Lab 7

Closed Loop Control

Author:

Jonas Lussi, Naveen Shamsudhin, Alexander Mesot, Viviane Gerber, Dominik Schulte and Prof. Bradley J. Nelson

Institute of Robotics and Intelligent Systems

Date: 2020

Version: 1.0

Summary: The purpose of this week's lab is to model and simulate a PID controller for our ball balancing system.

7.1 Background

This Prelab will focus on designing a PID controller for our ball balancing system. However, since we are not able to be in the lab, we will simulate a simplified version of the system in Matlab and design the PID controller for that system. The ball balancing system consists of 3 servo motors (similar to the motors that you have seen in Lab 04), joints that are connecting the motors to a plate, as well as a camera for visual feedback. The general setup is depicted in 7.1 The logical flow for the control of the system can be seen in Fig. 7.2

Note: The design of this system was inspired by Johann Link: https://www.instructables.com/id/Ball-Balancing-PID-System

7.2 Camera

The camera we use is a Pixy cam. It has integrated object detection and can directly send the position of the ball to the Udoo. The camera is designed for visual servoing and has a minimal latency of around 20 ms (=signal delay) and a high fps (frames per second = signal frequency). If we would use a standard camera, the images would need to be compressed for sending, decompressed on the Udoo and then analyzed with a computer vision process. This would lead to a large delay in the system which would make it impossible to control the ball with a PID approach. You will investigate how the delay impacts the performance of the system further in this Prelab.

Further information about the camera can be found here: https://pixycam.com

7.3 Kinematic Chain

To drive the plate that balances the ball, we use three servo motors that are connected to the plate via three separate two-link arms. This allows us to control the two rotational degrees of freedom of the plate, as well as its height.

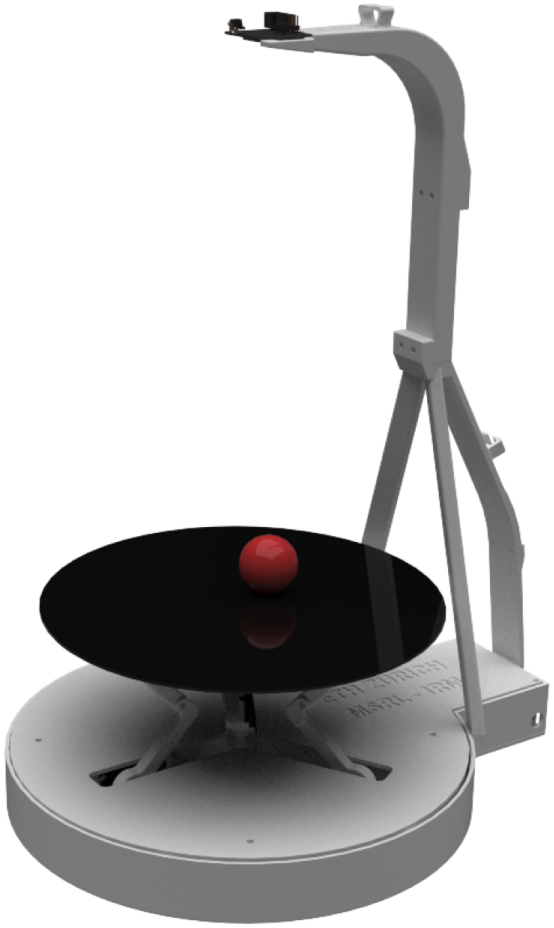

Figure 7.1: Ball balancing system you were gonna use in this lab if not for Covid-19.

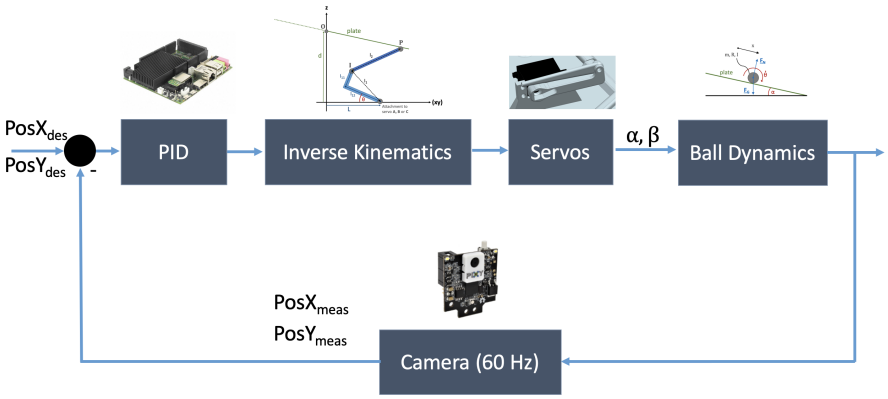

Figure 7.2: Control flow of the ball balancing system

However, for the purposes of this Prelab, we discard the relationship between motor angles and plate pose. To simplify our model of the system, we skip the motors and links and assume that the output of the controller is directly the plate angle.

