Master Thesis

Balancing a ball on a plate using stereo vision

Author(s):
Wettstein, Nathanael

Publication Date:
2013

Permanent Link:

https://doi.org/10.3929/ethz-a-010076634 →

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for
Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme
und Regelungstechnik

Nathanael Wettstein

Balancing a Ball on a Plate using Stereo Vision

Master Thesis

Institute for Dynamic Systems and Control
Swiss Federal Institute of Technology (ETH) Zurich

Supervision

Nico Hübel
Prof. Dr. Raffaello D'Andrea

April 2013

IDSC-RD-NH-19_Balancing_Ball_on_Plate

Contents

Abstract

In this work, we investigate the dynamics of a ball balancing on a plate that is mounted onto a robot arm. We obtain the nonlinear system dynamics, which are then linearized to design an LQR controller. We demonstrate in experiments that we can stabilize the ball in presence of measurement noise.
The goal of this ongoing project is to balance a ball on a ball. As a further step towards this goal, we model and simulate a ball on a rounded plate.

Chapter 1

Introduction

In applied control, the inverted pendulum is often used as a testbed for new algorithms and systems. The ball balancing setup is a variant of this problem serving as an intuitive demonstration of the capabilities of model-based control. This work was inspired by a video on YouTube, where a man balances three basketballs on top of each other, `http://youtu.be/DeLCc-CWTAQ`.
For the first author, the opportunity to build an educative setup to explain the possibilities of control systems to non-technical persons and potential future students provided additional motivation.

Chapter 2

Modelling

We want to balance a ball on a plate using an industrial robot arm. In this chapter, we describe the system and derive its dynamics.

2.1 Physical setup

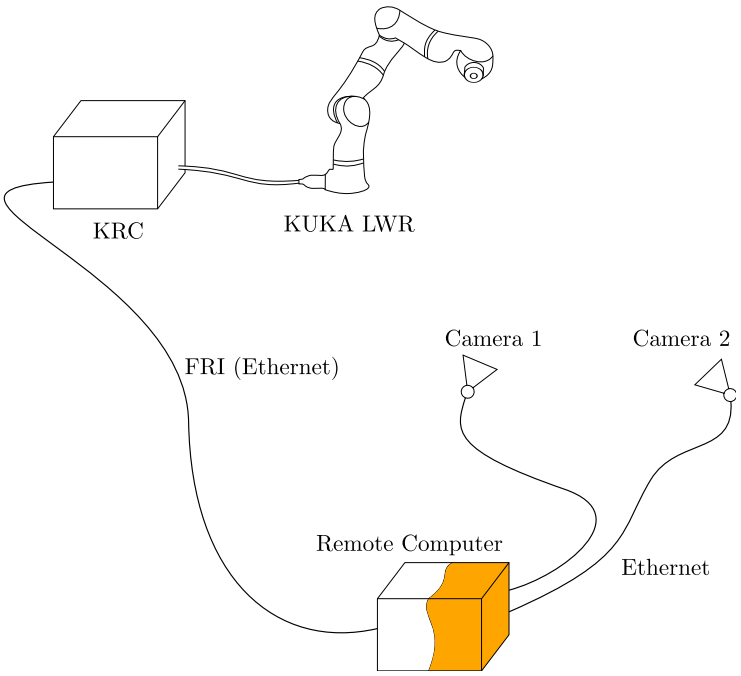

Figure 2.1: Hardware setup

The hardware setup illustrated in figure 2.1 features a 7 DoF KUKA light-weight robot (KUKA LWR, [2]). The robot arm is controlled by the KUKA Robot Controller (KRC), which provides the Fast Research Interface (FRI, [8]). This interface provides robot position and torque measurements and accepts control signals. It is implemented on a standard PC (Intel quad-core 3 GHz, 4 GB

RAM, running 64bit Ubuntu 10.04) by OROCOS [3] components. An instance of the ball controller implemented in OROCOS runs on this same computer.
Two color cameras (Prosilica GC655C, operated at 50 Hz) observe the scene and are attached via a Gigabit Ethernet connection to the PC, where the images are processed by a ROS[4] component. More details about the hardware components can be found in [5, Chapter 1.2].
A schematic overview of the system is given in figure 2.2. We are going to refer to this overview in the following sections.

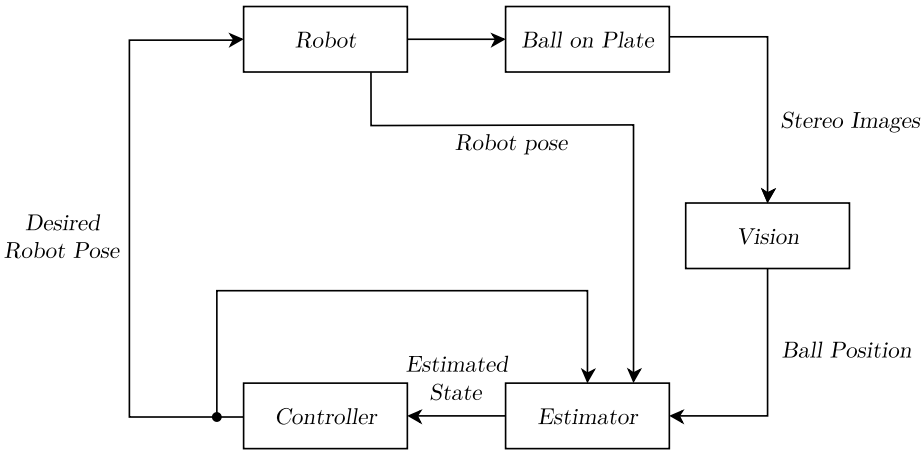

Figure 2.2: Schematic overview

2.2 Definitions and Assumptions

2.2.1 Coordinate systems

We use the following coordinate systems.

World The world coordinate system is an inertial system that describes the world our robot lives in. For simplicity, we let the origin of the world coordinate system coincide with the origin of the robot base coordinate system w.l.o.g.

Plate The plate coordinate system is centered at the surface in the middle of the plate. The x- and y-axis are parallel, the z-axis orthogonal to the plate surface. This coordinate system moves and rotates together with the plate.

Camera Each of the two camera observes the ball position in its own camera coordinate system. The resulting images are fused to the 3D ball position in robot coordinates. During the initial system calibration, the two camera transformations are determined. A detailled description of this process can be found in [5].

2.2.2 State definition

When modelling the physical system, we will use the following states as illustrated in figure 2.3. Ball positions and velocities are given in plate system P, plate states in world system W.

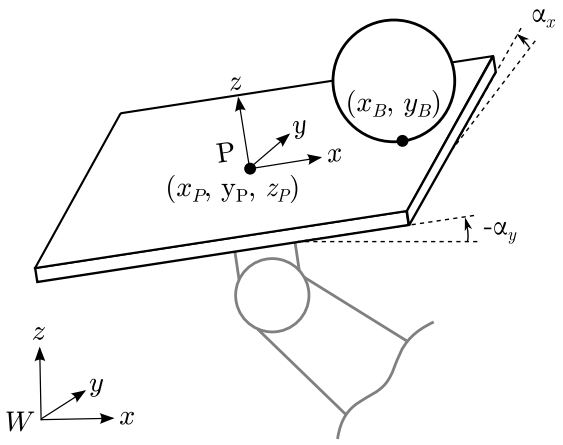

Figure 2.3: Ball on plate

- x_B, y_B: ball position
- $\dot{x}_B, \dot{y}_B$: ball velocity
- x_P, y_P, z_P: plate linear position
- $\dot{x}_P, \dot{y}_P, \dot{z}_P$: plate linear velocity
- $\alpha_x, \alpha_y, \alpha_z$: plate orientation (No-spin: $\alpha_z \equiv 0$)
- $\dot{\alpha}_x, \dot{\alpha}_y, \dot{\alpha}_z$: plate rotational velocity (No-spin: $\dot{\alpha}_z \equiv 0$)

For ball position control, we will use the homogenous coordinate transformation according to [1] from world coordinates x^W to plate coordinates x^P:

$$x^P = T_W^P x^W$$

where

$$T_W^P = \begin{bmatrix} R_W^P & Q \\ 0 & 1 \end{bmatrix} \text{ and } x = \begin{bmatrix} x_x \\ x_y \\ x_z \\ 1 \end{bmatrix}$$

where R_W^P is the 3x3 rotation matrix from W to P using the z-y'-x" convention of Euler angles, and Q is a 3x1 vector that describes the translation from the origin of W to the origin of T.

2.2.3 Physical properties

We define the following physical properties:

- Ball radius: r_B $[m]$
- Ball mass: m_B $[kg]$
- Ball moment of inertia: $I_B = I_{Bx} = I_{By} = I_{Bz}$ $[kgm^2]$
- Plate radius: r_P $[m]$ (in case of flat plate $r_P \to \infty$)
- Plate mass: m_P $[kg]$
- Plate moment of inertia: $I_P = I_{Px} = I_{Py}, I_{Pz}$ $[kgm^2]$

The ball (and the plate, respectively) is symmetrical with respect to the z-axis, so the moments of inertia in x- and y-direction are equal. Because of the no-spin assumption we do not consider moments of inertia in z-direction. The moment of inertia of the plate was determined using the available CAD model. The moment of inertia of a spherical shell is given as

$$I_B = \frac{2}{3} m_B r^2 \tag{2.1}$$

Due to the no-slip assumption, the ball's rotational position and velocity can be directly related to its linear velocity relative to the plate:

$$\begin{bmatrix} x_B \\ y_B \end{bmatrix} = r_B \begin{bmatrix} \theta_{By} \\ -\theta_{Bx} \end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix} \dot{x}_B \\ \dot{y}_B \end{bmatrix} = r_B \begin{bmatrix} \dot{\theta}_{By} \\ -\dot{\theta}_{Bx} \end{bmatrix} \tag{2.3}$$

2.2.4 Neglected time delays

We will not consider time delays in our model. Note that we do take into account the CV computation time. Sources of time delays in the system include

- camera frame rate
- controller computation time
- communication between ROS/OROCOS and FRI
- FRI internal time delay
- robot internal time delay

2.2.5 Robot compensating nonlinearities

We will model the robot dynamics as a linear, time-invariant (LTI) system. While this is an approximation, previous work on the KUKA LWR has shown that the internal robot controller is able to compensate for nonlinearities of its own dynamics very well. Furthermore, we assume that the robot takes into account the dynamics of the attached plate. To this end, the plate parameters (weight, moments of inertia) are configured in the KRC.
We will also assume that the effect of gravity acting on the ball does not influence the plate, meaning, this effect is compensated by the robot.

2.2.6 Ball on the plate assumptions

We make the following assumptions concerning the ball on the plate:

- No-slip: The ball always rolls on the surface and does not slip, i.e. infinite static friction.
- No-friction-loss: There is no rolling friction between ball and surface. The ball does not deform. No heat is generated.
- No-fly: The ball does not loose contact with the plate, i.e. the plate's acceleration in negative z-direction is small.
- No-flip: The plate does not flip over (i.e. $\alpha_x, \alpha_y \in [-\frac{\pi}{2}, \frac{\pi}{2}]$).
- No-spin: There is no rotation around the z-axis of either the ball or the plate.

2.3 Robot model

2.3.1 How to control the robot via FRI

Using the FRI, we can control the KUKA LWR robot arm using tree different control strategies:

- Position Control ("hard", joint position)
- Cartesian Impedance Control ("soft")
  - Cartesian position
  - Cartesian twist (linear/rotational velocity)
  - Cartesian wrench (force, torque)
- Joint Specific Impedance Control ("soft", joint torques)

In this work, we focus on the impedance control strategy which allows human interaction with the robot and allows an easier modelling of the robot and ball dynamics than the joint specific impedance control strategy. Human interaction is successfully illustrated by the operator pushing and pulling the robot while it is balancing the ball.
In cartesian impedance strategy, the FRI component may send either cartesian positions [cmd.data.cartPos] or cartesian wrench [cmd.data.addTcpFT] to the robot. The twist interface is simulated by integrating the commands over the sample time and sending positions. See LWR FRI source code, `https://github.com/IDSCETHZurich/trajectory-generator`, FRIComponent.cpp. We will therefore use the cartesian position interface in our modelling.
The cartesian wrench interface was not found to be useful for this application as it did not behave as expected, with specifications largely missing.

