Underactuated Trajectory-Tracking Control for Long-Exposure Photography

Christian Eilers*
christan.eilers@stud.tu-darmstadt.de

Jonas Eschmann*
jonas.eschmann@stud.tu-darmstadt.de

Robin Menzenbach*
robin.menzenbach@stud.tu-darmstadt.de

Abstract

Controlling underactuated system is still a challenge because of their chaotic behavior. In this report, we apply optimal control techniques to a rotary inverted pendulum, also known as Furuta pendulum. The goal is to track optimized trajectories in order to do light-painting with the tip of the pendulum using long exposure photography. Therefore, we compare trajectory optimization using the collocation and single shooting method. By formulating the loss function with time dependent RBF activations, we were able to overcome limitations of naive loss functions. To compensate for modeling errors, the optimized trajectories are tracked on the real system through linear quadratic control.

1 Introduction

In this project we aim to create pictures using long-exposure photography by attaching a light source at the tip of the Quanser QUBE, (Figure 1), a Furuta pendulum. This robot is supposed to be controlled in a way so that letters can be seen in the final picture. To accomplish this, we first created a simulator based on the equations of motion (EoM) of a Furuta pendulum as well as a corresponding 3D Model to visualize its behavior. Next, we applied optimal control techniques like direct single shooting and direct collocation to plan trajectories to reach prespecified points. As the straight forward static specification of via points does not allow us to succeed with our task, we came up with a time-dependent formulation based on a loss function activated by radial basis functions (RBFs). The generated trajectories do not lead to the expected results when applied on the robot because the model is not precise enough for open-loop control. We stabilize the execution on the robot by applying closed-loop control through a linear-quadratic regulator (LQR).

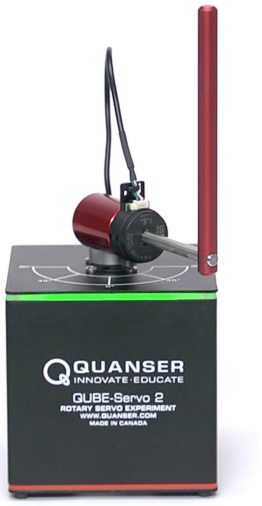


Figure 1: Quanser QUBE [4]

2 Related Work

The dynamics of a Furuta pendulum have been stated in multiple papers. For a derivation of its EoM, see [3]. A better understanding about the topic and methods we use for optimal control can be acquired from [6]. For formulating and solving the nonlinear programming (NLP) optimization problems, we use the symbolic computer algebra system with automatic differentiation (CasADi), see [2] for further information. This document also contains the

*equal contribution

application of the single shooting and collocation methods for trajectory optimization. To work with the real system, a controller needs to be applied to compensate for deviations from the model and to stabilize the robot around the desired trajectory. Previous papers like [7] and [1] came to the result, that the use of a LQR can give satisfying results for underactuated systems in general and also for a Furuta pendulum.

3 Model and Simulation

To prototype different trajectory optimization algorithms and to validate the model, we implemented a simulator for a Furuta Pendulum, particularly the Quanser QUBE [4].

3.1 Notation

For reference, we use the following mathematical notations

N	number of time steps
$h = \mathrm{d}t$	length of one time step
$k \in \{0, ..., N\}$	time discretization
$\mathbf{x}_k$	state at time step k
u_k	control at time step k
$\mathbf{f}(\mathbf{x}_k, u_k)$	system dynamics at time step k

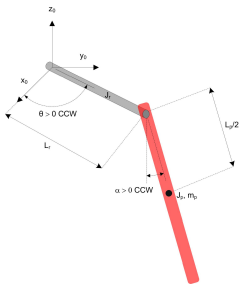


Figure 2: Furuta pendulum model [5]

