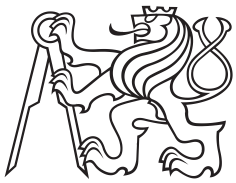
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Flying Ball and Beam System

Šimon Lehký

Supervisor: Ing. Jiří Zemánek, Ph.D.
Study program: Cybernetics and Robotics
May 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Lehký Šimon** Personal ID number: **498949**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Flying ball and beam system

Bachelor’s thesis title in Czech:

Systém létající kulička na tyči

Guidelines:

The aim of this bachelor thesis is to extend the existing education laboratory model "Ball in a hoop" with an additional mode of operation called "Flying ball on a beam".
1) Create a mathematical model describing the motion of a ball on a rounded rod and in the air. Create a visualization for the model.
2) Design and test a controller for the position of the ball on the rod (both on the flat and the rounded part).
3) Demonstrate throwing a ball from one side of the rod to the other.
4) Design the system to be easy to use in the classroom.
5) Review and complete the documentation for the system.

Bibliography / sources:

[1] M. Gurtner and J. Zemánek, “Ball in double hoop: demonstration model for numerical optimal control,” IFAC-PapersOnLine, vol. 50, no. 1, pp. 2379–2384, Jul. 2017, doi: 10.1016/j.ifacol.2017.08.429.
[2] D. O. Morales, P. L. Hera, and S. U. Réhman, “Generating periodic motions for the butterfly robot,” in 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Nov. 2013, pp. 2527–2532. doi: 10.1109/IROS.2013.6696712.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jiří Zemánek, Ph.D. Department of Control Engineering FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **17.02.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **16.02.2025**

Ing. Jiří Zemánek, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student’s signature

Acknowledgements

I would like to express my sincere gratitude to several people who have contributed to completing this thesis.
I want to thank my supervisor, Ing. Jiří Zemánek, Ph.D., for his support throughout the working process and for providing me with the opportunity to work on this project. His insights and comments, as well as constructive criticism, were crucial to the completion of this thesis.
I also thank the FEE members Ing. Loi Do and Ing. Krištof Pučejdl, who provided me with invaluable advice and were always willing to discuss and give insightful comments.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 26, 2023

_____

Signature

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 26. května 2023

_____

Podpis autora práce

Abstract

This bachelor's thesis aims to enhance an established ball and beam system by introducing a novel mode named Flying Ball and Beam. A designed hybrid system incorporates three distinct mathematical models representing different ball and beam movements. The hardware model is derived from the Flying Ball in Hoop laboratory model established at FEE CTU, which was modified and updated to create the Flying Ball and Beam. A visualization is developed to enable the verification of each mathematical model. Subsequently, three balancing tasks are implemented in Matlab and Simulink and tested on the real model. Matlab and Simulink are the principal software tools utilized in this work. The work culminates in writing documentation detailing the setup and utilization of the new laboratory model.

Keywords: Ball and beam, System modeling, Hybrid system, Visualization, Balancing task

Supervisor: Ing. Jiří Zemánek, Ph.D.
Department of Control Engineering

Abstrakt

Tato bakalářská práce si klade za cíl rozšířit známý systém kulička na tyči o režim s názvem Létající kulička na tyči. Navržený hybridní systém zahrnuje tři matematické modely představující různé pohyby kuličky na tyči. Hardwarový model vychází z laboratorního modelu Létající kulička v obruči navržený na FEL ČVUT, který byl modifikován s cílem vytvořit nový model Létající kulička na tyči. Vytvořená vizualizace systému umožňuje ověření matematických modelů hybridního systému. Následně jsou implementovány tři úlohy vyvažování v prostředí Matlab a Simulink a otestovány na reálném modelu. Matlab a Simulink jsou hlavními nástroji použitými v této práci. Práce je zakončena dokumentací, která popisuje nastavení a spuštění tohoto laboratorního modelu.

Klíčová slova: Kulička na tyči, Modelování systému, Hybridní systém, Vizualizace, Úloha vyvažování

Překlad názvu: Systém létající kulička na tyči

Contents

Figures

Chapter 1

Introduction

The ball and beam system is a classic and well-known problem in control engineering, widely used as a benchmark for developing and evaluating control strategies. The system has been described by a well-known control systems researcher Peter E. Wellstead, who has extensively studied different control systems, including the ball and beam system. He has made significant contributions to its analysis and control; see [1] and [2].

Wellstead describes the ball and beam system as an example of a control system that provides a simple and intuitive way to understand the principles of feedback control while representing a difficult open-loop unstable control problem. The system, depicted in Figure 1.1a, consists of a beam linked directly to a motor shaft or a linkage connecting the beam and a cam driven by a motor. The ball rolls freely on the beam, and the ball's position is controlled by varying the angle of the beam.

Another well-known system described by Wellstead is the ball and hoop system [3]. The system, seen in Figure 1.1b, contains a ball that can freely rotate within a hoop connected to a motor that can apply torque to it. The system represents another interesting control system rich in dynamics and can be, e.g., used to study the dynamics of certain liquid slosh problems.

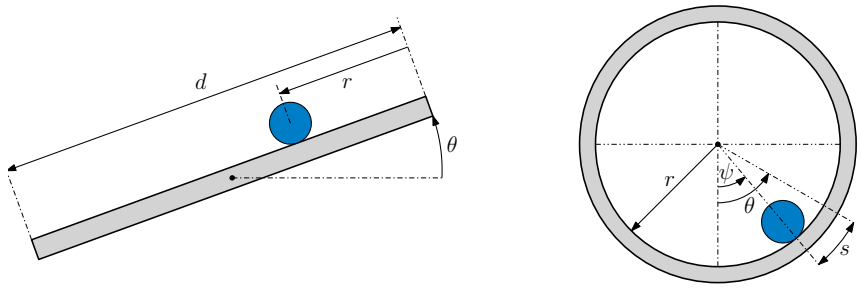
Based on the ball and hoop design, a system called *Flying Ball in Hoop* was created at FEE CTU and originally developed by the AA4CC[1] group members Jiří Zemánek and Martin Gurtner in 2017.

The Flying Ball in Hoop system[2] introduced a more complex task named *flying mode* on top of the relatively simple control of the ball in the hoop. Controlling the hoop's rotation, the ball could fly inside the hoop, demonstrating a more comprehensive range of system modeling and control design possibilities in control engineering.

This bachelor's thesis aims to expand upon this existing laboratory model. The *Flying Ball and Beam* system, which replaces the hoop with a beam attached to the motor, is heavily based on the Flying Ball in Hoop system and serves as the master model from which this work proceeds from [4].

[1] AA4CC, which stands for Advanced Algorithms for Control and Communication, is an academic research group fostered by the Department of Control Engineering at FEE CTU.

[2] More information about the Flying Ball in Hoop system, including the system description and its setup, can be found at `https://aa4cc.github.io/flying-ball-in-hoop`.