2.3.2 Inner robot model

Our measurements suggest that the robot follows our input positions very closely while showing some delay due to its dynamics. We therefore model the robot by an integrator chain, with an inner control loop. By tuning this inner loop, we are able to reproduce the robot dynamics. This model can be seen in figure 2.4, where the matrix K is the internal controller, the matrices N and C resize the input and output, respectively.

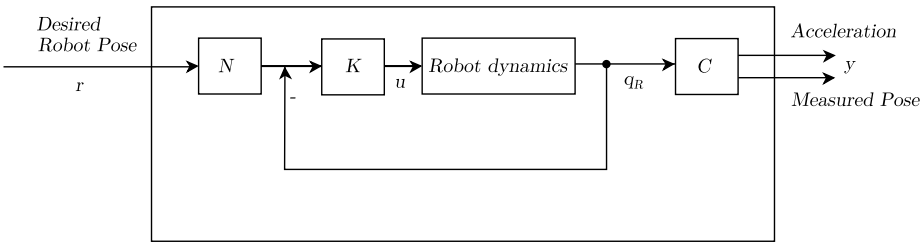

Figure 2.4: Robot inner model

We use the robot position (linear and rotational) in world coordinates as state. As the ball model depends on the plate accelerations, we use both the first and second derivative as states. Note that we always keep the plate rotation around z-axis at 0 and will therefore not include it in the state vector.
The state-space representation of the inner control loop is

$$q = \begin{bmatrix} x_P \\ y_P \\ z_P \\ \alpha_x \\ \alpha_y \end{bmatrix}; \quad q_R = \begin{bmatrix} q \\ \dot{q} \\ \ddot{q} \end{bmatrix}_{\{15x1\}} \tag{2.4}$$

$$\dot{q}_R = Aq_R + Bu = \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix} q_R + \begin{bmatrix} 0 \\ 0 \\ B_3 \end{bmatrix} u \tag{2.5}$$

$$u = -Kq_r + KNr \tag{2.6}$$

where u is the control input. We now find the linearized robot model of the state-space form, where r is the input to the robot system, i.e. the desired robot pose.

$$\dot{q}_R = A_R q_R + B_R r = (A - BK)q_R + BKNr \tag{2.7}$$

$$y = C_R q_R = \begin{bmatrix} I & 0 & 0 \\ 0 & 0 & I \end{bmatrix} q_R \tag{2.8}$$

Matrix dimensions are $A_{R\{15x15\}}$, $B_{R\{15x5\}}$ $C_{R\{10x15\}}$, $K_{\{15x15\}}$, $N_{\{15x5\}}$.

2.4 Ball on plate model

We are now going to model the ball on a plate as illustrated in figure 2.5.

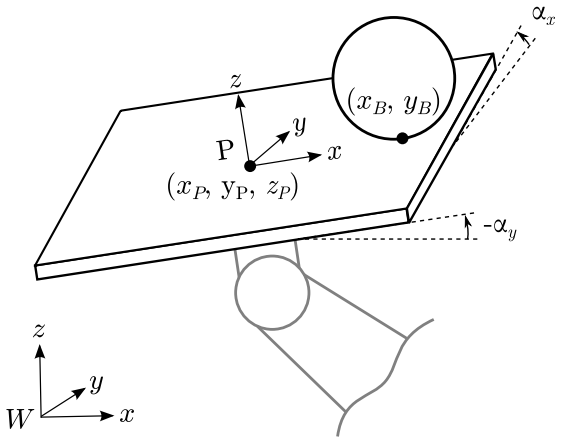

Figure 2.5: Ball on plate

2.4.1 Lagrangian

We use the method of Euler-Lagrange to obtain the dynamic equations of the system. The Lagrange equation of the second kind is given as

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} = 0 \tag{2.9}$$

We calculate the Lagrangian $L = T - V$ of the system using the kinetic and potential energy of the ball and the plate. The kinetic energy of the plate $T_{P,lin}$, $T_{P,rot}$ depends on the plate's linear movement in world coordinates, and its rotational movement around the center of mass. The potential energy of the plate V_B depends on the plate z-position in world coordinates. The same principle applies for the ball, where we also take into account the movement of the reference frame, i.e. the plate movement.

$$L = T_{P,lin} + T_{P,rot} - V_P + T_{B,lin} + T_{B,rot} - V_B \tag{2.10}$$

We define the plate linear position and velocity:

$$P_P = \left[x_P, y_P, z_P\right]$$
$$\dot{P}_P = [\dot{x}_P, \dot{y}_P, \dot{z}_P]$$

the plate rotational position and rotational velocity (assuming no rotation around z-axis),

$$v_P = \left[\alpha_x, \alpha_y, 0\right]$$
$$\omega_P = \left[\dot{\alpha_x}, \dot{\alpha_y}, 0\right]$$

and the rotation matrix from plate to world with it's time derivative,

$$R_P^W = \begin{bmatrix} \cos(\alpha_y(t)) & \sin(\alpha_x(t))\sin(\alpha_y(t)) & \cos(\alpha_x(t))\sin(\alpha_y(t)) \\ 0 & \cos(\alpha_x(t)) & -\sin(\alpha_x(t)) \\ -\sin(\alpha_y(t)) & \cos(\alpha_y(t))\sin(\alpha_x(t)) & \cos(\alpha_x(t))\cos(\alpha_y(t)) \end{bmatrix}$$

We define the ball linear position and velocity in plate coordinates,

$$P_B = \begin{bmatrix} x_B, y_B, r_B \end{bmatrix}$$
$$\dot{P}_B = \begin{bmatrix} \dot{x}_B, \dot{y}_B, 0 \end{bmatrix}$$

the ball position in world coordinates,

$$P_B^W = P_P + R_P^W P_B$$
$$\dot{P}_B^W = \dot{P}_P + R_P^W \dot{P}_B + \omega_P \times (R_P^W P_B)$$

and the ball rotational position and rotational velocity (assuming no slip, so we can link the ball rotational position to its position on the plate),

$$v_B = \begin{bmatrix} \frac{y_B}{r_B}, -\frac{x_B}{r_B}, 0 \end{bmatrix}$$
$$\omega_B = \begin{bmatrix} \frac{\dot{y}_B}{r_B}, -\frac{\dot{x}_B}{r_B}, 0 \end{bmatrix}$$

Therefore, the Lagrangian consists of the following parts.

$$T_{P,lin} = \frac{1}{2} m_P \left|\dot{P}_P\right|^2$$
$$T_{P,rot} = \frac{1}{2} I_P \left|\omega_P\right|^2$$
$$V_P = m_P g z_P$$
$$T_{B,lin} = \frac{1}{2} m_B \left|\dot{P}_B^W\right|^2$$
$$T_{B,rot} = \frac{1}{2} I_B \left|\omega_B\right|^2$$
$$V_B = m_B g \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} P_B^W$$

We use Mathematica to calculate the Lagrangian and solve for the dynamic equations of $\ddot{x}_B$ and $\ddot{y}_B$. Detailed calculations can be found in the Mathematica notebooks in the appendix B. Our non-linear ball model is now given by

$$q = \begin{bmatrix} x_B \\ y_B \end{bmatrix}; \quad q_B = \begin{bmatrix} q \\ \dot{q} \end{bmatrix} \tag{2.11}$$
$$q_B = \begin{bmatrix} \dot{q} \\ f(q_B, q_R) \end{bmatrix} \tag{2.12}$$
$$y = C_B q_B = \begin{bmatrix} I & 0 \end{bmatrix} q_B \tag{2.13}$$

2.4.2 Combined robot and ball model

We now combine the ball and robot model. Our combined state is

$$x = \begin{bmatrix} q_R \\ q_B \end{bmatrix} \tag{2.14}$$

We continue to linearize our ball model around the equilibrium point $x = \dot{x} = 0, u = 0$ to find the state space representation of the system.

$$\dot{x} = \begin{bmatrix} A_R & 0 \\ B_B C_R & A_B \end{bmatrix} x + \begin{bmatrix} B_R \\ 0 \end{bmatrix} u \tag{2.15}$$

$$y = \begin{bmatrix} C_R & 0 \\ 0 & C_B \end{bmatrix} x \tag{2.16}$$

where our control input u is the desired pose, y are robot position and ball position measurements, and

$$A'_B = \left.\frac{\partial f}{\partial q_B}\right|_{x=0}; \quad B'_B = \left.\frac{\partial f}{\partial q_R}\right|_{x=0} \tag{2.17}$$

and

$$A_B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}_{\{2x2\}}$$

$$B_B = \left[\begin{array}{ccccc|ccccc} 0 & 0 & 0 & 0 & g\zeta & -\zeta & 0 & 0 & 0 & -r_B\zeta \\ 0 & 0 & 0 & -g\zeta & 0 & -\zeta & 0 & r_B\zeta & 0 & \end{array}\right]_{\{2x10\}}$$

where

$$\zeta = \frac{m_B}{m_B + \frac{I_B}{r_B^2}}$$

2.5 Ball on rounded plate model

As a preparation for the ball on ball setup, we model and simulate a ball on a rounded plate as illustrated in figure 2.6.

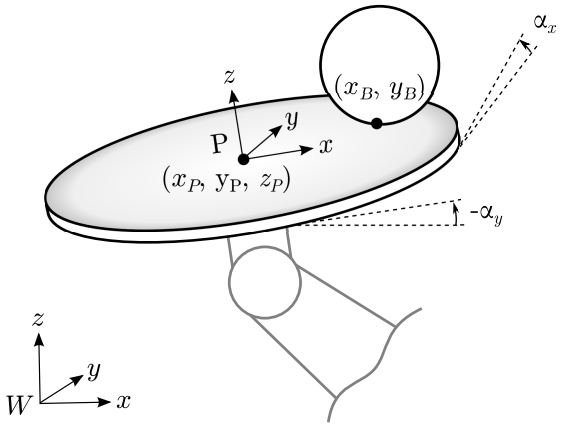

Figure 2.6: Ball on plate

We use the same approach and definitions as with the planar plate. We encounter the first difference at the calculation of the ball linear position and

velocity in plate coordinates:

$$P_B = \left[x_B, y_B, r_B + r_P - \sqrt{(r_B + r_P)^2 - x_B^2 - y_B^2}\right]$$
$$\dot{P}_B = \left[\dot{x}_B, \dot{y}_B, 0\right]$$

The resulting dynamic equations for $\ddot{x}_B$ and $\ddot{y}_B$ as calculated differ from the case "ball on plate". Detailed calculations can be found in the Mathematica notebooks in the appendix.
For the linearized model we now have:

$$A_B = \begin{bmatrix} -g\zeta\dfrac{1}{r_B + r_P} & \\ & -g\zeta\dfrac{1}{r_B + r_P} \end{bmatrix}_{\{2x2\}}$$

$$B_B = \left[\begin{array}{ccccc|ccccc} 0 & 0 & 0 & 0 & g\zeta & -\zeta & 0 & 0 & 0 & -r_B\zeta \\ 0 & 0 & 0 & -g\zeta & 0 & 0 & -\zeta & 0 & r_B\zeta & 0 \end{array}\right]_{\{2x10\}}$$

where

$$\zeta = \frac{m_B}{m_B + \dfrac{I_B r_P^2}{r_B^2 (r_B + r_P)^2}}$$

Note that if we let $r_P \to \infty$, all equations will be identical to the ones obtained for the flat plate in the previous sections.