3.2 Furuta Pendulum Model

The EoM can be derived using the Euler-Lagrange method [3]. We use slightly simplified non-linear EoM and assume small arm radii in comparison to their length. The resulting EoM are given as

$$\left(m_p L_r^2 + \frac{1}{4} m_p L_p^2 - \frac{1}{4} m_p L_p^2 cos(\alpha) + J_r\right) \ddot{\theta} + \left(\frac{1}{2} m_p L_p L_r cos(\alpha)\right) \ddot{\alpha} + \left(\frac{1}{2} m_p L_p^2 \sin(\alpha)\cos(\alpha)\right) \dot{\theta}\dot{\alpha} - \left(\frac{1}{2} m_p L_p L_r \sin(\alpha)\right) \dot{\alpha}^2 + D_r \dot{\theta} = \tau, \tag{1}$$

$$\left(\frac{1}{2} m_p L_p L_r \cos(\alpha)\right) \ddot{\theta} + \left(J_p + \frac{1}{4} m_p L_p^2\right) \ddot{\alpha} - \left(\frac{1}{4} m_p L_p^2 \cos(\alpha)\sin(\alpha)\right) \dot{\theta}^2 + \frac{1}{2} m_p L_p g \sin(\alpha) + D_p \dot{\alpha} = 0, \tag{2}$$

with the motor torque

$$\tau = \frac{k_m (V_m - k_m \dot{\theta})}{R_m}, \tag{3}$$

where V_m is controlled. The forward dynamics in state-space form are then given as

$$\mathbf{x} = \begin{bmatrix} \theta & \alpha & \dot{\theta} & \dot{\alpha} \end{bmatrix}^T, \tag{4}$$

$$\mathbf{f}(\mathbf{x}, u) = \dot{\mathbf{x}} = \begin{bmatrix} \dot{\theta} & \dot{\alpha} & \ddot{\theta} & \ddot{\alpha} \end{bmatrix}^T. \tag{5}$$

3.3 Point Projection to create Letters

We want to write letters with the tip of the pendulum using long exposure photography. But how do we know what states the joints have to be in, in order to create a trajectory in the form of a letter? To answer this, we first have to analyse the reachable space of the pendulum and then calculate the projection between the camera space, a 2D plane, and the pendulum's state space.
We plotted the reachable space using a point cloud sampled from different values of the joint space in Figure 3. It is a sphere with cut off bottom and top with just a small area at the back that is not

Table 1: Quanser QUBE System Parameters

Symbol	Description	Value
R_m	Terminal resistance	$8.4\,\Omega$
k_m	Motor back-emf constant	$0.042\,{}^{\text{Vs}}/_{\text{rad}}$
m_r	Rotary arm mass	$0.095\,\text{kg}$
L_r	Rotary arm length	$0.085\,\text{m}$
m_p	Pendulum link mass	$0.024\,\text{kg}$
L_p	Pendulum link length	$0.129\,\text{m}$
J_r	Rotarty arm inertia around the center of mass	$Mr \cdot Lr^2/12$
J_p	Pendulum link inertia around the center of mass	$Mp \cdot Lp^2/12$
D_r	Rotary arm damping coefficient	$0.0005\,{}^{\text{Nm s}}/_{\text{rad}}$
D_p	Pendulum link damping coefficient	$0\,{}^{\text{Nm s}}/_{\text{rad}}$

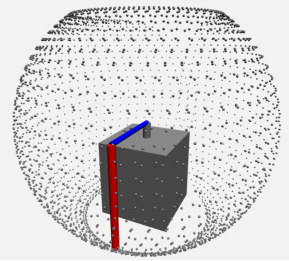


Figure 3: Reachable space

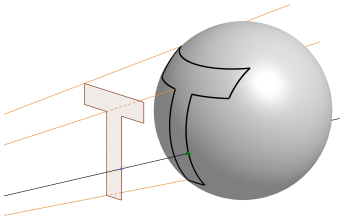


Figure 4: Projection visualization of the letter 'T'

reachable due to the boundary on θ. Mathematically expressed:

$$\begin{aligned} r &= \sqrt{L_r^2 + L_p^2}, \\ \text{s.t. } |z| &\leq L_p \\ |\theta| &\leq \theta_{max}. \end{aligned}$$

For the projection from the camera plane to the robots reachable space we place the camera in front of the pendulum parallel to its y-z-plane to make the projection as simple as possible. The intersection between the camera space and the pendulum's space can than be computed through

$$\mathbf{p} = \mathbf{p}_{\text{camera}} + k \left(\begin{bmatrix} 0 \\ y_{\text{proj}} \\ z_{\text{proj}} \end{bmatrix} - \mathbf{p}_{\text{camera}} \right).$$

Figure 4 shows how a letter is projected onto the reachable space of the pendulum, here graphically simplified as sphere.