7.4 Dynamic Model

7.4.1 Rolling motion

To model the plant (ball dynamics), we first have to derive the differential equations describing the motion of the ball in respect to the input to the plant, which is the plate angle. The ball is assumed to roll without slipping on an inclined plane. For simplicity, drag is omitted. The inclination angle α, the moment of inertia $I = \frac{2}{3}mR^2$ and the angular velocity $\dot{\theta}$ have to be taken into account. The movement of the ball can be analyzed in 2D, as $\ddot{x}$ only depends on the corresponding angle α. Similarly the acceleration in y-direction $\ddot{y}$ only depends on the corresponding angle β. Every other movement is a linear combination of the two.

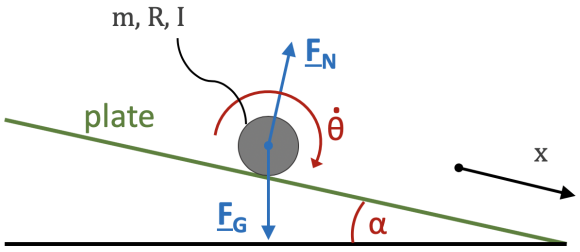

Figure 7.3: Simplified Model of the Ball Dynamics

$$
\begin{aligned}
\left(I + mR^2\right)\ddot{\theta} &= mgR\sin(\alpha) \\
I\frac{\ddot{\theta}}{R} + mR\ddot{\theta} &= mg\sin(\alpha) \\
\frac{2}{3}mR\ddot{\theta} + m\ddot{x} &= mg\sin(\alpha) \\
\frac{2}{3}m\ddot{x} + m\ddot{x} &= mg\sin(\alpha) \\
\frac{5}{3}m\ddot{x} &= mg\sin(\alpha)
\end{aligned}
$$

Finally, this gives the following accelerations:

$$\ddot{x} = \frac{3}{5}g\sin(\alpha), \quad \ddot{y} = \frac{3}{5}g\sin(\beta)$$

7.4.2 Camera Projection

Since the plate is tilted, the distances (and accelerations) have to be projected back onto the camera plane:

$$\ddot{x} = \frac{3}{5}g\sin(\alpha)\cos(\alpha), \quad \ddot{y} = \frac{3}{5}g\sin(\beta)\cos(\beta)$$

7.4.3 Linearization

The equations have to be linearized to calculate the transfer function. We use small perturbations of α around the equilibrium point α_0. We can then write: $\alpha = \alpha_0 + \partial\alpha$. Similarly the output is linearized with small perturbations as $x = x_0 + \partial\, x$. For $\alpha_0 = 0$, $x_0 = 0$ the linearized equations in x and y, are then derived with Taylor expansion

similarly as follows:

$$\begin{aligned} \partial\ddot{x} &\approx \tfrac{3}{5}g\cos^2(\alpha)|_{\alpha=\alpha_0}\partial\alpha - \tfrac{3}{5}g\sin^2(\alpha)|_{\alpha=\alpha_0}\partial\alpha \\ &\approx \tfrac{3}{5}g\partial\alpha \\ \partial\ddot{y} &\approx \tfrac{3}{5}g\partial\beta \end{aligned}$$

7.5 PID Controller

A proportional–integral–derivative controller (PID) is one of the most common controllers in the world. It is widely used in industry for its simple, model-free implementation and its versatility. A PID controller calculates the error $e(t)$ between a desired setpoint of a process variable (e.g. postion) and the measurement of said signal and applies a proportional, integral and differential operation to this time-dependent error. The output of the PID controller is then used as an input to the plant (e.g. actuator) to reach the desired setpoint of the process variable. In the frequency domain, the transfer function for an analog PID controller is:

$$T(s) = K_P + \tfrac{1}{s}K_I + sK_D$$

where K_P, K_I and K_D are the proportional, integral and derivative gains respectively.

Figure 7.4 shows the diagram for a PID controller with anti-windup. When the actuator saturates, the anti-windup scheme will be activated and prevent the controller output from wandering away. In our case this could happen when we give the motor commands that they cannot achieve due to their limited speeds. For this Prelab however, we will omit the anti-windup and focus on the PID controller itself. This is just for your personal reference.