2.6 Kalman Filter for state estimation and smoothing

As we will see in section 5.3.1, the camera measurements are noisy. This motivates us to introduce a Kalman Filter as a state estimator. The Kalman Filter provides a statistically optimal estimate of the system state. It works in two steps: In the prediction step, the Kalman Filter estimates the current state and the uncertainty (covariance) using the system model. In the update step, it updates the state and uncertainty using the obtained measurements weighted by the so-called Kalman gain.
We utilize a widely-used variant of the Kalman Filter for a non-linear system, the Extended Kalman Filter. It yields the following equations of the prediction step

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}) \tag{2.18}$$
$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_{k-1}^{\top} + \boldsymbol{Q}_{k-1} \tag{2.19}$$

and the update step

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1}) \tag{2.20}$$

$$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^\top + \boldsymbol{R}_k \tag{2.21}$$

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^\top \boldsymbol{S}_k^{-1} \tag{2.22}$$

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k \tag{2.23}$$

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k|k-1} \tag{2.24}$$

where Q and R are the process noise and measurement noise matrices. We note that our balls are not perfectly round and the weight is not exactly located in the center, while the robot compensates for its own imperfections (see assumptions in section 2.2.5). We therefore assign the process noise weights for the ball (0.02) and for the robot (0.001). From the observations described above, we assign a bigger weight to ball measurement noise (0.05) than to robot measurement noise (0.001).
The state transition and observation matrices are defined as the following Jacobians

$$\boldsymbol{F}_{k-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}} ; \qquad \boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}}$$

The qualitative results are displayed in figure 2.7. We observe oscillations in the first few seconds because the initial conditions don't match the current state. These oscillations might be further reduced by tuning the initial covariance matrix P_0.

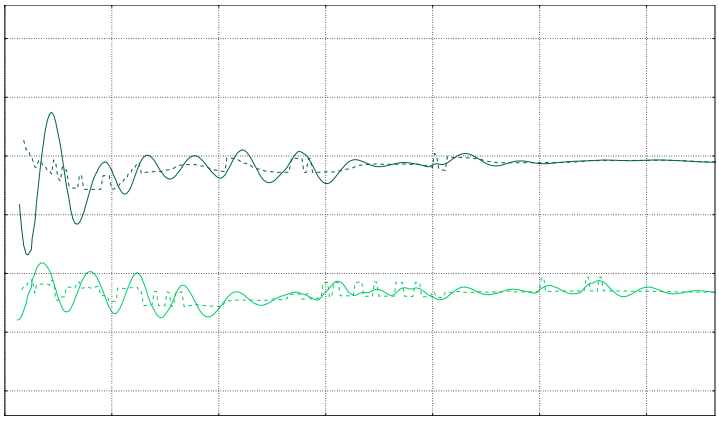

Figure 2.7: Ball position measurement (dashed line) and estimate provided by Kalman Filter (solid line).

2.6.1 Time delay compensation

As we learn in section 5.3.2, the ball position measurement by the computer vision system arrives with a delay of 3 timesteps, whereas the robot pose mea-

surement arrives in the next timestep. In order to compensate for this unequal delay, the robot pose measurement is artificially delayed (using a FIFO queue) to match the ball position measurement delay. Then, we forward simulate until the present using the update step of our Kalman Filter.
An alternative method to counter this problem would be to augment the state by including the past three states and measurements and design a new controller for this augmented system. This method is likely to be more precise but computationally more intensive.

Chapter 3

Controller design

3.1 Linear quadratic regulator (LQR)

We design a state-feedback controller as shown in figure 3.1 to stabilize the system about its equilibrium, $r(t) \equiv 0$. We will later superimpose a reference trajectory.

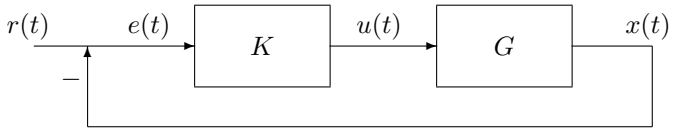

Figure 3.1: 1-DOF state feedback controller

To find the controller, we use an infinite-horizon linear-quadratic regulator (LQR) design and determine suitable weights. The LQR algorithm returns an optimal gain matrix K, such that the state-feedback law $u = -Kx$ minimizes the quadratic cost function

$$J = \int_0^\infty \left(x^T Q x + u^T R u + 2x^T N u\right) dt$$

The resulting dynamics equation is

$$\dot{x} = (A - BK)x$$

3.2 Tuning the controller

The weighting matrices Q, R, N were tuned manually. The best simulation results in terms of both settle time and accuracy of ball position were achieved using a large weight for ball position and velocity, and a small weight for the

other states and inputs. Based on our physical setup as described in 5, we find the following weighting matrices to be used in our simulations in chapter 4:

$$Q = \left[\begin{array}{cc|cccccc|cc|cccccc} 10 & 0 & & & & & & & & & & & & & & \\ 0 & 10 & & & & & & & & & & & & & & \\ \hline & & 1 & & & & & & & & & & & & & \\ & & & 1 & & & & & & & & & & & & \\ & & & & 1 & & & & & & & & & & & \\ & & & & & 1 & & & & & & & & & & \\ & & & & & & 1 & & & & & & & & & \\ & & & & & & & & 10 & 0 & & & & & & \\ & & & & & & & & 0 & 10 & & & & & & \\ \hline & & & & & & & & & & 1 & & & & & \\ & & & & & & & & & & & 1 & & & & \\ & & & & & & & & & & & & 1 & & & \\ & & & & & & & & & & & & & 1 & & \\ & & & & & & & & & & & & & & 1 & \end{array}\right],$$

$$R = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.3 \end{bmatrix},$$

$N = 0$.

3.3 Further improvements

In order to take into account input constraints, it might be beneficial to investigate other types of controllers, such as model predictive control (MPC). When we use our controller combined with a reference signal (see section 4.1.3), the use of a feed-forward term of the trajectory might be beneficial.

Chapter 4

Simulation

We first investigate how well the controller obtained in chapter 3 stabilizes the ball starting from different initial conditions.
The linearized model and the nonlinear model described in section 2.4.2 were implemented in MATLAB/Simulink and will be compared. Additionally, we will investigate the trajectory following performance of the ball and the plate. In a second step, we add a simulation of the CV time delay and compensation. In a third step, we model the ball on a rounded plate.

4.1 Linear vs. nonlinear model

4.1.1 Stabilize ball on plate without noise

We investigate the quality of our controller starting with different initial conditions, including displaced ball, rolling ball, displaced, moving and/or tilted plate. Some initial conditions are shown in figure 4.1. The goal of the controller is to bring the ball and plate to the equilibrium $x = 0$.

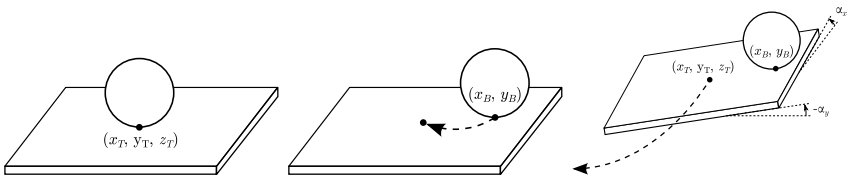

Figure 4.1: Different initial conditions used in simulations

Our results show that for all investigated initial conditions, the ball position is stabilized quickly. With the nonlinear model, there are more oscillation, some overshoot and a slightly longer settle time than with the linearized model.
As an example, figure 4.2 shows the results starting from an arbitrary initial condition. Graphs of all simulations can be found in appendix C.

4.1.2 Stabilize ball on plate with measurement noise

Starting with the same experimental setup as described above, we added Zero-Mean Gaussian White Noise (ZMGW) to the measured ball position in the order

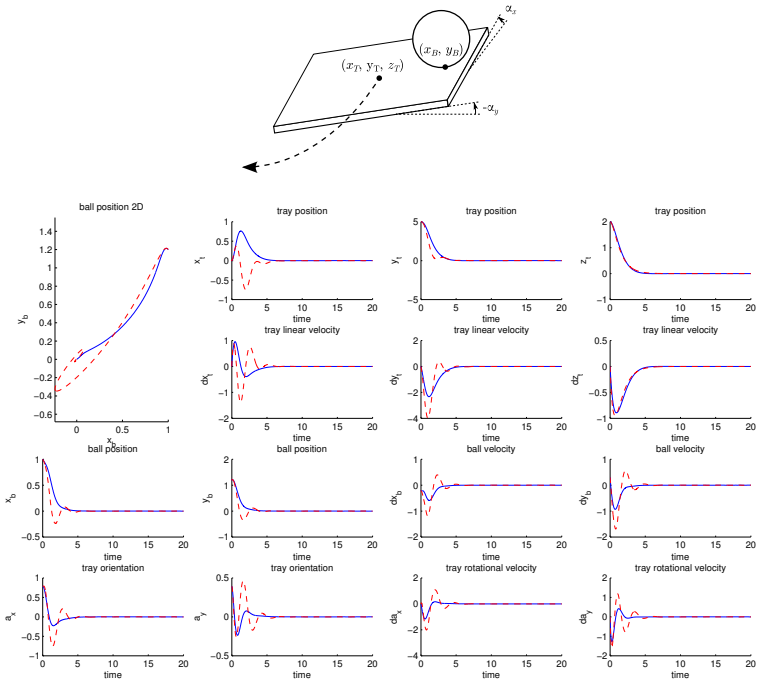

Figure 4.2: Displaced and rolling ball on tilted and displaced plate. Blue = linearized model, red (dashed) = nonlinear model

of 1-2 cm.
We find that the resulting position is indeed noisy with the deviation lying within < 1 mm. Figure 4.3 shows the simulation starting at the equilibrium initial position. The deviation is so small that it is not visible in plots where we start from another initial position.
This indicates that our controller is robust enough to handle the expected noise. Note, however, that we assume our robot is able to change the (linear and rotational) acceleration almost arbitrarily within one time step. This is yet to be shown in the experiment.

4.1.3 Reference following for ball and plate trajectories

Inspired by the flying inverted pendulum [6], we attempt to make the ball and plate follow a reference trajectory $r(t)$. We use a standard one-degree-of-freedom state feedback controller, resulting in the dynamics equation

$$\dot{x}(t) = Ax(t) - Ke(t) = (A - BK)x(t) + BKr(t) \tag{4.1}$$

We will start from the equilibrium position for all reference trajectory simulations. Different trajectories are shown in figure 4.4. In a second step, we will investigate the trajectory following performance for a trajectory for both the ball in plate coordinates, and the plate in world coordinates, as shown in figure 4.5.

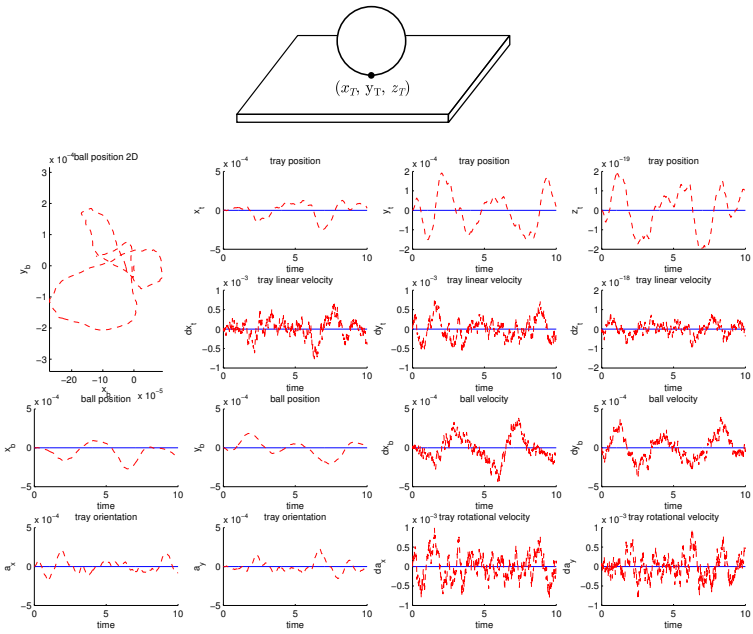

Figure 4.3: Simulation with measurement noise. Blue = linearized model, red (dashed) = nonlinear model noisy measurement

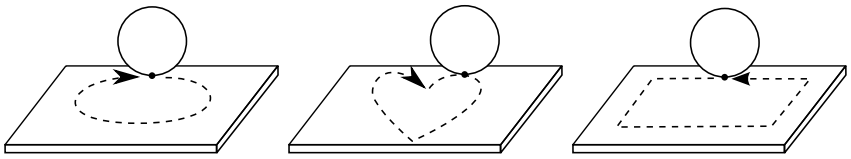

Figure 4.4: Different trajectories used in simulations

We find that the resulting position follows the circular trajectory quite closely, both for the linearized and the nonlinear model. This can be seen in figure 4.6 showing the results for a circular trajectory starting from the equilibrium initial position. We can even set a less smooth trajectory (e.g. rectangle or heart-shape), as shown in figure 4.7. However, if we increase the trajectory speed, the nonlinear system eventually becomes unstable.
As a further step, if we impose a circular trajectory for both the ball and the plate, we observe stable results. All graphs can be found in the appendix C.2.