(a) : Ball and beam. **(b) :** Ball in hoop.

Figure 1.1: Two examples of control systems studied and described by Peter Wellstead.

The main objective of this thesis is to create a mathematical model for the Flying Ball and Beam system, which will incorporate three control modes and form a hybrid system. The follow-up aims are to design a controller for balancing the ball in two modes on the beam and projecting the ball from one place to another within the beam based on a camera used to track the ball's position. The Flying Ball in Hoop is photographed in Figure 1.2 while the Flying Ball and Beam can be seen in Figure 2.1 in Section 2.1 on the next page.

Resembling the two examples with the flying ball is a similar system called Butterfly robot [5], whose task is to stabilize a ball in periodic motions on a beam suggestive of a butterfly's wings. A similar balancing task could also be considered a ball on plate system [6], where the ball is balanced on a plate that can be tilted along axis x and y using a camera in the same way.

Instructions on setting up and using the Flying Ball and Beam system are described in the system's documentation, available in the Flying Ball and Beam GitHub repository. Links to all software source codes I use in this work, including the thesis repository, can be found in Appendix A.

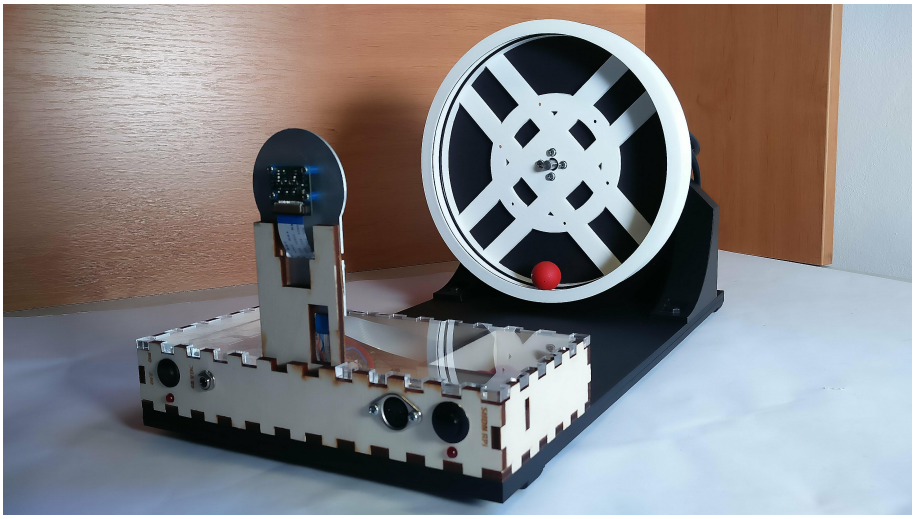


Figure 1.2: Flying Ball in Hoop system developed at FEE.

Chapter 2

System Description

2.1 System Structure

The Flying Ball and Beam system is divided into two separate units. The first unit is a user's computer, and the second is a ball and beam system. The principal parts of the system are a Raspberry Pi device, a motor, a camera, and finally, a ball and a beam. Communication is set between the Raspberry and the user's computer from which the system is operated. This means control design and its implementation are performed on the computer and then deployed on the Raspberry Pi.

Both the camera and the motor are connected to the Raspberry Pi, where the camera is utilized for detecting the ball's position. The motor controls the ball's position by altering the beam's angle using the camera's output. Constituent parts of the system mentioned above are minutely described in Section 3.1.

Providing that contact is ensured, a dashed line in Figure 2.2 shows coordinates of the ball's center of gravity representing the possible ball trajectory around the beam.

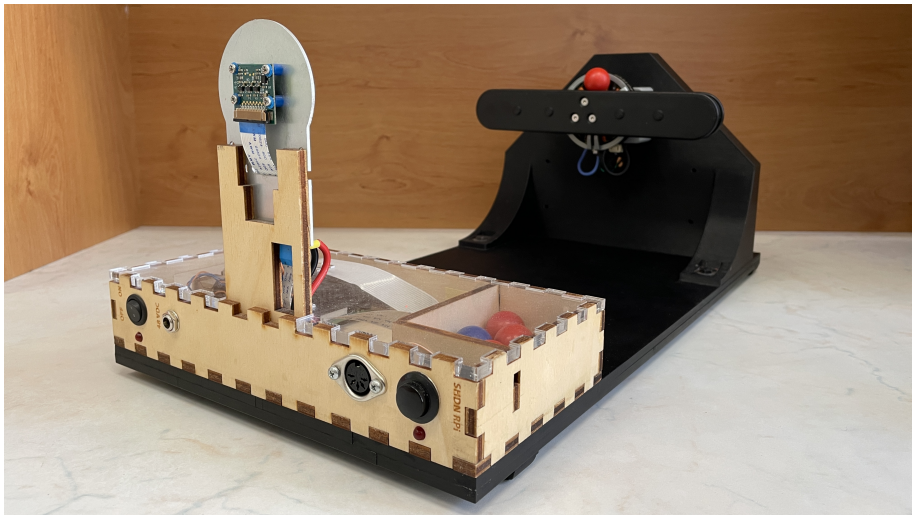


Figure 2.1: Flying Ball and Beam system developed at FEE.

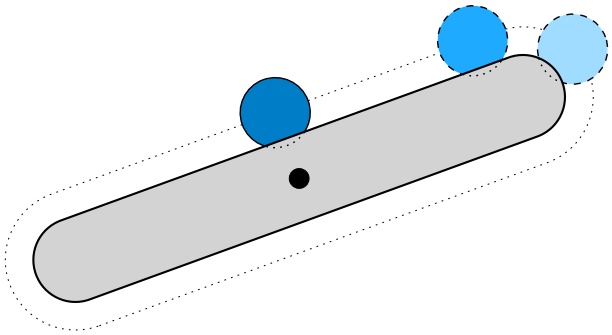


Figure 2.2: Possible trajectory of the ball on the beam.

Dimensions of the ball and the beam are presented in Figure 2.3 with parameters noted in Table 2.1, where a variable index 1 marks out a beam parameter, whereas an index 2 is a ball parameter.

For simplicity and clarity, the beam groove will not be considered for the modeling tasks and will be disregarded in all schemes. This simplification proffers one to consider the distance from the ball center to the beam center pivot as $r_1 + r_2$ in the middle of the beam. This distance is also relevant for the ball-beam distance along the rounded part of the beam. The beam length d_1 from the parameters table is computed as $d_1 = l_1 - 2r_1$.

Finally, I assume the gravitational acceleration to be $\mathrm{g} = 9.807\,\mathrm{m\,s^{-2}}$.