4 Optimal Control

The optimal control problem we are facing can be divided in two parts. First, we have to plan and optimize a trajectory with respect to desired properties and constraints (e.g. reaching a specific point). Therefore, we have to find the output actions that can generate this trajectory. But simply applying these actions to the real robot is not feasible because of the chaotic nature of the robot and deviations in our model. To track and stabilize the trajectory on the real system, we apply a linear quadratic controller which is supposed to compensate for deviations.

4.1 Trajectory Optimization

There are numerous approaches on how to formulate a trajectory optimization problem depending on the properties of the desired goal. Here, we focus on direct optimization methods, more explicitly direct single shooting and direct collocation, as described in [2], [6].

The *CasADi* framework exposes an interface for NLP solvers, which solve problems of the form

$$\begin{aligned} &\text{minimize } J(x,p) \\ &\text{s.t. } x_{lb} \leq x \leq x_{ub} \\ &\quad g_{lb} \leq g(x,p) \leq g_{ub} \end{aligned}$$

where x is the decision variable and p are the known parameters [2]. NLP refers to optimization problems that have a nonlinear objective function and nonlinear constraints. With CasADi these can be formulated in a symbolic way and then be optimized by a standard optimizer like e.g. *IPOPT*[2]. We also came up with a formulation for the loss function based on RBF activations, which suits our goal better than the obvious choices that come with the aforementioned optimization problem formulations.

4.1.1 Direct Single Shooting

The first type of formulation we applied to our problem was single shooting. It is based on a recursive definition of the state for each point in time:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\dot{\mathbf{x}}_k(\mathbf{x}_k, u_k) \quad \forall\, k \in 0 \ldots N-1 \tag{6}$$

Where $\dot{\mathbf{x}}_k$ is determined by our model of the robot. This resembles the EULER-Integration of the state in a symbolical way, to yield the state trajectory based on an initial state $\mathbf{x}_0$. We chose the EULER integration for sake of simplicity here but we were also able to employ RUNGE-KUTTA (RK) integration with single shooting to model the state trajectory with more precision. The result of iterating (6) k times is the state $\mathbf{x}_{k+1}(\mathbf{x}_0, u_{0:k})$ which depends on the initial state and all previous actions.

From this, we can now formulate a simple objective function that penalizes the distance at the final state $\mathbf{x}_N$ to the desired state $\mathbf{x}^d$ and constraints that restrict the state and action spaces to match the real robot. It turns out that the method works well and that the control sequence u lets the simulator reach a single state at a predefined time step very precisely, as long as the state is reachable in the desired number of time steps.

The problem arising with using a desired state as objective formulation is that we have to chose the length of the trajectory prior to the optimization. Because of this, the trajectories can not be considered optimal with respect to our goal as there may be some trajectory that also reaches the point but e.g., takes less time. This problem is tackled in Section 4.1.3 where we introduce a cost formulation that allows to additionally optimize the time of arrival.

4.1.2 Direct Collocation

In this section we will briefly explain the collocation method and how we apply it to our optimization problem. The most profound difference between collocation and single shooting is, that it takes the states along the trajectory as additional decision variables. In addition to that, the trajectory is modeled by chaining intervals constructed from Lagrange polynomials which fit the states at given collocation points. Because the Lagrange polynomials are differentiable by definition, the trajectory can be tied to the model of the robot by forcing their derivatives to fit the *EoM* at each collocation point via constraints. These constraints are also called collocation equations [2]. To ensure the chain of intervals yields a viable trajectory, the last state of each interval is forced to equal the first point of the next interval. These constraints are labeled as continuity equations.

In this scenario all states are decision variables. It is straightforward to add constraints such that a certain desired point is reached at a given time. Therefore, we can simply add constraints at different points in time to reach desired states. But despite being able to incorporate multiple via-points by adding constraints on the state we still can not know in advance what points in time are optimal or even feasible. To address this issue we derived a new cost formulation.