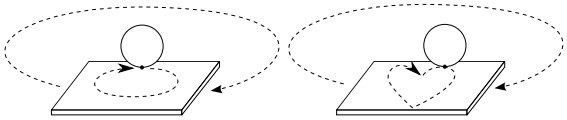

Figure 4.5: Separate superimposed trajectories for both ball and plate

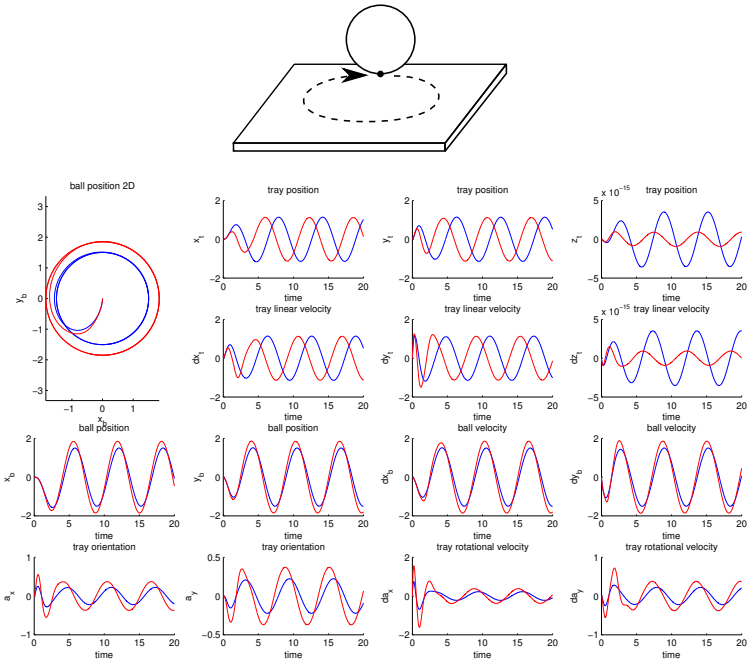

Figure 4.6: Circular reference trajectory for ball. Blue = linearized model, red (dashed) = nonlinear model

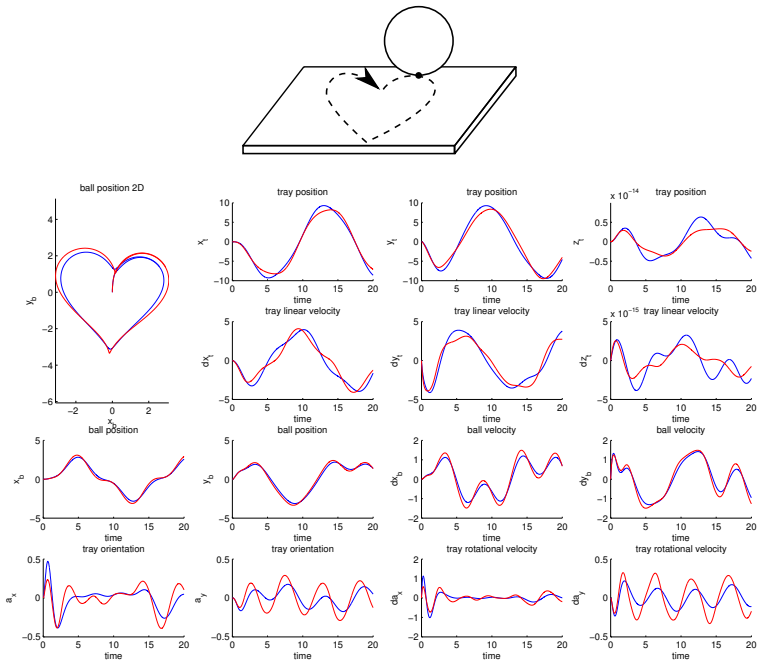

Figure 4.7: Heart shape reference trajectory for ball, circular trajectory for plate. Blue = linearized model, red (dashed) = nonlinear model

4.2 Time delay compensation

We improve our simulation by adding the CV time delay and the compensation mechanism as described in 5.3.2. For this, we implement the procedure described in 2.6.1 in MATLAB. In figure 4.8, the resulting simulations are compared with the previous nonlinear model.

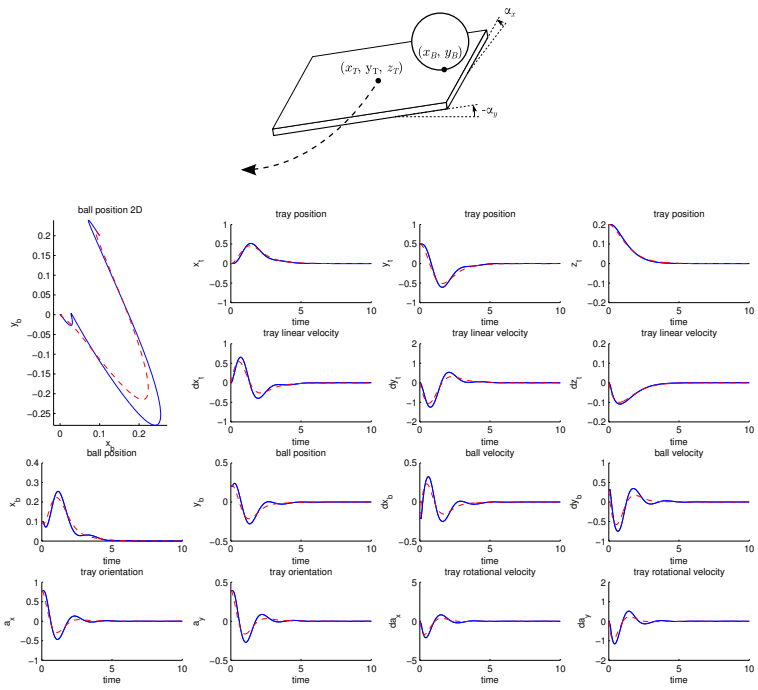

Figure 4.8: Rolling ball on displaced plate. Blue = model with time delay compensation, red (dashed) = simplified model without time delay

While the resulting ball trajectories and plate movements are stable and qualitatively similar to the existing simulations, we observe two differences. The improved model shows more overshoot (10-20 %) and slightly longer rise times. The settle time remains largely unchanged.

4.3 Stabilize ball on rounded plate

We simulate a ball on a rounded plate with $r_P = 0.4\ m$ and compare the resulting ball and plate movements with the flat plate. The results are shown in figure 4.9.

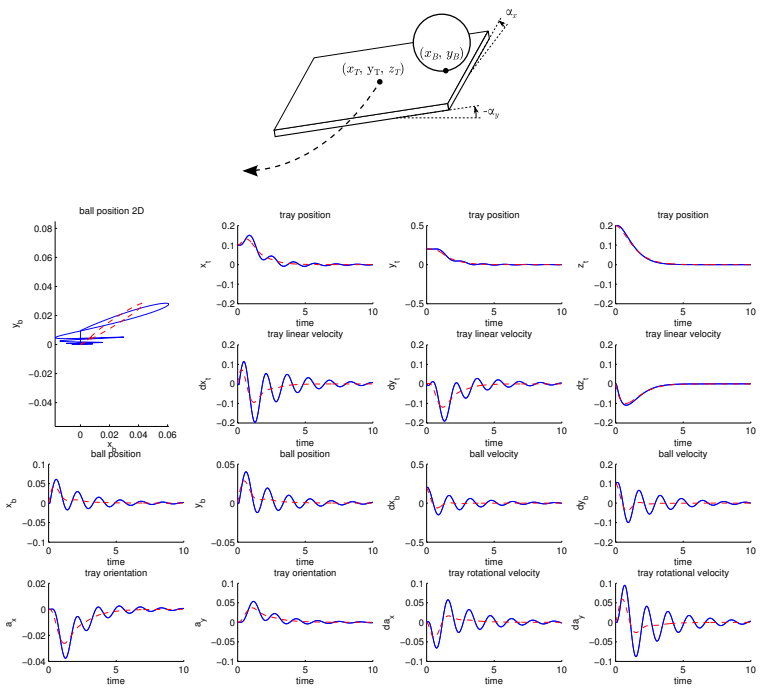

Figure 4.9: Rolling ball on displaced rounded plate. Blue = model with time delay compensation, red (dashed) = ball on flat plate

We observe larger oscillations in almost all states and measurements compared to the flat plate. The only exception is the robot z-axis x_z, which is not affected by the plate change. The settle time rises from approx. 5 s to 10 s.
This simulation suggests that our controller can be used to stabilize the ball on a rounded plate. This is yet to be proven in an experimental setup.

Chapter 5

Experiments and results

We verify our simulation results using the KUKA LWR robot and the physical system described in section 2.1. A video showing the experiment is available online at `http://youtu.be/4m6qwCTJRW8` and in the "ETH Zurich, IDSC" YouTube channel. We also conduct a series of measurements in order to better understand and describe our system.

5.1 Experimental setup

The physical properties of our system as introduced in section 2.2.3 are: $r_P \to \infty$, $m_P = 1.65$, $I_P = I_{Px} = I_{Py} = 0.0064$. We use two different balls, an NBA-endorsed basketball ($r_B = 0.122$, $m_B = 0.94$, $I_B = I_{Bx} = I_{By} = 0.0093$) and a gymnastic ball ($r_B = 0.085$, $m_B = 0.417$, $I_B = I_{Bx} = I_{By} = 0.0012$).
Experiments are initialized by manually placing the ball on the center of the plate. The balancing controller is switched on if x_B and y_B are sufficiently small for 0.5 seconds.

5.2 Stabilizing the ball on the plate

5.2.1 Plate at fixed position

Figure 5.1 shows the ball position errors (x_B, y_B). The ball is placed on the plate at t = 0 s, and the control is switched from safety stop to balancing at approximately t = 0.50 s. The ball position errors display relatively large deviations of about 4-5 cm but can be stabilized. Note that this measurement uses a controller based on a robot model that is not fully tuned to the real setup.

5.2.2 Set point tracking

Though originally designed for a constant position, this controller has been successfully tested for set point tracking at moderate speeds. Figure 5.2 shows the ball position errors (x_B, y_B).
The set point is moved from $y = 0.0\ m$ to $y = 0.2\ m$ at approximately $t = 1.50\ s$. The ball position errors are relatively large in the beginning, but quickly converge to values close to zero. The ball settles at a stationary offset on the

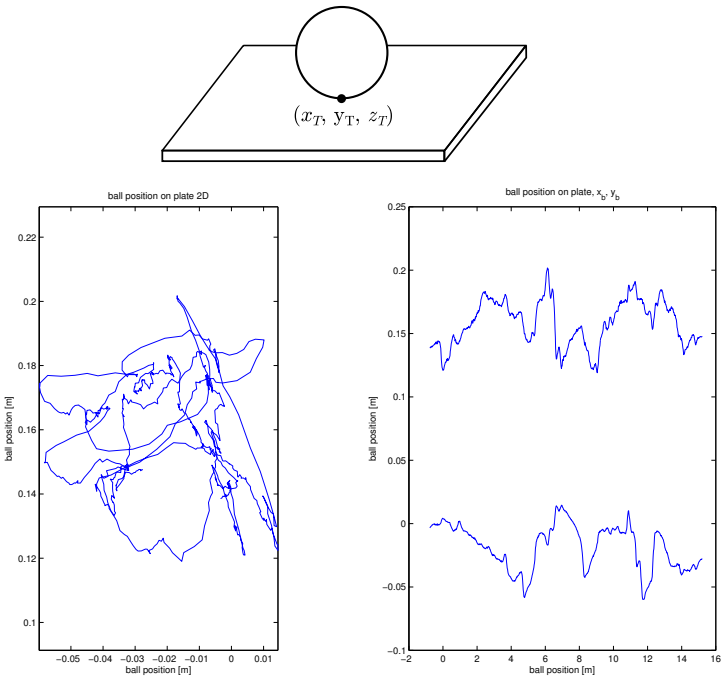

Figure 5.1: Ball balancing on plate in standstill

order of 5 cm from the desired position. Note that the balancing controller does not provide feedback on integrated errors.
The main suspected reason for these steady-state errors are unmodelled effects and ball imperfections. Deviations in the center of gravity of the ball can lead to a constant torque on the ball.