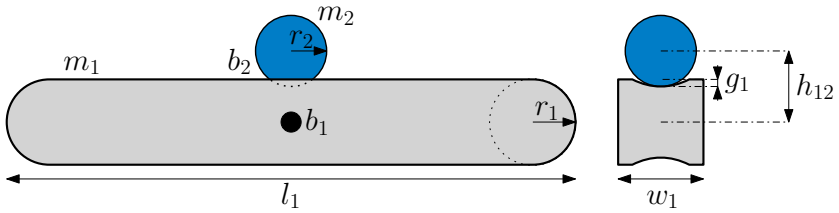


Figure 2.3: Significant system parameters including dimensions, masses, and frictions.

system part	parameter	symbol	value	unit
beam	mass	m_1	64	g
	length	l_1	204	mm
	length	d_1	170.5	mm
	width	w_1	20	mm
	height	h_1	34.5	mm
	radius	r_1	17.3	mm
	groove	g_1	3	mm
ball	mass	m_2	32.7	g
	diameter	d_2	20	mm
	radius	r_2	10	mm
beam-ball	height	h_{12}	24.3	mm

Table 2.1: Enumerated parameters of the system.

Chapter 3

Hardware and Software

3.1 Hardware

The Flying Ball and Beam system consists of a baseboard with a motor holder and an electronics box. The board is a laser-cut plexiglass board coated with matte black paint. A controller and a motor from the MJBOTS company are powered by 24 VDC and are mounted on a holder made of PET plastic. A beam that can freely rotate is attached directly to the motor shaft. The beam has a groove around its whole perimeter in which a metal ball rolls. Two rubber bands are stretched around the perimeter on both sides of the beam serving as high-friction rails and preventing the ball from bouncing in the groove.

The electronics box is a wooden case built from individual rectangular sides and houses a Raspberry Pi, powered by 5 VDC via a USB-C cable. The box also includes a small area to store the ball with additional ones. A plexiglass is used as a cover keeping the box away from possible dust. The last component of the system is a Raspberry camera attached to the box's front side and connected to the Raspberry.

A scheme in Figure 3.1 shows how these key hardware components are connected, forming the Flying Ball and Beam system. Details about the electronic components are mentioned below.

- **Raspberry Pi** – The fundamental system constituent is the Raspberry Pi Model 4 B with a 32-bit Raspberry Pi OS installed. I discovered that a 64-bit operating system could not be used as it does not support the `picamera` package in Python, which is necessary for the RaspiCam camera operation. The Raspberry is connected to the user's computer by an ethernet cable. One can then communicate with the Raspberry via an SSH connection from the computer's terminal window. Due to heating issues, the passive cooling on the Raspberry was insufficient, and I had to add a Raspberry Armor Fan case to cool the device.

- **MJBOTS motor** – For spinning the beam, I use an MJBOTS Moteus high-performance 5208 brushless motor attached to the motor mount on the baseboard.

- **MJBOTS controller** – The motor is controlled using a controller named MJBOTS Moteus r4.11. The communication between the Raspberry and the Moteus motor is arranged by an MJBOTS FDCANUSB device, which provides a USB 2.0 interface to a CAN-FD bus. This setup connects the controller to the Raspberry Pi via a USB-A port.
- **RaspiCam** – The RaspiCam module consists of a Raspberry Pi camera and two rings of LEDs. Twenty-four LEDs, powered by 48 VDC, illuminate the scene, resulting in a better input image. The camera is connected to the Raspberry Pi via a ribbon cable.

3.1.1 System Modification

Initially, the Flying Ball and Beam system fully adopted the hardware setup from the Flying Ball in Hoop platform except for the beam. It used a brushless motor and a controller from the company ODrive to spin the beam.

The motor should have been able to be controlled in three different modes: *positional*, *velocity*, and *current* mode. Even with the help of the ODrive documentation and its online community discourse with various tutorials, I could not resolve how to enable the velocity or current mode. I could only control the motor in the positional mode, which allows one to set a position the motor is supposed to reach. Being unable to use the motor in the current mode could potentially introduce significant limitations in future system control, so I had to find an alternative for the ODrive gear.

Eventually, I replaced the original gear with an MJBOTS motor and controller used in other AA4CC projects. In this case, I could easily set up both positional and current modes in the controller and control the motor.

The motor change meant I had to redesign the beam, as the Moteus motor has different mounting points than the original motor. I designed the beam in Fusion 360, a 3D CAD software developed by Autodesk. The beam design can be found in the system's repository with an illustration in Appendix A.

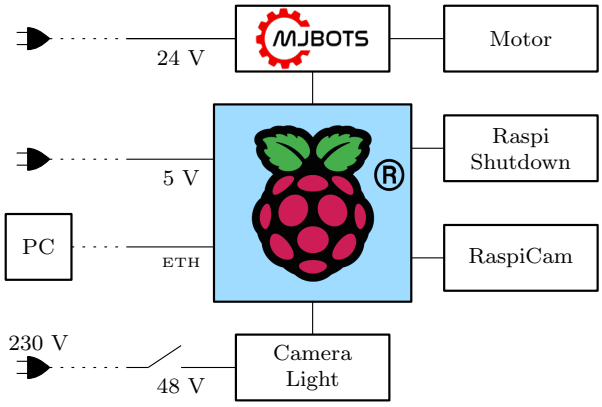


Figure 3.1: Scheme of the system's architecture with its principal components.[1]

[1]Raspberry and MJBOTS company logo images were downloaded from their websites `https://www.raspberrypi.com/trademark-rules` and `https://mjbots.com`.

3.2 Software

The principal software tools employed in this work are Matlab and Simulink. Matlab is used to develop a mathematical model of the Flying Ball and Beam system, model visualization, and identification of parameters. Simulink is used to design and implement control for the ball and beam system. Using Simulink, one connects to the Raspberry, which controls the system.

For a Simulink–Raspberry communication, I use the ERT Linux, which stands for *Minimalist Simulink Coder Target for Linux*. This software, created at FEE, permits Simulink-generated code to be compiled, deployed, and launched on a Linux device - in this case, on the Raspberry Pi.

For motor control, I take advantage of a Moteus API in Simulink devised by Loi Do. The interface includes a System Objects block which allows the user to control the motor within the Simulink workspace.

Lastly, based on image processing, I use RaspiBallPos in Python to detect the ball's position created at FEE. RaspiBallPos, previously utilized in the Flying Ball in Hoop system, enables one to use a System Objects block in Simulink to obtain a real-time ball position from the image.

The Raspberry can be shut down with a button and a Python script, enabling the Raspberry shutdown service introduced in the Flying Ball in Hoop system. The button is connected to the Raspberry via a GPIO pin.

3.3 Simulation Setting

The sampling time of the RaspiCam, including the image processing, is set to 30 ms. Initially, I assumed that the whole system's simulation had to be performed using the same sampling time. However, the control of the balancing tasks was far from fast and smooth with this sampling time.

I then discovered that the simulation sampling time could be chosen independently on the RaspiCam, meaning that the system would receive the same ball's position for the duration of one sampling step of the RaspiCam. The sampling time set to 1 ms was too high and caused aliasing. The simulation frequency was thus empirically set to 5 ms.