4.1.3 RBF Based Cost Formulation

We enabled the optimizer to chose (and thus optimize) the time at which a certain via-point should be reached. Since the system is underactuated, it takes thorough planning to reach points. This evokes

[2] An open source library for nonlinear optimization. `https://projects.coin-or.org/Ipopt`

the need to exploit the properties of our goal, especially, it requires to only somehow reach the point at some predictable time. With this formulation a via point's loss contributes to the overall loss only when it is activated by its time dependent RBF. This allows the optimizer to chose a variable time point at which a via point should be reached. Thus the loss of all via points is invariant with respec to states where the loss of none of the via points is activated. This increased freedom allows the optimizer to e.g., accumulate energy and prepare for reaching certain points.

To allow the optimization of time points, a new set of decision variables is introduced:

$$\mathbf{t} = \begin{bmatrix} t_0 \\ \vdots \\ t_{n_{via}-1} \end{bmatrix}, 0 \leq t_i \leq 1 \; \forall \; i \in 0 \ldots (n_{via}-1), \quad \mathbf{V} = \begin{bmatrix} \mathbf{v}_0 \\ \vdots \\ \mathbf{v}_{n_{via}-1} \end{bmatrix} \tag{7}$$

The variable time-points t_i are restricted to be between 0 and 1 and define the phase which is a portion of the whole trajectory with 0 being the first time step and 1 being the last one. The given desired via-points $\mathbf{v}$ are used to calculate the loss which is then weighted by the RBF activations depending on the temporal difference between the selected time t and each time step of the trajectory:

$$J_{via} = \sum_{k=0}^{n-1} \sum_{j=0}^{n_{via}-1} \varphi\left(t_j, \frac{k}{n-1}\right) \cdot d\left(\mathbf{x}_k, \mathbf{v}_j\right), \tag{8}$$

where in the simplest case the RBF-activation-function φ is defined as

$$\varphi\left(t, s\right) = e^{-\left(\frac{t-s}{\sigma}\right)^2}, \tag{9}$$

and $d(\mathbf{x}_k, \mathbf{v}_j)$ being a metric.

The RBF for each via-point is centered around the time the optimizer choses for that particular via-point. The bandwidth parameter σ allows to restrict ($\sigma > 1$) or expand ($\sigma < 1$) the temporal extent of the RBFs. This relaxes or tightens the restrictions around the via points. The *LossFunction* for the trajectory states and the particular via-point is activated in a time-frame (with a size depending on σ) around this point. With more temporal distance the loss connected to that particular via-point becomes less relevant.

From our point of view, this formulation is more favorable because it is differentiable and we get the chosen (optimal) time points $\mathbf{t}$ as a result of the optimization. With this approach it is also possible to establish ordering constraints on the via-points. The time values of $\mathbf{t}$ can also be used to penalize late arrival at via-points. The combined loss can be written as

$$J_{combined} = J_{via} + \alpha \cdot \sum_{j=0}^{n_{via}-1} t_j. \tag{10}$$

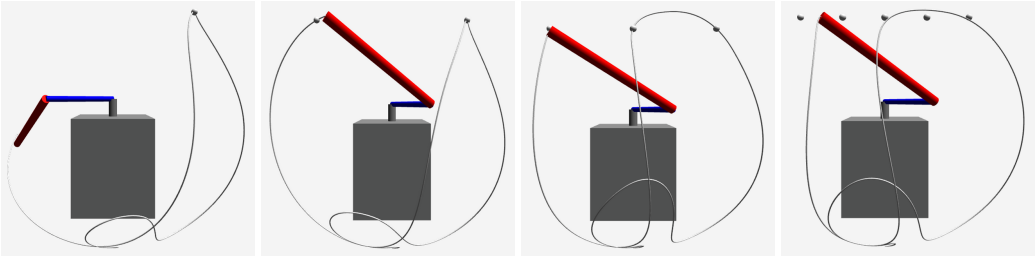
It appears that the solution of the optimization is very sensitive to the choice of α and that its tuning is a non-trivial endeavor. Nevertheless, very satisfying solutions may arise, see Figure 5, 8.

In all cases the $LossFunction$ can be defined in either task- or joint-space. In our experiments with the simulator this did not make a significant difference but using the task-space loss tends to produce trajectories that resemble the expected behavior with respect to our goal a bit better. Moreover, using the task-space loss also seems to have a slight impact on the performance as the optimizer needs to evaluate the forward pass through the kinematic chain multiple times. For latter results, we used squared task space loss.