5.2.3 Basketball vs. gymnastic ball

We manage to balance various balls, and are able to switch between them rapidly during demonstrations. We measure the size and weight of each ball and use the LQR to calculate the new gain matrix K and verify the stability in simulation using the non-linear model. We then have to train the CV to detect the ball and store the relevant config files.
A qualitative comparison between the different balls shows that the basketball is harder to control than our gymnastic ball and our controller is able to stabilize it within 6-8 cm deviation in x_B and y_B. This is mostly due to the unmodelled effects of the about 1 mm deep rims that cause the ball to roll irregularly.

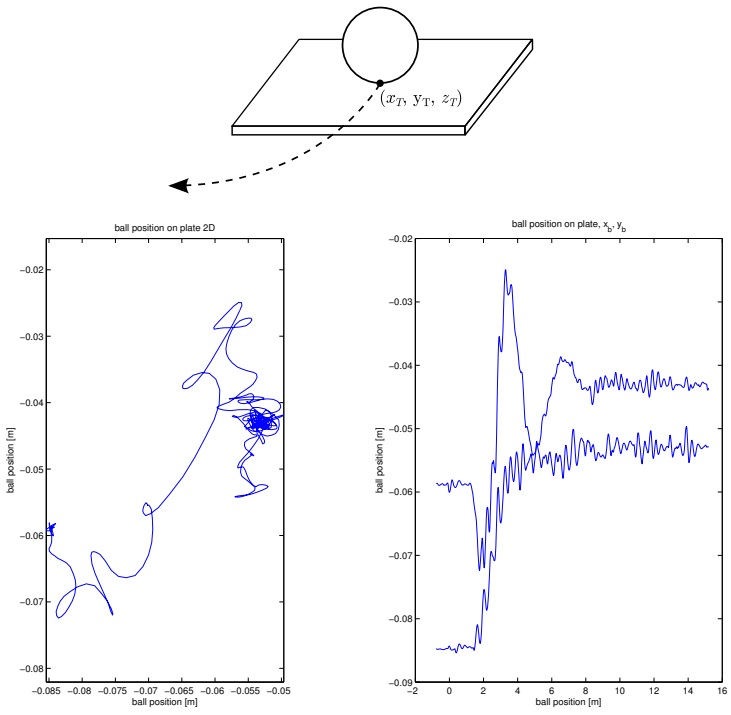

Figure 5.2: Ball balancing on plate with set point tracking

5.3 Sensor measurements and investigations

5.3.1 Stereo cameras specifications

We use a stereo camera setup developed in [5] to measure the ball position. The sensor information is found to have "discrete character".
We find that the stereo camera measurement contains noise in the order of 1-2 cm. Also, there are occational outliers that differ from the estimated position by over 10 cm, as seen in figure 5.3. We detect and filter measurements that exceed a certain threshold d_{max} between two consecutive time steps,

$$\Delta x_B = x_{B,k} - x_{B,k-1} > d_{max}$$

In our configuration, d_{max} is set to 4 cm, which - at 50 Hz - corresponds to a maximum allowed ball velocity of 7 m/s.

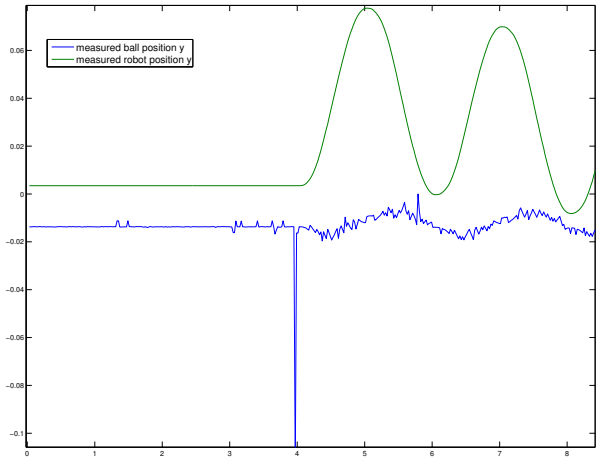

Figure 5.3: Fixed ball on plate. Green: ball position measurement by the stereo cameras, blue: ball position measurement by robot

5.3.2 Computer vision delay

In order to measure the delay caused by the image processing in our CV component, we firmly attach a fake ball (styrofoam block featuring colored circles) to the plate. We move the robot in a sinusoidal way and measure its current position using both the FRI plate position measurement y_P and the ball position measurement y_B.
The results are shown in figure 5.4. The resulting curves are very similar in shape and amplitude, but have a time delay. This time delay can be determined by calculating the crosscorelation of the two curves. The resulting values lie between 53 and 64 ms, so we find the CV time delay at 50 Hz to be 3 time steps (i.e. 60 ms).

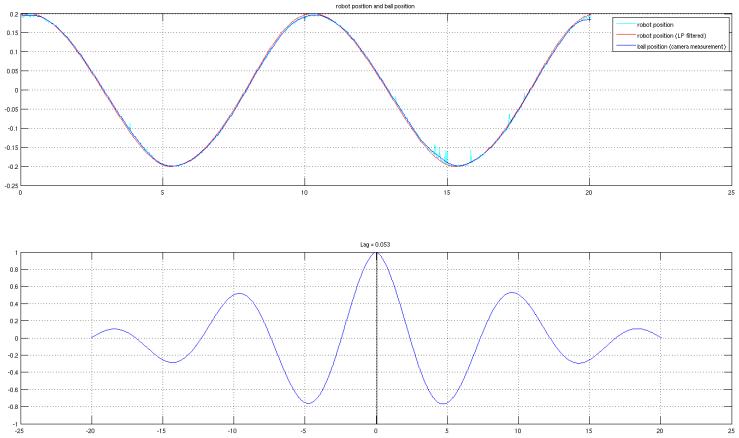

Figure 5.4: Top: Robot and ball position measurement. Note the noise and occational spikes in unfiltered ball position measurements (cyan line). Bottom: Crosscorrelation between ball and robot position measurement curves.

5.3.3 Robot pose measurements

We receive the cartesian robot pose measurements via the FRI Interface [8] using the port `msrCartPos`. Specifically, this port contains a message describing the cartesian position as a 3D vector, and the orientation in the form of a quaternion. A sample position measurement can be found in figure 5.3. We find that the position and orientation measurements from the robot have only little noise. For x_P, y_P, z_P, it is typically < 0.1 mm.

5.3.4 Robot cartesian torque measurements

Because of the noisy nature of the ball position measurements by the cameras, we investigate how we can improve our estimate of the ball position by using the torque measurements of the robot. The torque depends on the gravitational force on the ball and its position and the plate acceleration:

$$M_x = m_B y_B \big(gcos(\alpha_x) + \ddot{y}_P sin(\alpha_x)\big)$$
$$M_y = m_B x_B \big(gcos(\alpha_y) - \ddot{x}_P sin(\alpha_y)\big)$$

In the first experiment, we roll the ball over the plate. In the second experiment, we use our ball controller to balance the ball on the plate. The ball controller sends cartesian twist values to the FRI (which will be converted to cartesian position values internally).
From our measurements in figure 5.5 we find that if the plate is in the equilibrium position and the robot is not actuated, the ball position measurement from the estimated external wrench interface qualitatively matches the one from the stereo camera system. We observe a static offset, for which we could compensate using the measurement without ball.
From our second measurements in figure 5.6 we find that if the plate is actuated, the measurements do not match the camera values. We cannot find a

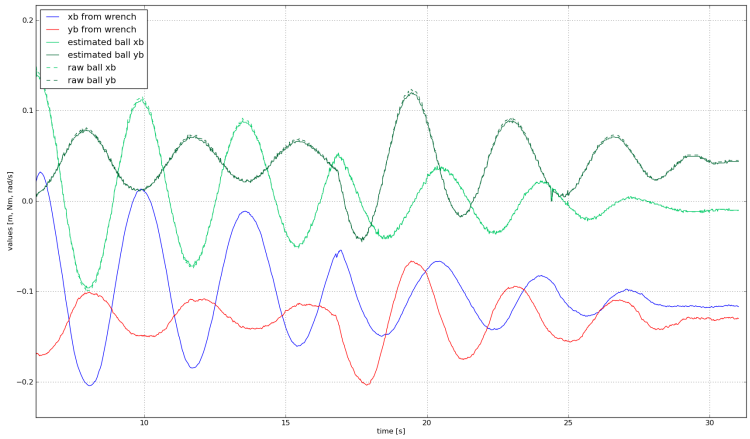

Figure 5.5: Torque measurements to determine ball position with robot in equilibrium

relation between the two ball position measurements, also when considering the commanded rotational speed ($u_{\alpha x}$, $u_{\alpha y}$) of the plate.

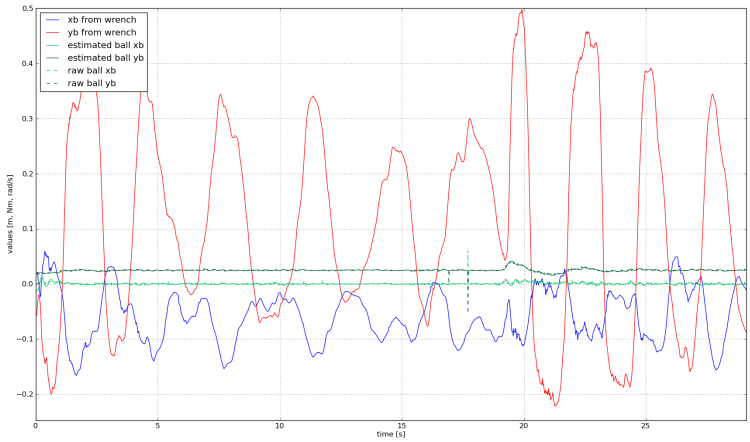

Figure 5.6: Torque measurements to determine ball position with actuated robot

We assume that the robot does not correctly compensate it's own joint actions for the estimated wrench given by `extTcpWrench` port when controlling the ball. Therefore, we conclude that - while the wrench measurements are much smoother - we cannot use this measurement for our purposes.

5.3.5 Investigating static offsets

Using a more fine-tuned controller, we are able to reduce the ball position errors as shown in figure 5.7. We see large static offsets in ball position x_B, y_B of 6 cm and 8 cm, respectively. Also, we note a static offset in plate rotation α_x, α_y of +/- 0.02 rad (approx. 1°). This means, the controller applies a (near constant) control input on the ball.
We conclude that an unmodelled effect on the ball dynamics (e.g. non-spherical shape) causes the ball to cancel the effect of this static control input and to find itself in an equilibrium position. As our controller has no integrator action, this static offset remains.