3.4 RaspiCam Setting

Before using the RaspiCam, it is necessary to calibrate the camera to capture the ball's position on the beam properly. After turning on the camera with a command in the Raspberry terminal window, the camera is calibrated in a web interface. In the interface, the user can set the number of objects appearing in the image, including their parameters, such as radius. Then, one can adjust the color and white balance of the objects in the image while watching a real-life masked output image from the camera.

3.5 Camera Homography

The output of the RaspiCam is pixel coordinates x and y of the ball's position in the image with the origin pixel $[0, 0]$ in the bottom right corner and resolution of 480 x 480 px. The Raspberry camera delivers the image upside down, resulting in having the origin pixel in the image's bottom right corner.

I used the camera homography $\mathbf{H}$ to map the image coordinates to the world coordinates. The homography is calculated based on a set of calibration points, which are the coordinates on the image plane $\mathbf{x}$ and their corresponding world coordinates $\mathbf{X}$ complying

$$\mathbf{X} = \mathbf{H}\mathbf{x}\,. \tag{3.1}$$

The calibration points are depicted in Figure 3.2. For the homography calculation, I used a homogeneous estimation method based on singular value decomposition (SVD) [7].

Since the beam can rotate, only two colorful markers were individually stuck on one of the two projection points included on one side of the beam. The image plane coordinates were captured for 10 s, then the median value was chosen as the output from the RaspiCam slightly fluctuated. The world coordinates were calculated using goniometric functions with the origin in the center pivot of the beam. Although only four points were necessary, I decided to get twelve of them to obtain better accuracy. The world coordinates were obtained by making combinations from a set of lengths $\mathcal{D} = \{30, 60\}$ mm and a set of angles $\mathcal{A} = \frac{k\pi}{3}$, $k \in \{0, ..., 5\}$. I used a Matlab script named `hom_calib` available in the RaspiBallPos repository to get the homography.

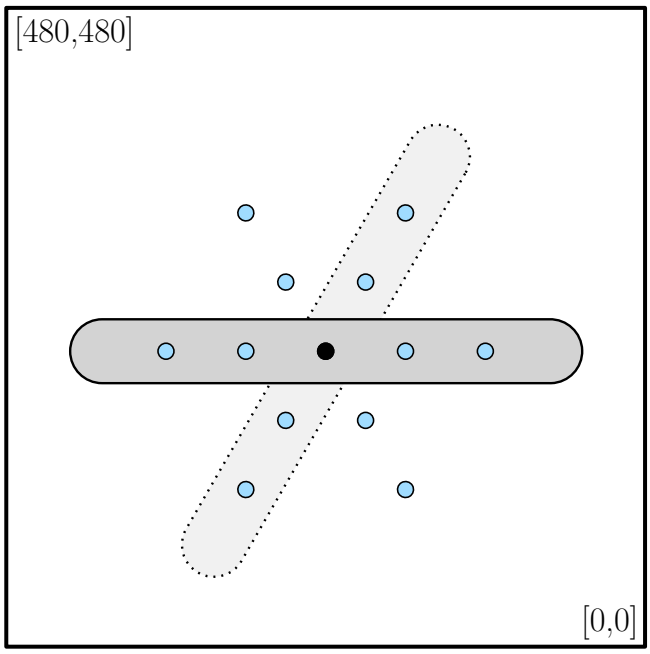


Figure 3.2: Calibration points utilized for the homography with the beam in positions where the markers were captured.

Chapter 4

Parameters Identification

Having measured parameters describing the Flying Ball and Beam system enumerated in Table 2.1, I will now determine the remaining unknown parameters: motor friction b_1 and ball friction b_2. I will also verify the motor torque by comparing the input reference torque with the real torque spinning the motor with the beam.

4.1 Motor and Beam Friction

Regarding the ball friction coefficient b_2, I consider the same ball friction previously computed for the Flying Ball in Hoop since I use an identical ball as in the Flying Ball in Hoop system, and the beam is printed from the same material as the hoop. The friction b_2 is thus given by

$$b_2 = 2.574 \cdot 10^{-6}\,\mathrm{N\,m\,s\,rad^{-1}}\,. \tag{4.1}$$

The friction of the Moteus motor b_1 is left to be determined. For its estimation, I used a method called *Grey-Box Model Estimation* in Matlab, which enables one to estimate coefficients of linear and nonlinear differential equations. Considering the motor friction linear, I derived the differential equation representing the motor from a bond graph[1], as shown in Figure 4.1.

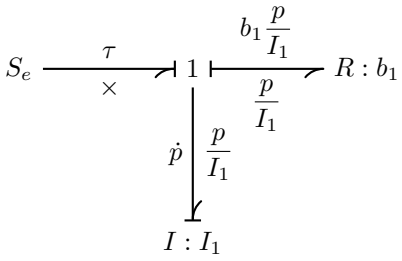


Figure 4.1: Bond graph model of the motor.

[1]The bond graph was generated using a LaTeX Bond Graph Package created by Martin Gurtner. The package is available at `https://github.com/martingurtner/mgbondgraph`.

The differential equation is derived from the bond graph as

$$\dot{p} = -\frac{b_1}{I_1} p + \tau \,, \tag{4.2}$$

where p is generealized momentum and τ the input torque. Firstly, I had to create a rectangular waveform of various input torques $\tau \in [-0.05, 0.05]\,\mathrm{N\,m}$, which spun the motor. Considering the equation (4.2), I decided to estimate not only the motor friction b_1 but also the moment of inertia of the beam I_1.

Recording the input torque waveform and the spinning motor's corresponding velocity in Simulink, the two parameters were estimated using `idnlgrey` and `nlgreyest` functions. Firstly, `idnlgrey` created a linear grey-box model of the motor and `nlgreyest` then estimated the two coefficients.

After that, I used a different set of input torques to simulate the motor model with the estimated parameters using `ode45`, a numerical method for solving differential equations, and to spin the hardware motor. Comparing the velocities from the simulation and the real data, I then slightly adjusted the parameters' values. The motor friction b_1 and moment of inertia of the beam I_1 are thus given by

$$b_1 = 15.893 \cdot 10^{-4}\,\mathrm{N\,m\,s\,rad^{-1}} \qquad \text{and} \qquad \mathrm{I}_1 = 8.860 \cdot 10^{-4}\,\mathrm{kg\,m^2}\,. \tag{4.3}$$

A comparison of the velocities from the simulation and the recorded real data is shown in Figure 4.2.

The inaccuracy in the estimation is likely caused since I approximated the friction with a linear parameter to model the motor and did not consider cogging torque characteristic of a brushless DC motor. The model is, however, helpful in estimating the behavior of the Moteus motor and assessing the unknown parameters. Regardless of the simplified model, I will consider the estimated parameters to enumerate nonlinear models of the hybrid system developed in Chapter 5.