In Figure 5 the result of four different optimization problems can be seen. Each of them is based on a prespecified set of points in task-space that should be reached within that trajectory. The aforementioned loss formulation J allows the optimizer to chose when to reach them, which can be observed in the first picture where only one via-point has to be passed. The optimizer choses to reach the point at $t_0 \approx 0.58$ which corresponds to step $t_0 \cdot step\_count \approx 58$. This can also be observed in Figure 6, where the raw loss and the RBF activated loss are shown over time. The loss without activation would be quite high even though the robot reaches the point very precisely. The

RBF activation around t_0 applied to this loss signal gives a more precise loss with regard to our goal. It is invariant to the state of the robot at most time points, but ensures that at least at some point the squared task-space loss needs to be taken into account.

For *LossFunctions* with more via-points, given the fixed boundary conditions (collocation with 100 steps, $\mathrm{dt} = 0.01$, collocation points [0, 0.1, 0.5, 0.8, 0.9] and $\alpha = 10^{-7}$) the trajectories share some similarities but also differ conceptually. We can observe that the trajectory for two via-points is more or less an extension for the first one. With three via-points this gets extended so that a third point in the middle is reached while still maintaining high precision. With five via-points this behavior changes and the optimizer stops reaching the points exactly. This can be attributed to the constrained trajectory length and the choice of α. Figure 7 shows the corresponding raw and activated loss values over time. It can be seen that the first three points are activated in short succession while the last two points are activated at the same time, just at the end. This behavior can be tweaked by choosing other parameters while giving us a dynamic framework for reaching very distinct sets of points. In our opinion, there is also no way for a human to estimate this behavior beforehand so we see this formulation as an essential baseline to build further steps of abstraction upon.



(a) One Via-Point (b) Two Via-Points (c) Three Via-Points (d) Five Via-Points

Figure 5: Trajectories generated by collocation with RBF activated loss

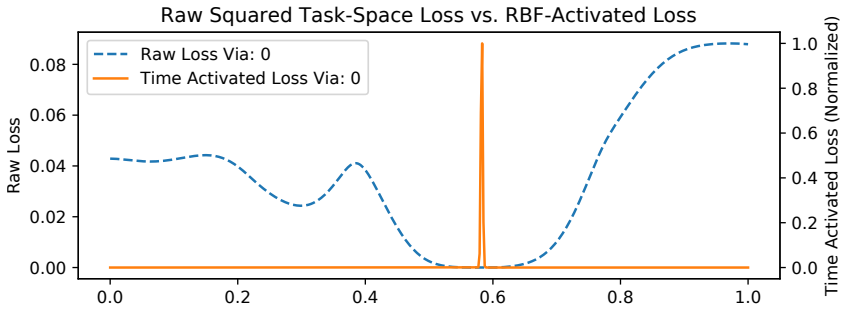


Figure 6: Comparing the raw squared task-space loss to activated loss for one via point given its the chosen time point

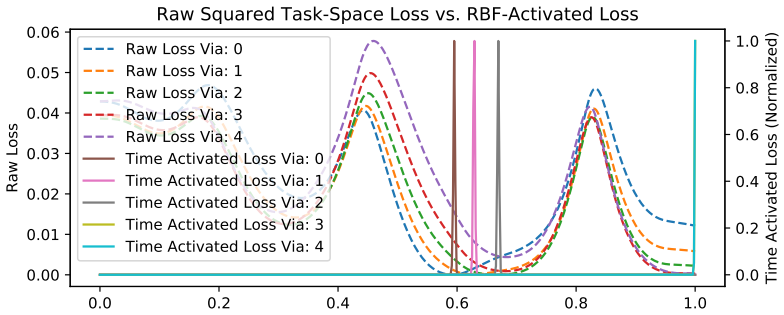
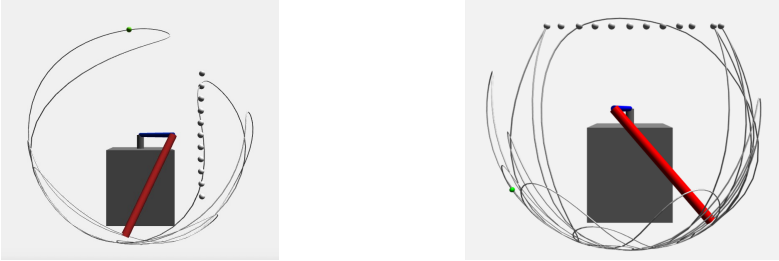


