x)^T(\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{Y},$$

where $\mathbf{k}(x)$ is a vector with elements $[\mathbf{k}(x)]_n = k(x, \mathrm{x}_n)$, $\mathbf{K}$ is the *Gram* matrix with $[\mathbf{K}]_{ij} = k(\mathrm{x}_i, \mathrm{x}_j)$ and $\mathbf{Y} = (\mathrm{y}_1, ..., \mathrm{y}_N)$. k is a Gaussian kernel of the form

$$k(x_i, x_j) = \exp(-\frac{1}{\sigma^2}||x_i - x_j||^2).$$

We use Gradient Descent optimization [2, p. 466] to determine the σ for the kernel that leads to the minimal Mean Squared Error to the validation data set.

2.5 Planning a Motion

Now that we learned a Forward Model of the robot, we want to use it to find the correct parameters to cast the ball into the cup standing at a given position. Our model gives us a mapping from motion parameters $\mathbf{x}$ to ball impact locations $\mathbf{y} = m(\mathbf{x})$, so we use Gradient Descent optimization to find motion parameters that minimize the model-predicted impact location to the cup position $\mathbf{y}_{\text{cup}}$:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla (m(\mathbf{x}_i) - \mathbf{y}_{\text{cup}})$$

We estimate the model error's gradient $\nabla(m(\mathbf{x}_i) - \mathbf{y}_{\text{cup}})$ using the finite-difference method. As this only requires evaluation of the learned model, less data from the real robot is needed.

The step width α for every gradient descent step is chosen proportionally to the distance of the current impact location prediction to the target location

$$\alpha \sim ||m(\mathbf{x}_i) - \mathbf{y}_{\text{cup}}||.$$

The step width therefore gets smaller the better the parameters get.

We end the gradient descent once the predicted impact location $\mathbf{x}_i$ is within a range of 1 cm of the cup's center position $\mathbf{y}_{\text{cup}}$, which is close enough to get the ball into the cup, or once we did a certain maximum number of steps.

3 Evaluation

To test and tune our implementation, we sampled 1300 training data sets together with 300 validation and 100 test data sets. Figure 4 shows Histograms of the uniform random motion parameters of the training data and their corresponding ball impact locations.

Figure 5 shows the Root Mean Square Error (RMSE) towards the validation data of a Kernel Regression using different σ. The lowest error is reached with $\sigma = 0.35$. The RMSE towards the test data using this σ is 2.24 cm.

As a reference, we also tried linear regression with fitting the training data to polynomials [1, chapter 1.1]. A 12-th grade polynomial proved to be best, leading to an RMSE of 21.7 cm towards the test data. Obviously, Kernel Regression performs a lot better.

The results of using the learned model to bring the ball into cups sitting at different positions can be seen in Figure 6. The policy search was able to find parameters for which the model predicted the impact location to be a mean of 8.5 mm off of the center of the cup.

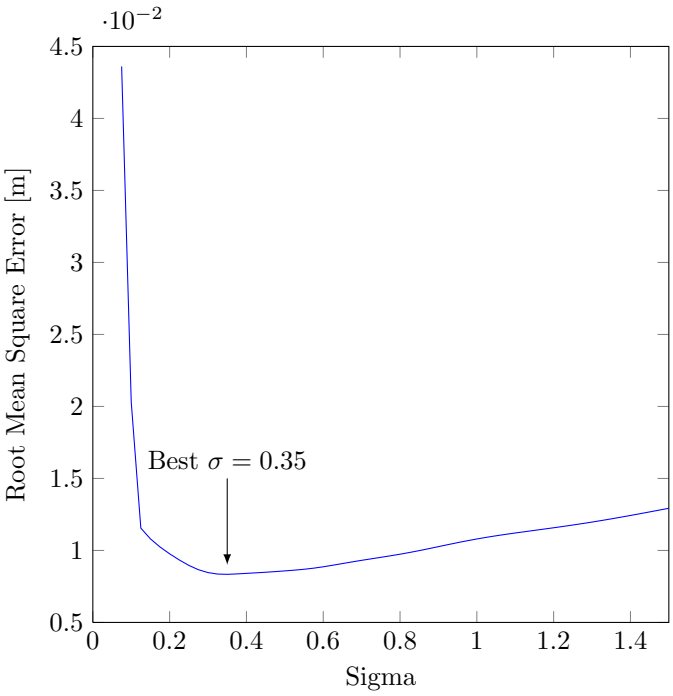

Figure 5: Root Mean Square Error towards the validation data using different σ

As the cup's inner radius is 3 cm, all except two tries hit the cup. For these two tries, the policy search predicted an impact location about 2 cm off the center of the cup. The gradient descent could not find better motion parameters, most likely due to a local minimum in the training data. The ball then bounced off of the rim of the cup, leading to an impact location 5 cm off the cup's center.

4 Conclusion

As described above, our approach works quite well to solve the problem. It is able to bring the ball into the cup for most cup positions. The few times the cup could not be hit are due to local minima in the model which might be avoided using different optimization methods for the policy search.

A future addition to the implementation would be to add the possibility to move the cup in two dimensional space on the floor instead of just one.