4.2 Motor Torque Verification

In this section, I want to verify that the real torque, which spins the motor, corresponds to the input reference torque. I compare these two quantities with a digital force gauge named SHITO FS-20N.

To compute the torque τ from the force of the motor F measured by the force gauge, I consider an equation

$$\tau = Fl\sin(\psi)\,, \tag{4.4}$$

where l is the distance measured from the beam center pivot to a point where the force is applied to the sensor of the force gauge. Angle ψ is the angle between the beam and the gauge's sensor. I used a rectangular waveform with different amplitudes $\tau \in [0.02, 0.7]\,\mathrm{N\,m}$ as a reference torque. The lower limit of the interval is the minimum torque required to spin the motor. Torque $\tau = 0.01\,\mathrm{N\,m}$ was too low and could not move the motor. The upper limit

$\tau = 0.7\,\mathrm{N\,m}$ was chosen because I could no longer hold the beam and carry out the force measurement with the device for higher torques.

The force was measured for $l = 0.06\,\mathrm{m}$ with the beam in the vertical position so that $\psi = \frac{\pi}{2}$. I verified that the measured force corresponds to the reference torque. The slight inaccuracy is likely caused by the force gauge not being perfectly rectangularly aligned with the beam while measuring. The comparison of the reference and measured torque is shown in Figure 4.3.

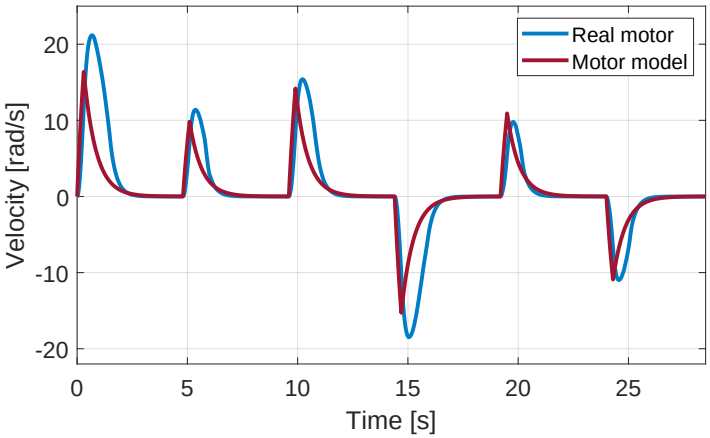


Figure 4.2: Comparison of the simulated model's velocity determined using the identified parameters and the real motor's velocity.

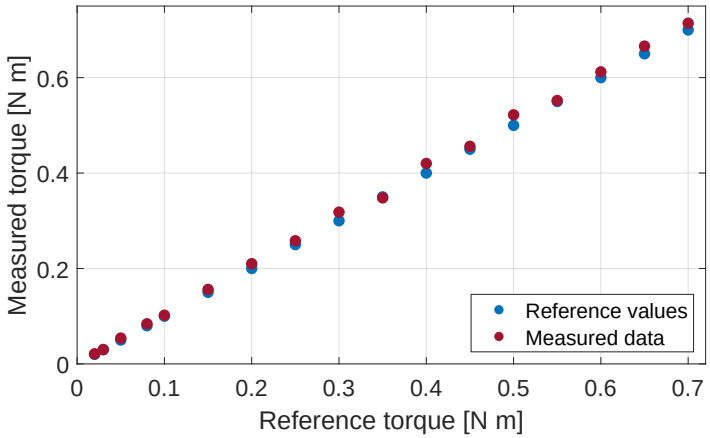


Figure 4.3: Comparison of the input reference torque and the real torque measured with the force gauge.

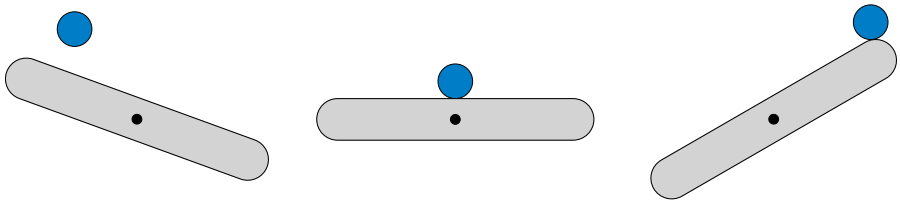
Chapter 5

Hybrid System

5.1 Hybrid System Description

As the introduction mentions, I will incorporate three modes describing different ball motions on the beam. The three separate modes signify creating a system that contains discrete-valued signals with if-then-else conditions and also involves real-valued signals. Such a system is called a *hybrid system* and describes dynamical interactions between continuous and discrete signals in one common framework [8].

The modes, which represent the components of the hybrid system, are namely a *balancing mode*, a *revolving mode*, and a *projectile mode*. All three modes are portrayed in Figure 5.1. The aim is to create a model of the hybrid system, which will be able to transition between these modes depending on the ball's position in relation to the beam. Figure 5.2 exhibits a schematic diagram comprising all three modes of the model, with index i denoting a specific mode and γ_{ij} signifying a guard condition for transitioning from the current mode i to the next mode j. All three modes will be delineated now in detail below. To describe the hybrid system, I distinguish two parts of the beam: the flat part of length d_1 and the rounded part of radius r_1.



(a) : Projectile mode. **(b) :** Balancing mode. **(c) :** Revolving mode.

Figure 5.1: Three modes forming the hybrid system.

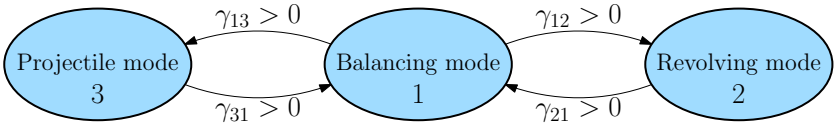


Figure 5.2: Diagram of the hybrid system, including the guard conditions.

- **Balancing mode** – This is the fundamental mode of the hybrid system, where the goal is to balance the ball anywhere on the flat part of the beam. Depending on the RaspiCam positional output of the ball, the motor with the beam tilts and moves the ball into the desired position.

  The system always begins with this mode on start-up and then, if desired, changes the mode to one of the other two. The system enters this mode before and after the revolving, and the projectile mode run.

- **Revolving mode** – This mode commences with a slight beam tilt with two possible scenarios when the ball approaches one of the beam's ends. The first option is that the beam makes a half rotation (π rotation), and the ball simultaneously traverses the rounded part and reaches the other flat part of the beam. The balancing mode is activated afterward to balance the ball on the beam. The second option is to balance the ball on the rounded part in the beam's vertical position.

  It is essential that the ball stays in contact with the beam and does not leave the groove throughout any of the two motions.