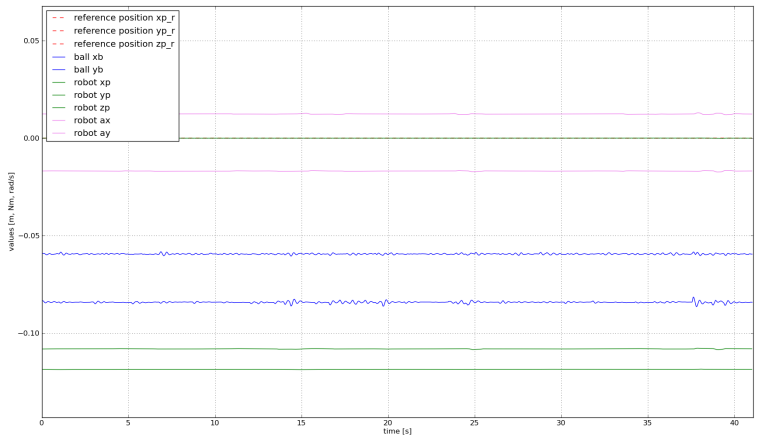

Figure 5.7: Ball balancing on plate in standstill with offset in ball position and plate rotation

5.4 Comparison between experiments and simulation

We compare our experimental results with our simulation using a set point tracking example. At $t = 2$ s, the setpoint of the robot reference changes from $y_P = 0$ to $y_P = 0.2$. All other references remain unchanged.
We observe in figure 5.8 that the simulation and experiment behave similarly, namely they both show an overshot and oscillations in robot and ball position, and a settling time of about 10 seconds. Notable differences include a static offset after stabilization of robot and ball position in the experiment. The simulation shows a faster response and less noise than the experimental values. The experiment, however, displays less overshoot in robot position/orientation and ball position.

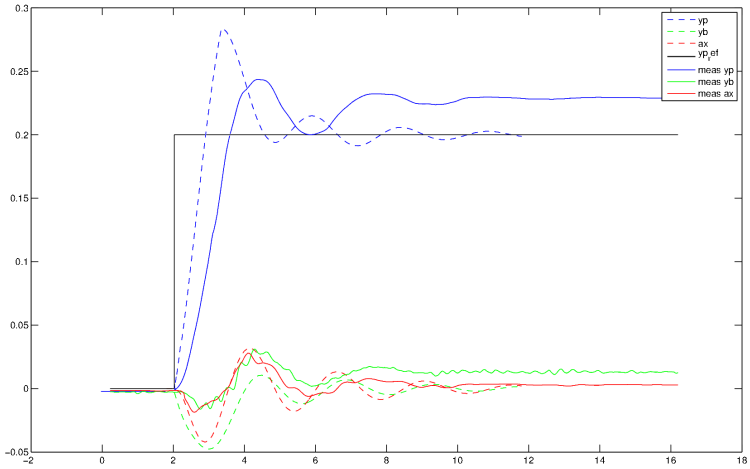

Figure 5.8: Set point tracking. Dashed lines: simulation results, solid lines: experimental values. Measurements have been shifted to match 0 at $t < 2\ s$.

Chapter 6

Conclusion and outlook

We have developed a linear controller for stabilizing a ball on a plate. The controller is found to be robust enough to stabilize the ball in the presence of measurement noise acting on the computer vision component. Controllers for standstill and circular motion have been validated experimentally and are shown to stabilize the ball on the plate. This key milestone allows us to shift our focus towards improving system performance and adding complexity (i.e. more balls). Experimental results revealed systematic errors when applying the control laws. There appear to be different sources of these errors (based on [6]):

- There are unmodelled physical effects acting on the ball. This includes friction losses of the ball on the surface and irregularities of the ball shape.
- Miscalibrations of the external camera parameters cause biases in the experimental setup. These errors are observed in the ball measurements and vary throughout the robot work space.
- The simplifying assumption that the plate rotates around a point on the surface is violated in the experimental setup, where the robot rotates the plate around a point about 2 cm below the surface. Rotations of the plate therefore cause an unmodelled effect on the ball.

We have identified two approaches for extending the controller design presented in this paper.
The first approach is to include states that represent the integrated errors, and to weigh them appropriately in the controller design. This would permit compensation for some of the systematic errors. For instance, one would expect this to drive the vehicle position errors in the standstill case to zero. Alternatively, the desired trajectory can delivered as a feed-forward term to the controller. This will help to anticipate motions and improve controller performance in case of set point tracking.
The next step towards the goal of balancing three balls is to verify the obtained controller for the ball-on-rounded-plate setup on the KUKA LWR, and then adding more balls.

Appendix A

Source code and documentation

The source code of this project, including simulation and report, may be found in the github repository at `https://github.com/IDSCETHZurich/Ball-balancing`, as of 2013-04-22.

A.1 Software architecture

A.1.1 Libraries

This project uses the following libraries:

- ROS[4]
- OROCOS[3]
- boost.Math [7]
- libBallFind: The CV library detecting the ball [5]
- several Python packages

A.1.2 OROCOS and ROS Components

If nothing else is stated, these are OROCOS components.
The following components are used during production:

- `prosilica_camera` (ROS): Receive the camera image
- `camera_sync`: Gather camera images synchronously from two `prosilica_camera` ROS nodes
- `ball_find`: Detect the ball in the images, makes use of libBallFind
- `position_estimator`: Estimate the ball position by fusing the two images
- `ball_controller`: Control the ball by commanding the robot

The following components are used during calibration:

- `camera_calibration`: Detect intrinsic camera parameters (e.g. distortion)
- `system_calibration`: Detect extrinsic camera parameters (position and orientation relative to robot)
- `ball_train`: Learn ball colors by user input

The other parts of the code are:

- `utils` (ROS): Helper nodes for various tasks
- `config`: config files for nodes, plus dynamic configs for calibration
- `docs`: documentation, incl. diagrams
- `report`: report for semester / master thesis
- `simulation`: MATLAB simulations
- `various`: launch files for quick start

A.2 Component interaction

Figure A.1 shows how the ball balancing components interact during calibration. There are three calibration steps:

1. Camera calibration (internal calibration of each camera)
2. System calibration (external calibration, i.e. position of cameras relative to robot)
3. Ball train (learn ball color)

Figure A.2 shows how the ball balancing components interact during production.

A.3 Tutorials and operations

To get a quick start into the project and to find useful hints, take a look at the `README` file in the github repository at `https://github.com/IDSCETHZurich/Ball-balancing`.

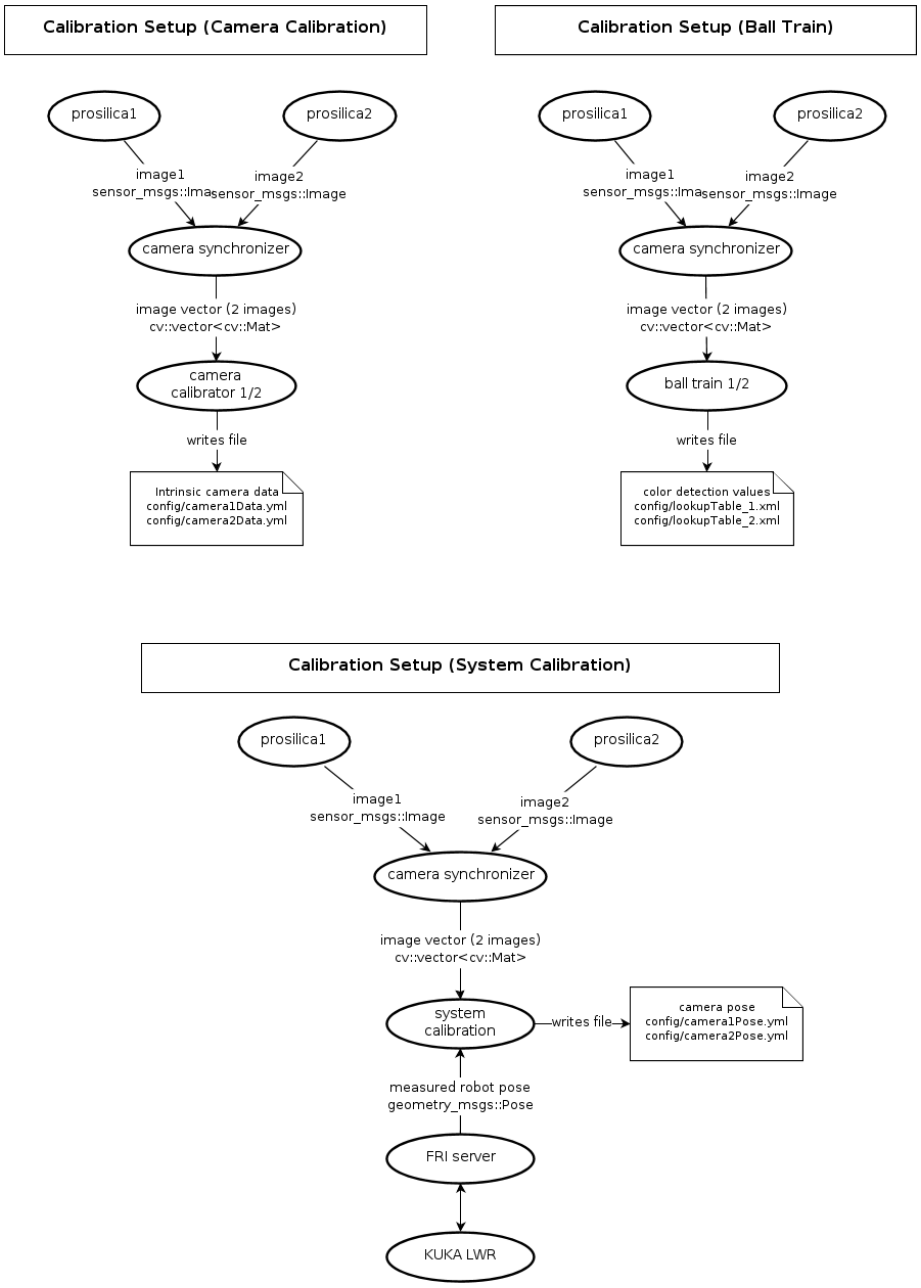

Figure A.1: Ball-balancing components-calibration

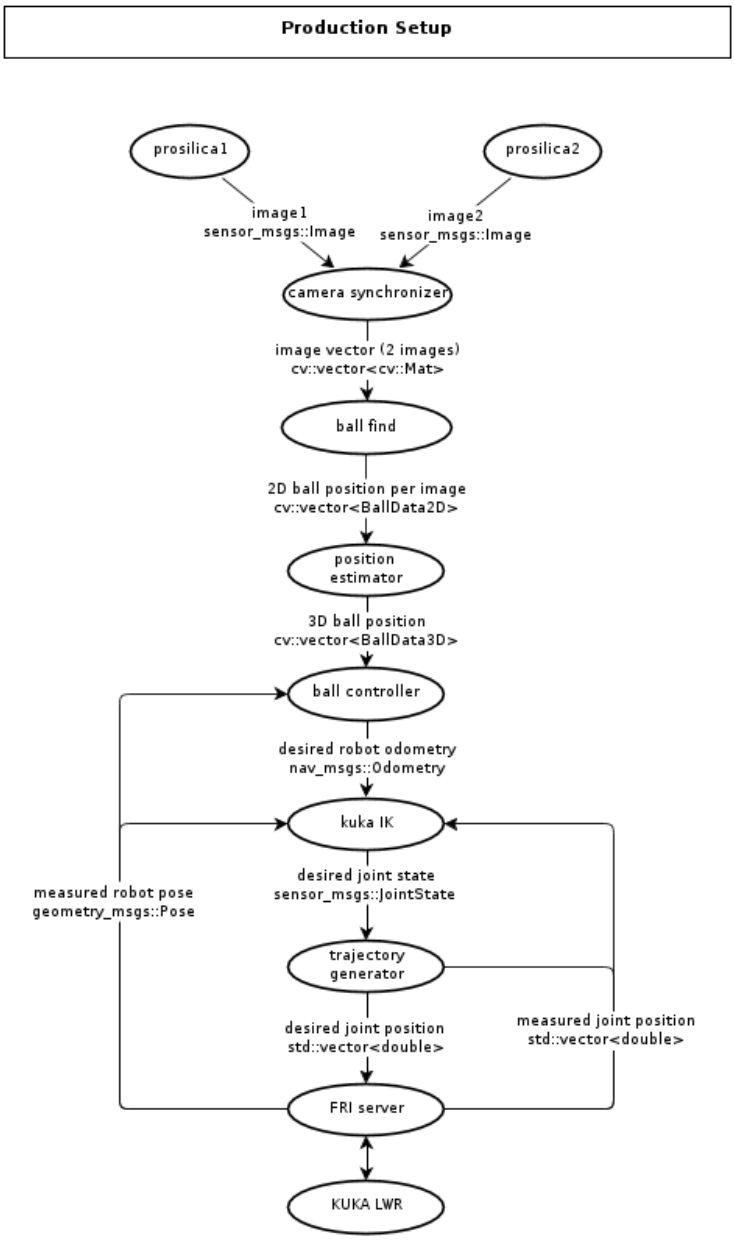

Figure A.2: Ball-balancing components-production

Appendix B

Mathematica calculations

All Mathematica calculations and notebooks can be found in the repository, in the folder `calculations/`.