Figure 7: Comparing the raw squared task-space loss to activated loss for multiple via points given their chosen time points

With an increasing number of via points the optimization problem gets more complex. To reduce run time and stabilize convergence, we came up with the idea of chaining multiple optimization problems together. The simplest case is to split the optimization problem in two parts. The first one uses the RBF activated loss function to pass through the given via points leaving out the inital condition on the state. The second part optimizes a trajectory from the resting position to the chosen initial state of the afore mentioned optimization.

The simulated results for two different trajectory optimization problems can be seen in Figure 8. The white dots show the desired trajectory which was discretized through multiple points and the green one is the chosen initial state of the first optimization problem.



(a) Vertical line (b) Horizontal line

Figure 8: Passing through Multiple Points using the RBF activated loss function and splitting of the optimization problem

4.2 Trajectory-tracking Using Linear Quadratic Control

After successful passing through multiple points in simulation, the results need to be verified on the Quanser QUBE (Figure 1). But executing the same commands to the real system as to the simulator results in deviation. We need a controller which compensates the error and tracks the desired trajectory through local stabilization. For that, we use a finite-horizon discrete-time LQR. The dynamics of the Furuta Pendulum are non-linear, therefore we have to linearize the dynamics around an equilibrium $(\mathbf{x}^d, u^d)$ using the TAYLOR expansion and the difference $\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}^d$ as new state coordinate. $\mathbf{x}$ is the feedback state of the real system.

$$\begin{aligned}\dot{\tilde{\mathbf{x}}} &= \dot{\mathbf{x}} - \dot{\mathbf{x}}^d = \dot{\mathbf{x}} - \mathbf{f}(\mathbf{x}^d, u^d) \\ &= \frac{\partial \mathbf{f}(\mathbf{x}^d, u^d)}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}^d) + \frac{\partial \mathbf{f}(\mathbf{x}^d, u^d)}{\partial u}(u - u^d) \\ &= \mathbf{A}\tilde{\mathbf{x}} + \mathbf{B}\tilde{u}.\end{aligned}$$

The quadratic cost function in discrete form is defined as

$$J = \mathbf{x}_N^T \mathbf{Q} \mathbf{x}_N + \sum_{k=0}^{N-1} (\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + u_k^T R u_k). \qquad (11)$$

The optimal feedback control sequence which minimizes the cost function 11 is given by

$$\begin{aligned}\tilde{u}_k &= -\mathbf{K}_k \tilde{\mathbf{x}}_k, \\ u_k &= u_k^d - \mathbf{K}_k(\mathbf{x}_k - \mathbf{x}_k^d),\end{aligned}$$

where

$$\mathbf{K}_k = (R + \mathbf{B}_k^T \mathbf{P}_{k+1} \mathbf{B}_k)^{-1} (\mathbf{B}_k^T \mathbf{P}_{k+1} \mathbf{A}_k).$$

$\mathbf{P}_k$ is found by solving the Riccati equation iteratively backwards in time:

$$\mathbf{P}_{k-1} = \mathbf{A}_k^T \mathbf{P}_k \mathbf{A}_k - (\mathbf{A}_k^T \mathbf{P}_k \mathbf{B}_k)(R + \mathbf{B}_k^T \mathbf{P}_k \mathbf{B}_k)^{-1} (\mathbf{B}_k^T \mathbf{P}_k \mathbf{A}_k) + \mathbf{Q}$$

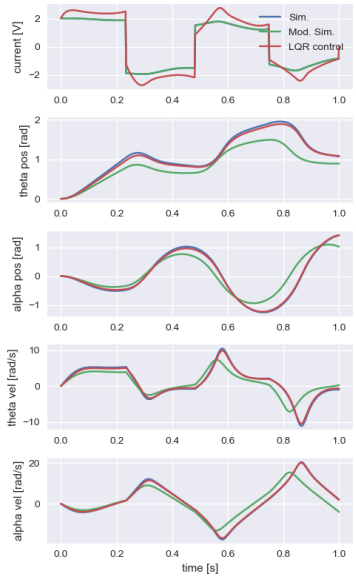


Figure 9: LQR on modified simulator

with the terminal condition $\mathbf{P}_N = \mathbf{Q}$.