- **Projectile mode** – With a rapid beam tilt, the beam projects the ball, starting a free-fall motion. The beam returns to its horizontal position. Before the ball touches the beam, the beam slightly tilts in the same direction as the landing ball to dampen the fall. The balancing mode is then activated to balance the ball.

  The ball can be projected from any point on the beam. The landing point on the flat part of the beam is determined based on the tilt causing the projectile motion.

Derivations of the model motion equations will be described in Sections 5.2, 5.3, and 5.4. I will use the *Euler-Lagrange formalism* to derive the equations and model the individual modes. Before delving into the derivations, I would like to break down the selection of the coordinate systems in which the models will be described.

Firstly, polar coordinates are an obvious choice for the revolving mode, as the mode represents a circular motion. The polar coordinates will also be used for the balancing mode. Although expressing all three modes in the same coordinate systems would be convenient, Cartesian coordinates are more suitable for the projectile mode because the mode's motion equations will be linear. With this choice of coordinates, one has to reconcile himself to a transformation of coordinates when transitioning between two modes with different coordinates. The transformations will be described in Section 5.5.

To simplify the model equations, I will not state the time dependence of generalized coordinates. Also, I will not display the angle unit rad for enumerated angles for the rest of the text.

For a beam moment of inertia I_1, I assume the beam to be a bar of length l_1 and mass m_1 with the axis of rotation going through the center pivot of the bar. For the ball, I consider a moment of inertia I_2 of a homogeneous

sphere with radius r_2 and mass m_2 with the axis of rotation going through the sphere's center of gravity. The formulas and values for both moments are

$$I_1 = \frac{1}{12} m_1 l_1^2 = 2.220 \cdot 10^{-4}\,\mathrm{kg\,m^2}\,, \tag{5.1}$$

$$I_2 = \frac{2}{5} m_2 r_2^2 = 1.308 \cdot 10^{-6}\,\mathrm{kg\,m^2}\,. \tag{5.2}$$

Having both estimated and calculated the beam moment of inertia, I can now draw a comparison between the estimated (4.3) and the computed (5.2) value. One can see that the moments' orders correspond; the estimated value is, however, four times bigger. For the following computations, I will consider the moment of inertia discovered with the Grey-Box Model Estimation as the formula (5.2) is only a theoretical approximation neglecting the exact dynamics of the bar.

5.2 Balancing Mode

In this section, I will derive the motion equations for balancing the ball on the beam based on [9]. I choose length d and angle θ as generalized coordinates, resulting in $\mathbf{q} = \begin{pmatrix} \theta & d \end{pmatrix}^\top$. The coordinate d corresponds to the distance of the ball center to a position where the ball is situated in the middle of the beam's flat part. Angle θ is the angle of the beam w.r.t. the horizontal axis of the world frame. The coordinates are shown in Figure 5.3.

The kinetic co-energy is

$$T^* = \frac{1}{2} I_1 \dot{\theta}^2 + \frac{1}{2} m_2 (d^2 \dot{\theta}^2 + \dot{d}^2) + \frac{1}{2} I_2 (\dot{\theta} + \dot{\varphi})^2\,. \tag{5.3}$$

A rotation of the ball φ can be expressed as

$$\varphi = \frac{d}{r_2} + \theta\,. \tag{5.4}$$

The potential energy is given by

$$V = m_2 g h\,, \tag{5.5}$$

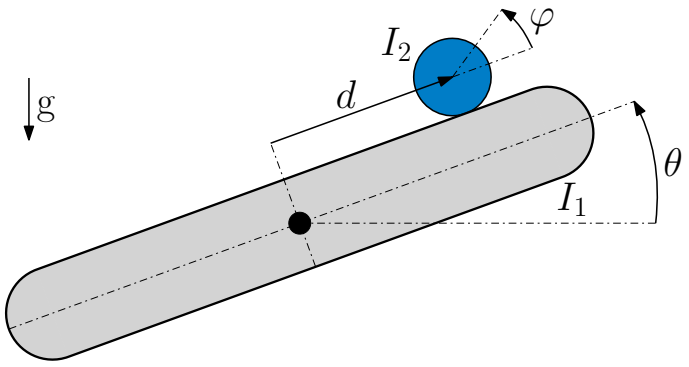


Figure 5.3: Balancing the ball on the beam denoted with the generalized coordinates θ and d and the moments of inertia I_1 and I_2.

where

$$h = d\sin(\theta) + (r_1 + r_2)\cos(\theta)\,. \tag{5.6}$$

The dissipation function of the ball and the beam is

$$D = \frac{1}{2}b_1\dot{\theta}^2 + \frac{1}{2}b_2\dot{\varphi}^2\,. \tag{5.7}$$

Considering the Lagrangian

$$\mathcal{L} = T^* - V\,, \tag{5.8}$$

I can now write the Euler-Lagrange equations

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{\theta}} - \frac{\partial\mathcal{L}}{\partial\theta} + \frac{\partial D}{\partial\dot{\theta}} = \tau\,, \tag{5.9}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{d}} - \frac{\partial\mathcal{L}}{\partial d} + \frac{\partial D}{\partial\dot{d}} = 0\,, \tag{5.10}$$

where τ represents the external torque applied to the beam from the motor. I will not state the differential equations directly as I think expressing the equations in a matrix form will be more apparent. For this purpose, I use a dynamic equation

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{Q}\,, \tag{5.11}$$

where $\mathbf{M}$ is the inertia matrix, $\mathbf{C}$ is the Coriolis matrix, $\mathbf{G}$ is the gravity vector, and $\mathbf{Q}$ is the vector of external forces.

Substituting the equations (5.9) and (5.10) into the matrices, one gets

$$\begin{aligned}
\mathbf{M}(\mathbf{q}) &= \begin{bmatrix} I_1 + 4I_2 + m_2d^2 & 2\dfrac{I_2}{r_2} \\ 2\dfrac{I_2}{r_2} & \dfrac{I_2}{r_2^2} + m_2 \end{bmatrix}, \\
\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) &= \begin{bmatrix} b_1 + b_2 + 2m_2\dot{d}d & \dfrac{b_2}{r_2} \\ \dfrac{b_2}{r_2} - m_2\dot{\theta}d & \dfrac{b_2}{r_2^2} \end{bmatrix}, \\
\mathbf{G}(\mathbf{q},\dot{\mathbf{q}}) &= \begin{bmatrix} m_2\mathrm{g}d\cos(\theta) - m_2\mathrm{g}(r_1 + r_2)\sin(\theta) \\ m_2\mathrm{g}\sin(\theta) \end{bmatrix}, \\
\mathbf{Q} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix}\tau\,.
\end{aligned} \tag{5.12}$$

5.2.1 Balancing Mode Linearization

Unlike the projectile mode, the balancing mode's equations are nonlinear. This means that the mode linearization in a relevant equilibrium point is required to design a control for the system in this mode. The linear model will be described using a state-space representation of the form