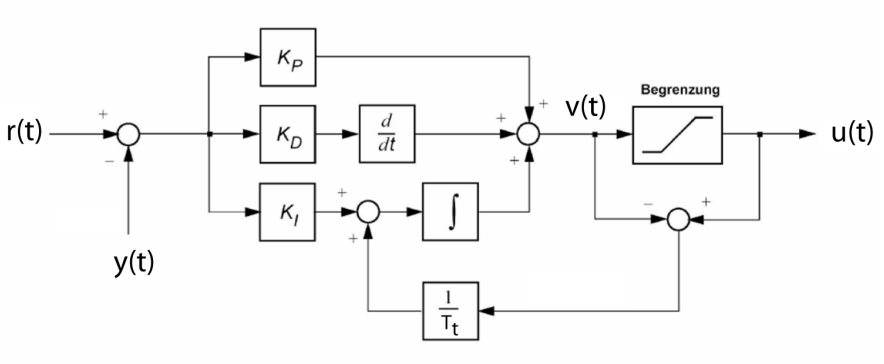

Figure 7.4: PID controller with windup protection.

7.5.1 Discrete Implementation

We will not use a discrete implementation in the simulation, however this section is for your personal reference when implementing a PID controller in a real system in the future (or when you will join the optional intensive Lab-Week). At any time t_n, the output of the controller is given by:

$$v(t_n) = P(t_n) + I(t_n) + D(t_n)$$

where

$$\begin{aligned} P(t_n) &= K_P \cdot e_n \\ I(t_n) &= I(t_{n-1}) + K_I \cdot \frac{e_n + e_{n-1}}{2} \cdot T_s \\ D(t_n) &= K_D \cdot \frac{e_n - e_{n-1}}{T_s} \end{aligned}$$

Note: there are other possibilities to implement the PID controller. For example, a common choice for $I(t_n)$ is also

$$I(t_n) = I(t_{n-1}) + K_I \cdot e_n \cdot T_s \quad \text{or} \quad I(t_n) = I(t_{n-1}) + K_I \cdot e_{n-1} \cdot T_s.$$

The algorithm for PID control thus becomes:

Initialization

1. Set the values of K_P, K_I and K_D.
2. Set the initial values of $I(t_{n-1})$, the sample time T_s and e_{n-1}.

Controller algorithm

1. Input the error e_n.
2. Calculate $P(t_n) + I(t_n) + D(t_n)$ using the above equations.
3. Calculate the temporary output $v(t_n)$ using the above equation.
4. Check for saturation and calculate the controller output $u(t_n) = sat(v(t_n))$.
5. Output $u(t_n)$ (used as input to your motor)

Update controller variables

1. Update the value of the error by setting e_{n-1} equal to e_n.
2. Update the value of the position.
3. Update the value of the integral by setting $I(t_{n-1})$ equal to $I(t_n)$
4. Wait for the sampling interval to elapse.
5. Repeat the loop.

7.5.2 Step Response

Depending on the damping in a system and the controller gain that is used, the step response can be oscillatory (under-damped), critically damped or over-damped.

Figure 7.5 shows the step response for an arbitrary system (not the ball balancing system) for different gain values of $K_p > K_{cr}$ (underdamped), $K_p = K_{cr}$ (critically damped) and $K_p < K_{cr}$ (overdamped).

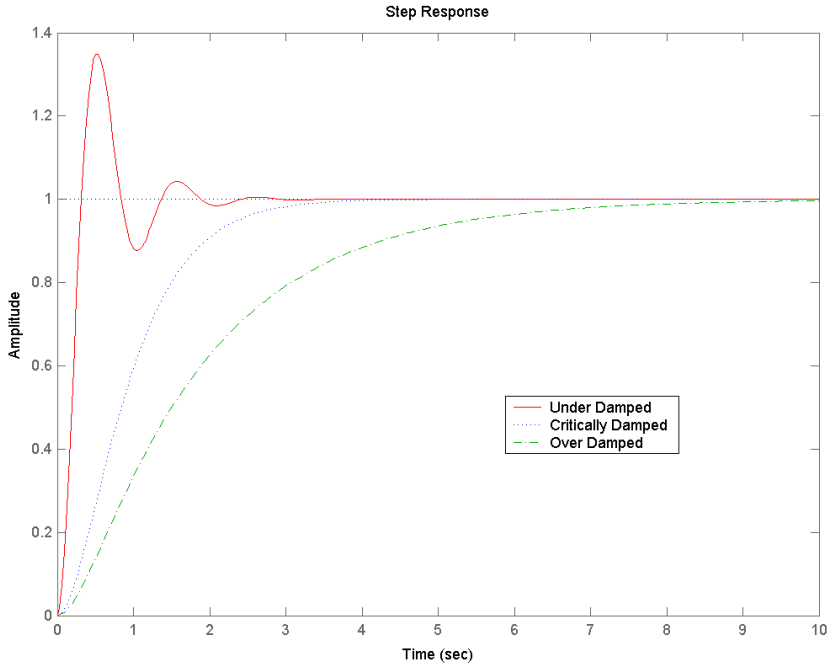

Figure 7.5: response for under damped, critically damped and over damped cases.