Appendix C

Additional simulation results

In this chapter, we show a few more simulation results. Videos of 3D visualizations may be found at `http://n.ethz.ch/~wettsten/download/ball-balancing/`. Note: In the 3D visualizations, the plate may be seen "going through" the ball. This happens when the ball is clearly off center, and is due to the controller rotating the plate around its center of mass, regardless of the current ball position.

C.1 Stabilizing the ball on the plate

If not mentioned otherwise, we assume zero states $q = \dot{q} = 0$ and no input $u = 0$.

Equilibrium We consider the equilibrium case. The results in figure C.1 show that the ball is stabilized in the origin, with no difference between the linearized and the nonlinear model.

Displaced ball The ball is off center, $x_B = y_B = 1$. The results in figure C.2 show that the ball position is stabilized quickly, with settle time similar in both models. There is some overshoot in the nonlinear model not present in the linearized case. Also, plate position and velocity shows oscillation that is larger in the nonlinear case by a factor of 2.

Displaced plate The plate is off center, $x_T = y_T = 5, z_T = 1$. The results in figure C.3 show that the ball position is stabilized quickly, with a slightly longer settle time in the nonlinear model.

Rolling ball The ball is rolling, $\dot{x}_B = \dot{y}_B = 1$. The results in figure C.4 show that the ball position is stabilized quickly, with more oscillation, some overshoot and a slightly longer settle time in the nonlinear model.

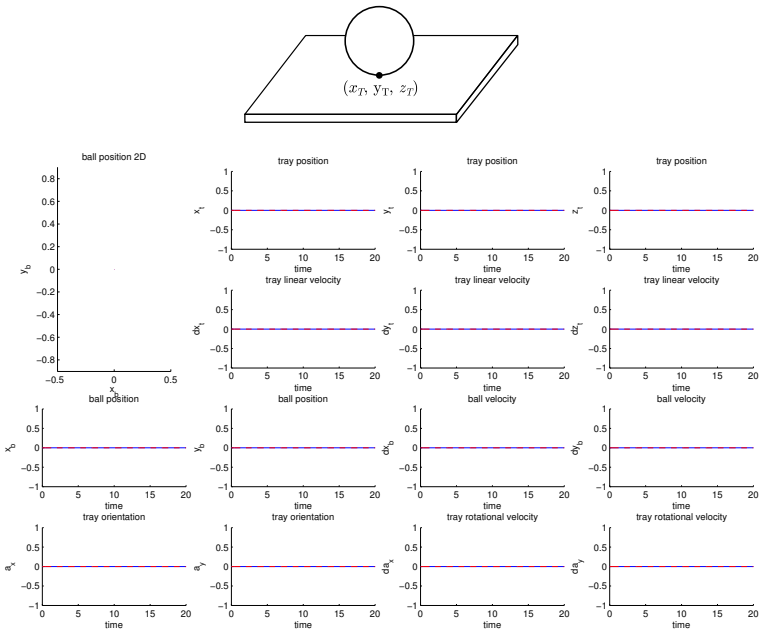

Figure C.1: Equilibrium. Blue = linearized model, red (dashed) = nonlinear model

Tilted plate The plate is tilted, $\alpha_x = \pi/8, \alpha_y = \pi/4$. The results in figure C.5 show that the ball position is stabilized quickly, with more oscillation, some overshoot and a slightly longer settle time in the nonlinear model.

Displaced and rolling ball on tilted and displaced plate Combining all of the above, the results in figure C.6 show that the ball position is still stabilized quickly, with more oscillation, some overshoot and a slightly longer settle time in the nonlinear model.

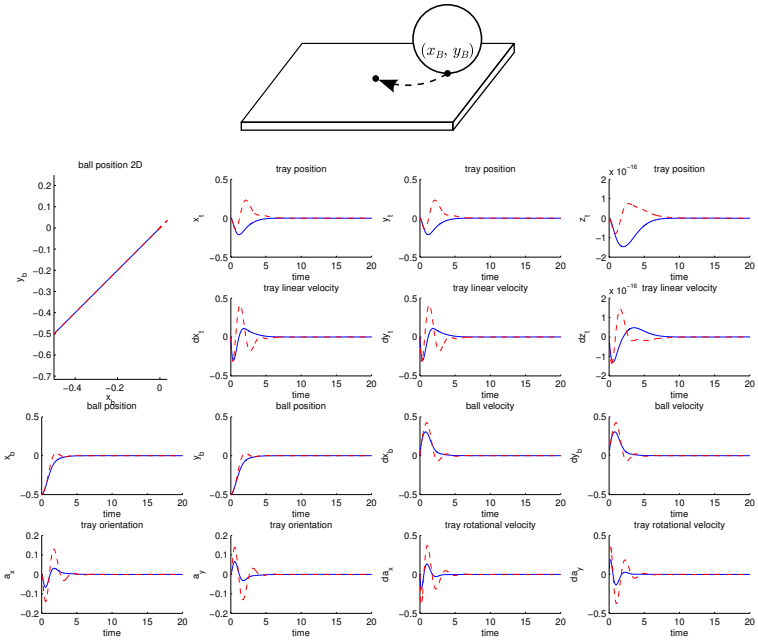

Figure C.2: Displaced ball. Blue = linearized model, red (dashed) = nonlinear model

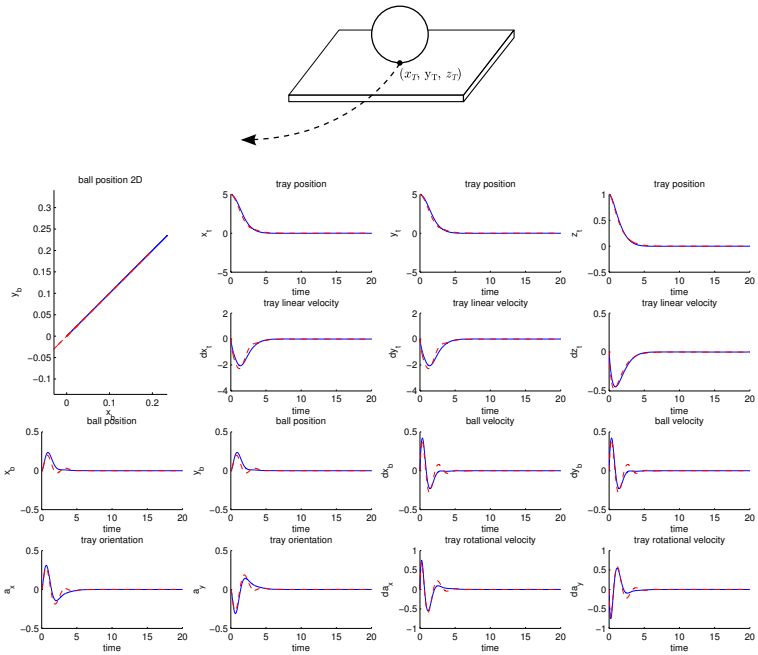

Figure C.3: Displaced plate. Blue = linearized model, red (dashed) = nonlinear model

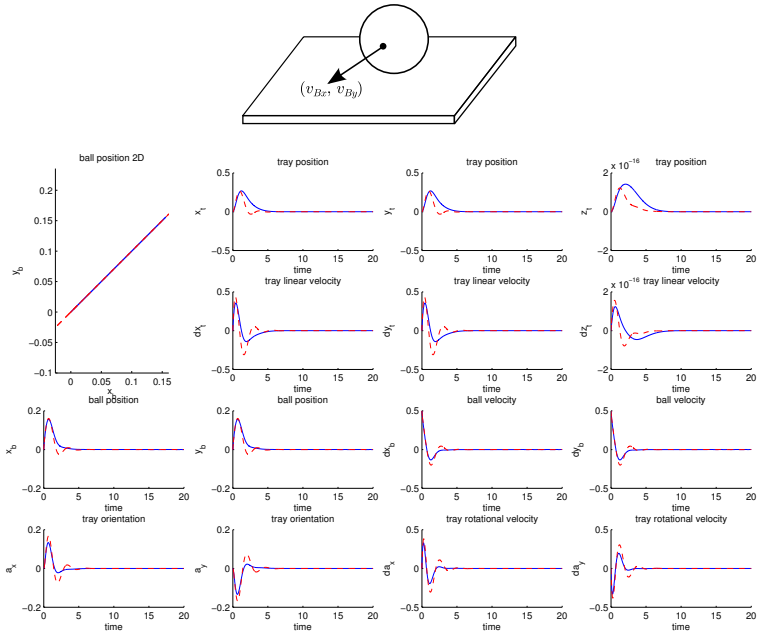

Figure C.4: Rolling ball. Blue = linearized model, red (dashed) = nonlinear model

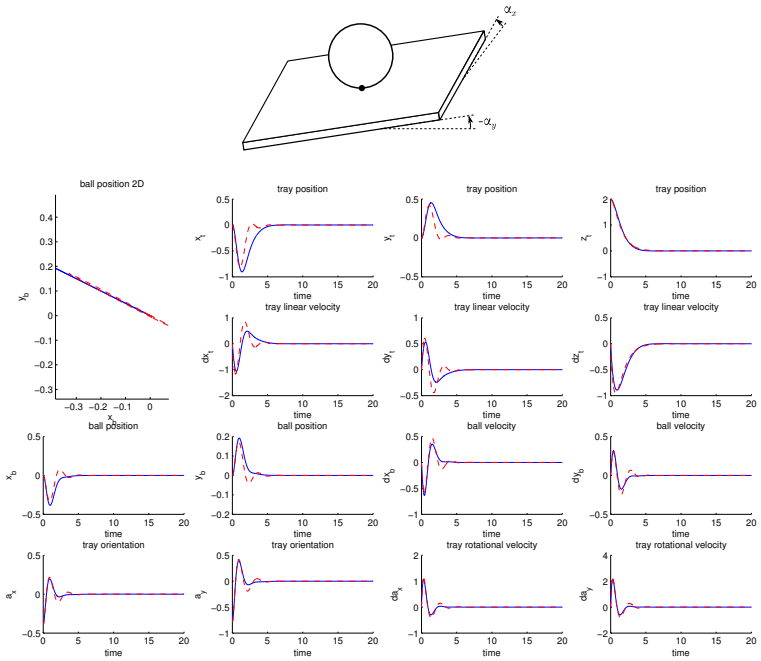

Figure C.5: Tilted plate. Blue = linearized model, red (dashed) = nonlinear model

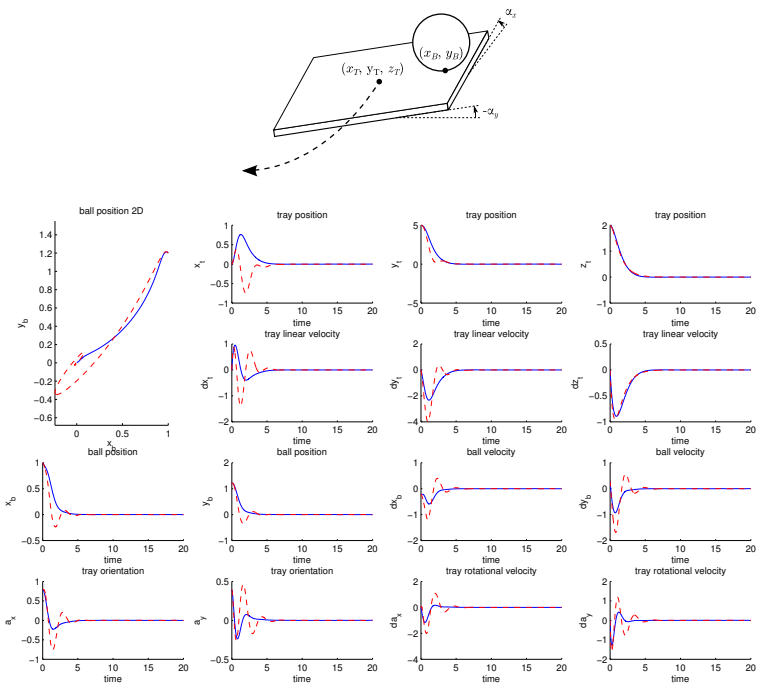

Figure C.6: Displaced and rolling ball on tilted, moving and displaced plate. Blue = linearized model, red (dashed) = nonlinear model

C.2 Ball and plate follow reference trajectory

We will start from the equilibrium position for all reference trajectory simulations.