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}\,, \\
\mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}\,,
\end{aligned} \tag{5.13}$$

where $\mathbf{x} = \begin{pmatrix} \theta & \dot{\theta} & d & \dot{d} \end{pmatrix}^\top$ is the vector of the system's states and $\mathbf{u}$ the system's input torque. Deriving a second-order system from the matrices (5.12) and creating the state-space model, one can linearize the model around the equilibrium point $\tilde{\mathbf{x}} = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^\top$. The enumerated state-space matrices of the linearized model are

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 11.881 & -0.484 & -360.429 & -3.720 \\ 0 & 0 & 0 & 1 \\ -7.073 & -0.166 & 2.060 & -16.847 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1123.9 \\ 0 \\ -6.423 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{5.14}$$

5.3 Revolving Mode

For the revolving mode, a sketch of the ball in a revolving motion is shown in Figure 5.4 with angles α and θ chosen as generalized coordinates having $\mathbf{q} = \begin{pmatrix} \theta & \alpha \end{pmatrix}^\top$. Again, angle θ is the angle of the beam w.r.t. to the horizontal axis of the world frame. Angle α is the angle of the ball center position w.r.t. the center of the beam's rounded part.

The ball position on one of the beam's rounded parts can be determined with the generalized coordinates, where the position of the rounded part's center is labeled with coordinates x_1 and y_1. Knowing this position, one can then define the ball center position with coordinates x_2 and y_2. Both positions are highlighted with a light blue mark in the figure.

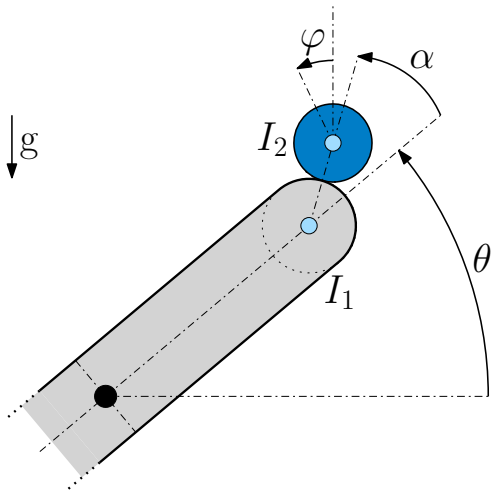


Figure 5.4: Revolving ball on the beam denoted with the generalized coordinates θ and α and the moments of intertia I_1 and I_2.

The computation of the coordinates is given by

$$\begin{aligned} x_1 &= \frac{d_1}{2}\cos(\theta)\,, & y_1 &= \frac{d_1}{2}\sin(\theta)\,, \\ x_2 &= x_1 + (r_1 + r_2)\cos(\theta + \alpha)\,, & y_2 &= y_1 + (r_1 + r_2)\sin(\theta + \alpha)\,. \end{aligned} \tag{5.15}$$

The ball angle φ can be expressed as

$$\varphi = \theta + \alpha\frac{r_1}{r_2}\,. \tag{5.16}$$

The kinetic co-energy is

$$T^* = \frac{1}{2}I_1\dot{\theta}^2 + \frac{1}{2}m_2(\dot{x}_2^2 + \dot{y}_2^2) + \frac{1}{2}I_2(\dot{\theta} + \dot{\varphi})^2\,. \tag{5.17}$$

The potential energy is

$$V = m_2 g y_2\,. \tag{5.18}$$

The dissipation function of the ball and the beam is

$$D = \frac{1}{2}b_1\dot{\theta}^2 + \frac{1}{2}b_2\dot{\varphi}^2\,. \tag{5.19}$$

Reckoning up the Lagrangian (5.8), I can compute the Euler-Lagrange equations of the generalized coordinates θ and α, similarly to (5.9) and (5.10). The matrices of the two differential equations describing the revolving motion are

$$\begin{aligned} \mathbf{M}(\mathbf{q}) &= \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}\,, \\ \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) &= \begin{bmatrix} b_1 + b_2 - m_2 R d_1 \sin(\alpha)\dot{\alpha} & b_2\frac{r_1}{r_2} - m_2 R\frac{d_1}{2}\sin(\alpha)\dot{\alpha} \\ b_2\frac{r_1}{r_2} + m_2 R\frac{d_1}{2}\sin(\alpha)\dot{\theta} & b_2\frac{r_1^2}{r_2^2} \end{bmatrix}\,, \\ \mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}) &= \begin{bmatrix} m_2 g\frac{d_1}{2}\cos(\theta) + m_2 g R\cos(\theta + \alpha) \\ m_2 g R\cos(\theta + \alpha) \end{bmatrix}\,, \\ \mathbf{Q} &= \begin{bmatrix} 1 \\ 0 \end{bmatrix}\tau\,, \end{aligned} \tag{5.20}$$

where

$$\begin{aligned} R &= r_1 + r_2\,, \\ m_{11} &= I_1 + 4I_2 + m_2 R^2 + m_2\frac{d_1^2}{4} + m_2 R d_1\cos(\alpha)\,, \\ m_{12} &= 2I_2\frac{r_1}{r_2} + m_2 R^2 + m_2 R\frac{d_1}{2}\cos(\alpha)\,, \\ m_{21} &= 2I_2\frac{r_1}{r_2} + m_2 R^2 + m_2 R\frac{d_1}{2}\cos(\alpha)\,, \\ m_{22} &= I_2\frac{r_1^2}{r_2^2} + m_2 R^2\,. \end{aligned} \tag{5.21}$$

5.3.1 Revolving Mode Linearization

For the revolving mode, the required linearization of the state-space description using the matrices (5.20) is analogous to the balancing mode linearization. The vector of the system's states is $\mathbf{x} = \begin{pmatrix} \theta & \dot{\theta} & \alpha & \dot{\alpha} \end{pmatrix}^\top$ and the equilibrium point is $\tilde{\mathbf{x}} = \begin{pmatrix} \frac{\pi}{2} & 0 & 0 & 0 \end{pmatrix}^\top$. The enumerated state-space matrices of the linearized model are

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 3.916 & 0.023 & -25.945 & 0.788 \\ 0 & 0 & 0 & 1 \\ 295.617 & -4.812 & 406.644 & -11.085 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1092.3 \\ 0 \\ -4061.2 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{5.22}$$

5.4 Projectile Mode

For the projectile motion of the ball, a ball position x, y and a ball angle φ will be considered generalized coordinates meaning $\mathbf{q} = \begin{pmatrix} x & y & \varphi \end{pmatrix}^\top$. Angle φ is the angle of the ball w.r.t. the vertical axis of the ball frame.

The kinetic co-energy is

$$T^* = \frac{1}{2} m_2 (\dot{x}^2 + \dot{y}^2) + \frac{1}{2} I_2 \dot{\varphi}^2 \,. \tag{5.23}$$

The potential energy is given by

$$V = m_2 g y \,. \tag{5.24}$$

This model will not consider the ball's air resistance in the free-fall motion. Because of the mass and the size of the metal ball, the air resistance can be omitted without any detriment to correctness resulting in $D = 0$.