7.6 Prelab Procedure

Note: Prelab assignments must be done before reporting to the lab and must be turned in to the lab instructor at the beginning of your lab session. Additionally, you also have to upload your solution as a single PDF-file to Moodle. Make sure to upload the file before your lab session starts, late submissions will not be corrected. The prelab needs to be handed in as a group. All the prelab tasks are marked with **PreLab Qx**.

We will investigate the performance of a PID controller on the given system. For that, we use frequency domain methods, to easily simulate the closed-loop control. *lab07.m* contains the skeleton of the code. You can find the file on Moodle. Download it and follow the steps listed below. Put your code in the respective sections and hand in all the code together with the answers to the questions and all the plots we ask for.

1. To analyze the system, it is sufficient to only consider one of the two directions of movement. We will look at the behaviour of the PID controller with movement in x-direction. Calculate the open loop transfer function of this plant. You can use the linearized differential equation to calculate the transfer function. The transfer function is given by the laplace transform of small changes in output over small changes in the input. The constant g can be approximated by 10 m/s^2. Plot the impulse and the step response of the open loop transfer function using the functions impulse() and step() in Matlab. Use a step of 0.1 magnitude, this means your input signal wants to control the ball 10 cm off-center. On your report, show the plots as subplots next to each other and comment and explain the difference between step and impulse response. **(PreLab Q1)**

2. Create the transfer function of the controller and close the feedback loop according to Fig. 7.2. Note that the inverse kinematics and servos are now skipped. You can further assume that the camera feedback has a transfer function of 1. For control in x-direction, this results in the simple system depicted in Fig. 7.6:

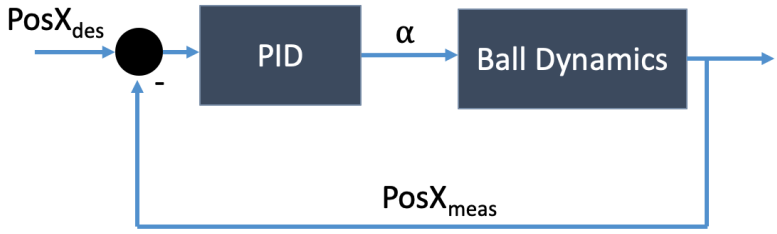

Figure 7.6: Simplified contol scheme to analyze PID feedback control.

Plot the Step (0.1 magnitude) response for K_p=20, K_i=1, K_d=2. Is this response under damped, critically damped or over damped? What happens to your response if you increase K_d? If you consider our real system with visual Feedback, what could be the risk of increasing K_d too high? On your report, hand in the plot for the two different K_d. **(PreLab Q2)**

3. The real system has a small latency due to the image-processing part happening on the Pixy cam. Although the setup with the Pixy cam minimizes this delay, it is still big enough to impact our controller design. Simulate this time-delay by adding it to our control loop. A time delay in the frequency domain is given by the exponential function e^{-sT}, where s is the laplace variable and T the delay. You will see that the system is no longer stable with this time delay. Find parameters that stabilize the system with a steady state error at 5 seconds that is smaller than 0.001 for a step input of 0.1 magnitude. Hand in the plot for your report. What happens if the delay increases to 300 ms? Are you able to stabilize the system within 5 seconds? **(PreLab Q3)**

4. We have simplified our system to quickly see if controlling it with a PID is possible. By simplifying it so much, we did not consider physical constraints on the motors and system at all. To extend our analysis of the control system, we now assume that the plate can not be tilted more than 25 degrees at any time. How could you use this constraint to see if the PID controller's output exceeds the capabilities of our system? (Hint: use the linearized differential equation and the data after the initial delay of 80 ms from the step function). Make the necessary calculations in Matlab and decide if your controller would satisfy this condition. **(PreLab Q4)**

5. When analyzing the system further you notice that the PID controller's output also exceeds the speed limits of your motors. What could be one of the problems you face? Give a short answer in your report. **(PreLab Q5)**