Before applying the LQR controller on the Quanser QUBE, we tested and validated its behavior on the simulator. Therefore, we created a trajectory by executing some actions on our simulator. To model the behavior of a real system we created a modified simulator by changing some parameter values. Executing the same actions on this system, we get a diverging trajectory. Applying the LQR with $\mathbf{Q} = \mathbf{1}$ and $R = 1$ to our modified simulator, the original simulator's trajectory can be tracked (see Figure 9).

5 Photography

In order to evaluate the trajectory in terms of long exposure photography, a camera sensor is simulated. The basic idea is to have a matrix in which every element corresponds with a virtual pixel of the camera sensor. A simple way to do it is to use the optical model of a pinhole camera (see:[8]).

Taking long exposure pictures (also called "light-painting") requires the tip to be illuminated. Attaching a light source to the tip is not feasible because cables affect the robots behavior too much. Likewise, a battery powered source would be to hard to control. Thus, we decided to use a combination of reflective material and ultraviolet (UV) light to illuminate the tip. The UV-LEDs are connected to the control-pc through a serial connection over USB. We used UV-LEDs with a wavelength of 400nm. An Arduino is used to control the LEDs using pulse-width modulation. Figure 10 shows how the system is structured. The process of light-painting requires a long exposure and a dark environment. The control-pc triggers the camera and starts to execute the trajectory on the robot. Simultaneously the brightness of the UV light is adjusted because the light should be on when the tip is close to the desired trajectory.

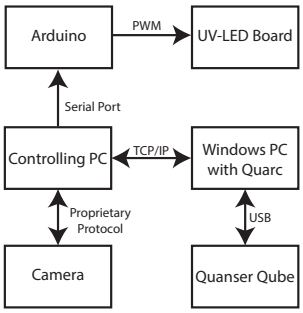


Figure 10: System Overview

6 Conclusion & Outlook

In this report, we presented how optimal control can be applied to an underactuated system, called Furuta Pendulum. We realized the optimization of simple trajectories by discretizing them and minimized the error through RBF activation in the objective function. But for big optimization problems like complete letters we regularly observed that the selected solver(s) did not converge. Therefore, we split the problem into smaller ones, or in other words, construct a letter by splitting it in multiple trajectories. This should be easier and faster to compute.
In the future, we will apply model learning to better fit the parameters of our model to behave even closer to the real system, because we still aim to construct letters on a real system using long exposure photography with the Quanser QUBE. For this, parameter optimization for the LQR could also help, to get more stable trajectories. A first look on how the long exposure photography with UV light could look like can be seen in Figure 11.

Figure 11: Long exposure photography result for a section of a swing-up trajectory

References

[1] Akash Gupta, Varnita Verma, Adesh Kumar, Paawan Sharma, Mukul Kumar Gupta, and C. S. Meera. Stabilization of Underactuated Mechanical System Using LQR Technique. In Rajesh Singh and Sushabhan Choudhury, editors, *Proceeding of International Conference on Intelligent Communication, Control and Devices*, Advances in Intelligent Systems and Computing, pages 601–608. Springer Singapore, 2017.

[2] Joel Andersson, Joris Gillis, and Moritz Diehl. User Documentation for CasADi.

[3] Ben Cazzolato and Zebb Prime. On the dynamics of the furuta pendulum. 8, 03 2011.

[4] Quanser Inc. QUBE - servo 2.

[5] Quanser Inc. QUBE-SERVO 2 workbook - student, 2013.

[6] Matthew Kelly. An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation. *SIAM Review*, 59(4):849–904, January 2017.

[7] M Siva Kumar, B Dasu, and G Ramesh. Design Of LQR Based Stabilizer For Rotary Inverted Pendulum System.

[8] Henri Matre. *From Photon to Pixel: The Digital Camera Handbook*. Wiley-IEEE Press, 2nd edition, 2017.