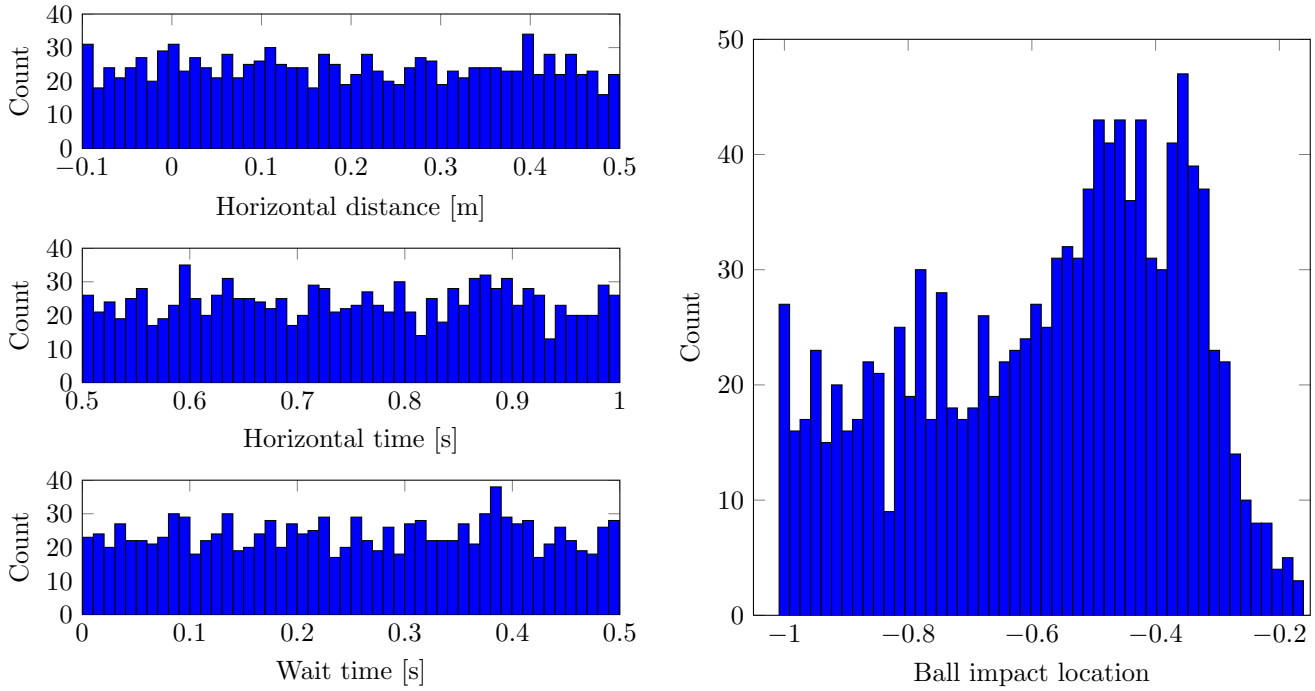

Figure 4: Histogram of parameter distributions and ball impact locations in training data

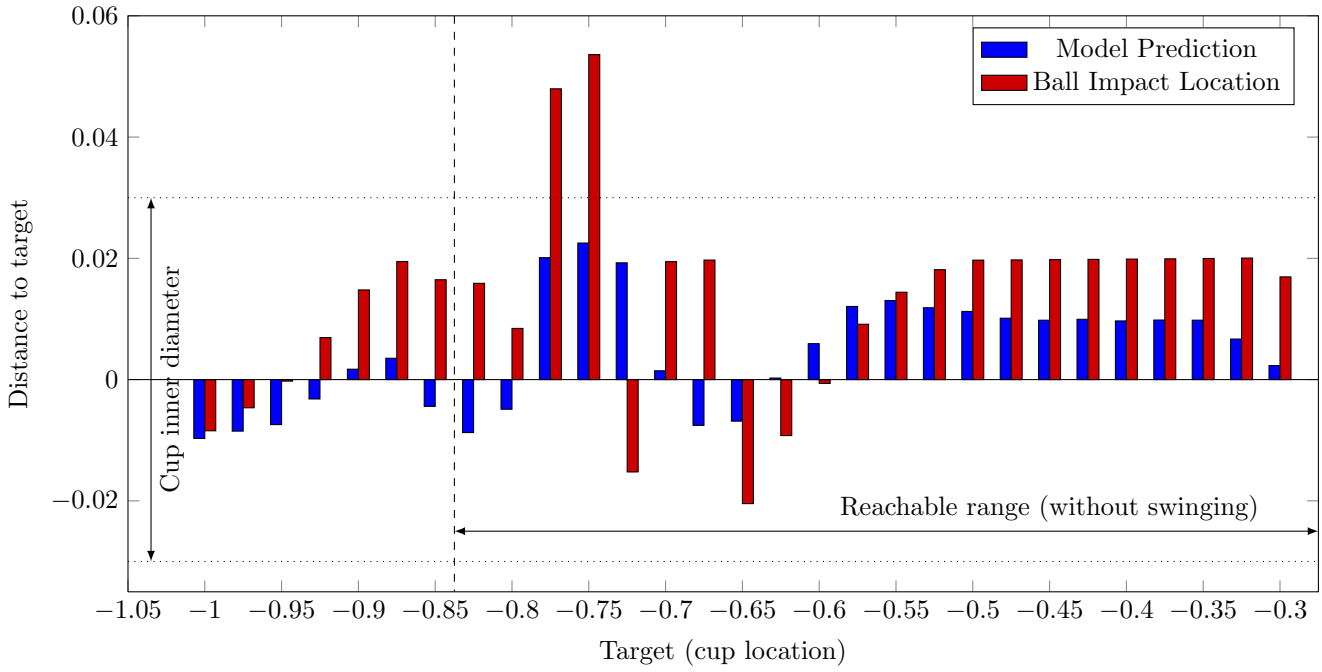

Figure 6: Performance of the model-based policy search

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science + Business Media, LLC, 2006.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] Afshin Rostamizadeh Mehryar Mohri and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, Cambridge, MA, 2012.

[4] S. Schaal. The sl simulation and real-time control software package. Technical report, 2009.

[5] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.