Circular reference trajectory for ball The ball needs to follow a circular reference trajectory.

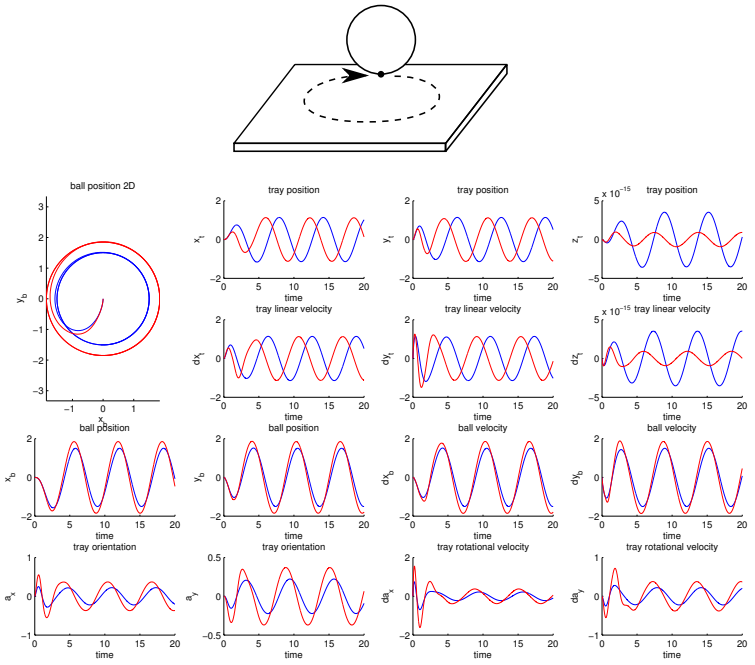

Figure C.7: Circular reference trajectory for ball. Blue = linearized model, red (dashed) = nonlinear model

Circular reference trajectory for both ball and plate The ball needs to follow a circular reference trajectory on the plate, and the plate follows a circular trajectory in world coordinates.

Heart shape reference trajectory for ball, slow The ball needs to follow a heart shape reference trajectory, and the plate follows a circular trajectory in world coordinates.

Heart shape reference trajectory for ball, fast The same setup as above, but with a faster speed. We observe that the system becomes unstable as the ball may not follow the reference.

Rectangular reference trajectory for ball The ball needs to follow a rectangular reference trajectory. We see that with this shape, the linearized model behaves quite well, but the nonlinear model turns unstable.

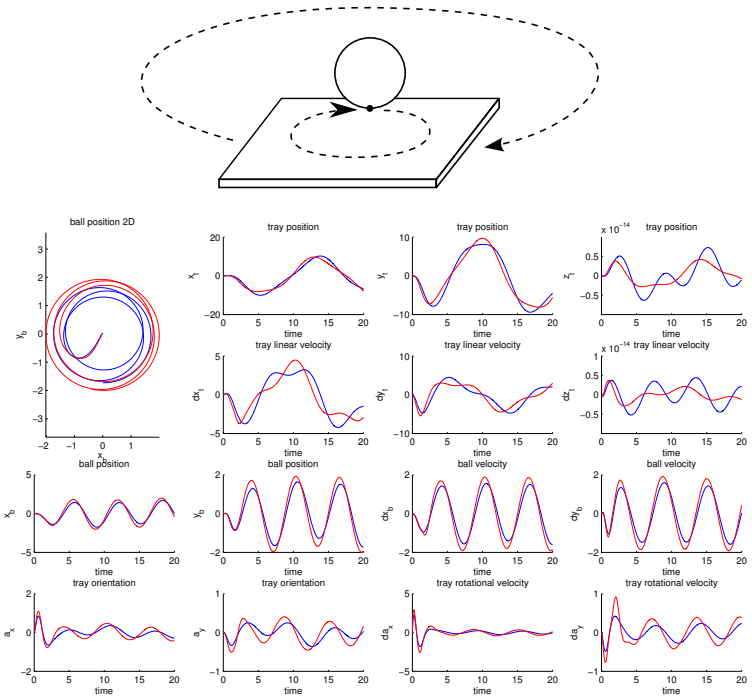

Figure C.8: Circular reference trajectory for both ball and plate. Blue = linearized model, red (dashed) = nonlinear model

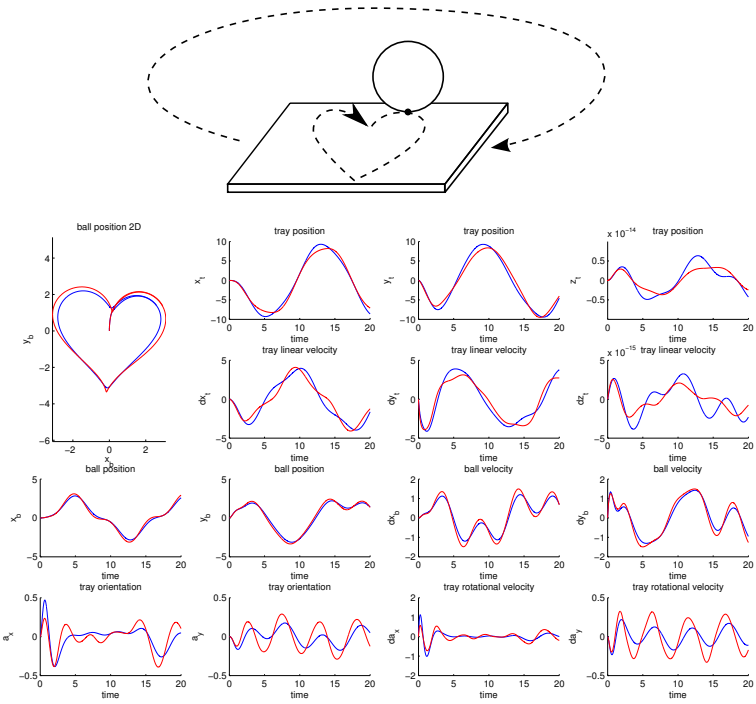

Figure C.9: Heart shape reference trajectory for ball, circular trajectory for plate. Blue = linearized model, red (dashed) = nonlinear model

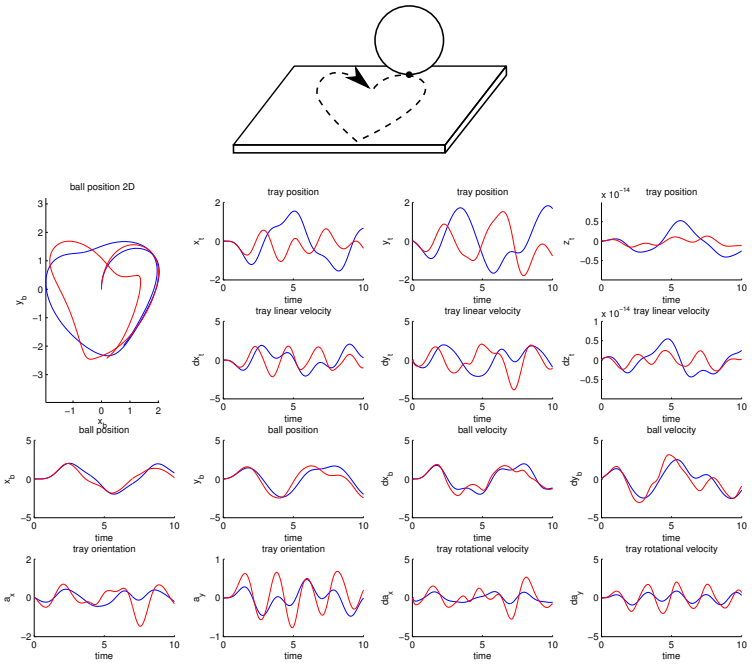

Figure C.10: Heart shape reference trajectory for ball. Blue = linearized model, red (dashed) = nonlinear model

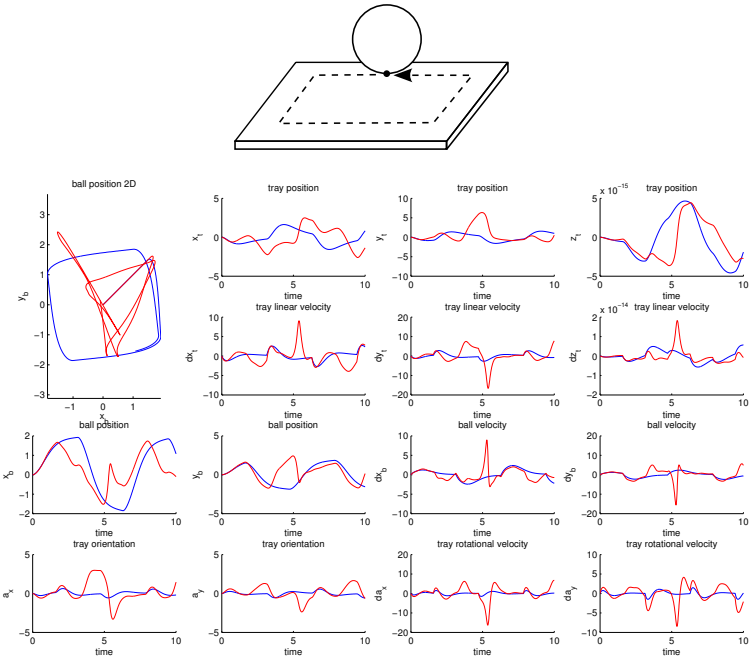

Figure C.11: Rectangular reference trajectory for ball. Blue = linearized model, red (dashed) = nonlinear model

Bibliography

[1] Siciliano B., Sciavicco L., Villani L., and Oriolo G. *Robotics - Modelling, Planning and Control*. Springer, 2011.

[2] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, et al. The kuka-dlr lightweight robot arm-a new reference platform for robotics research and manufacturing. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1–8. VDE, 2010.

[3] OROCOS community. Open robot control software. `http://www.orocos.org/`, 2012. [Online; accessed 2013-04-22].

[4] ROS.org community. Robot operating system. `http://www.ros.org/`, 2012. [Online; accessed 2013-04-22].

[5] D. Eberli and N. Huebel. Modeling and control of a ball balancing system using stereo vision. Master's thesis, ETH Zurich, 2012.

[6] M. Hehn and R. D'Andrea. A Flying Inverted Pendulum. In *IEEE International Conference on Robotics and Automation*, 2011.

[7] Rene Rivera and Boost Community. Boost c++ library. `http://www.boost.org/`, 2013. [Online; accessed 2013-04-22].

[8] G. Schreiber, A. Stemmer, and R. Bischoff. The fast research interface for the kuka lightweight robot. In *IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications How to Modify and Enhance Commercial Controllers (ICRA 2010)*, 2010.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control
Prof. Dr. R. D'Andrea, Prof. Dr. L. Guzzella

Title of work:

Balancing a Ball on a Plate using Stereo Vision

Thesis type and date:

Master Thesis, April 2013

Supervision:

Nico Hübel
Prof. Dr. Raffaello D'Andrea

Student:

Name:	Nathanael Wettstein
E-mail:	nate@student.ethz.ch
Legi-Nr.:	07-906-787
Semester:	10

Statement regarding plagiarism:

By signing this statement, I affirm that I have read the information notice on plagiarism, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Information notice on plagiarism:

`http://www.ethz.ch/students/semester/plagiarism_s_en.pdf`

Zurich, 24. 4. 2013: _____