With (5.8), the Euler-Lagrange equations are

$$m_2 \ddot{x} = 0 \,, \qquad m_2 \ddot{y} + m_2 g = 0 \,, \qquad I_2 \ddot{\varphi} = 0 \,. \tag{5.25}$$

The equations (5.25) can be modified to get their final form. The equations of the ball in the free-fall motion are thus given by

$$\ddot{x} = 0 \,, \qquad \ddot{y} = -g \,, \qquad \ddot{\varphi} = 0 \,. \tag{5.26}$$

As I mentioned in the description of the hybrid system, the choice of the Cartesian coordinates for this mode signifies that linearization of the projectile mode is unnecessary because the equations (5.26) are already linear.

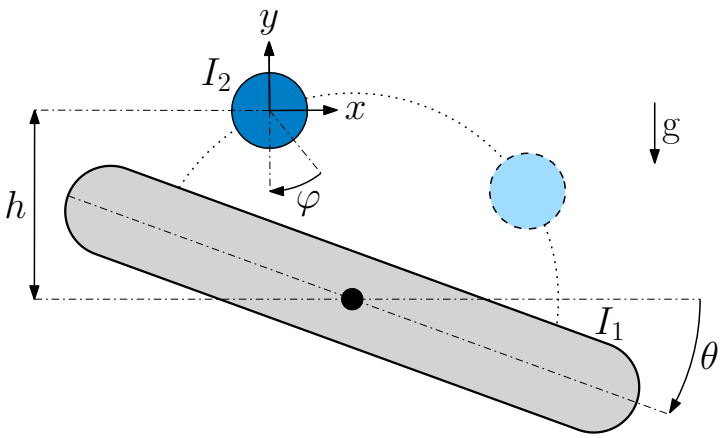


Figure 5.5: Projecting the ball on the beam denoted with the generalized coordinates x, y and φ and the moments of inertia I_1 and I_2.

5.5 Coordinates Transformation

Having computed the homography in 3.5, the origin of the world plane **O** is set to the center pivot of the beam. To control the Flying Ball and Beam system, one has to compute two transformations to obtain the generalized coordinates **q** necessary for the balancing, revolving, and projectile modes.

As the angle θ is obtained directly from the motor block in Simulink, the length d is the only unknown coordinate for the balancing mode. The coordinate d is considered to be a *signed distance*, which is given by

$$d = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}^{\top} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} P_x \\ P_y \end{pmatrix} \right) , \tag{5.27}$$

where x and y are the Cartesian coordinates of the ball. The point **P** is calculated as

$$\begin{pmatrix} P_x \\ P_y \end{pmatrix} = \mathbf{R} \begin{pmatrix} 0 \\ h_{12} \end{pmatrix} , \tag{5.28}$$

where **R** is the 2-dimensional rotation matrix. For the revolving mode, the angle α is given by

$$\alpha = -\operatorname{atan}\left(\frac{x - P_x}{y - P_y} \right) , \tag{5.29}$$

where

$$\begin{pmatrix} P_x \\ P_y \end{pmatrix} = \mathbf{R} \begin{pmatrix} 0 \\ \frac{d_1}{2} \end{pmatrix} . \tag{5.30}$$

The negative sign in (5.29) is added because of the angle α orientation in Figure 5.4. No transformation is needed for the projectile mode because the ball position x and y provide the generalized coordinates.

The ball rotation φ cannot be precisely determined or computed during the projectile motion as the camera's output is only 2-dimensional. Still, the rotation of the ball could be estimated, which is, however, out of the scope of this work.

5.6 Transitioning between Modes

The hybrid system was implemented using a *Stateflow diagram* in Simulink. Stateflow (SF) provides a graphical language that includes state transition diagrams, flow charts, state transition tables, and truth tables. One can use Stateflow to describe how Matlab algorithms and Simulink models react to input signals, events, and time-based conditions [10].

The SF diagram of the hybrid system consists of three states, which correspond to the three modes of the Flying Ball and Beam system. The states are represented by the state-space description of the modes, while a set of guard conditions triggers the transitions between the states (see Figure 5.2) analyzed below. Every transition includes the guard condition and generalized coordinates transformations between the leaving and the upcoming state. The initial values of the upcoming state's transformed coordinates are marked with index 0.

- **Transition** γ_{12} – The system switches from the balancing mode to the revolving mode the moment the ball reaches the end of the beam's flat part, where the ball starts the revolving motion. The modes transition is executed if the condition γ_{12} is satisfied:

$$\begin{aligned} \gamma_{12}(\mathbf{q}) &= |d| - \frac{d_1}{2} > 0\,, \\ \alpha_0 &= \pm\frac{\pi}{2}\,, \end{aligned} \tag{5.31}$$

  where the sign of α_0 depends on the direction of the ball's movement. If the ball moves towards the right end of the beam, the distance d is positive, and the sign of α_0 is positive, too. For the ball moving towards the left, it is vice versa. The angle θ remains unchanged.

- **Transition** γ_{21} – The system switches from the revolving mode back to the balancing mode as soon as the ball finishes circumscribing the rounded part of the beam and returns to the flat part of the beam. The modes transition is executed if the condition γ_{21} is satisfied:

$$\begin{aligned} \gamma_{21}(\mathbf{q}) &= |\alpha| - \frac{\pi}{2} > 0\,, \\ d_0 &= \pm\frac{d_1}{2}\,, \end{aligned} \tag{5.32}$$

  where the sign of d_0 depends on the direction of the beam's rotation. If the beam rotates clockwise, the sign of d_0 is positive, and vice versa. The angle θ remains unchanged.

- **Transition** γ_{13} – As the description of the projectile mode states, the ball can be projected from any point on the beam's flat part. As soon as the ball approaches the desired point, where the ball is released, the system switches from balancing to projectile mode. The modes transition is executed if the condition γ_{13} is satisfied:

$$\begin{aligned} \gamma_{13}(\mathbf{q}) &= d + s - d_p > 0\,, \\ x_0 &= d\cos(\theta) - h_{12}\sin(\theta)\,, \\ y_0 &= d\sin(\theta) + h_{12}\cos(\theta)\,. \end{aligned} \tag{5.33}$$

  where d_p is the distance between the desired release point and the center ball position on the beam. The distance s is a non-zero threshold, which signifies the approach distance to the release point.

- **Transition** γ_{31} – When the ball approaches the beam after reaching the apex point of the projectile motion, the system switches from the projectile mode back to the balancing mode. This switch sets in right before contact between the ball and the beam emerges. The modes transition is executed if the condition γ_{31} is satisfied:

$$\begin{aligned} \gamma_{31}(\mathbf{q}) &= s - p > 0\,, \\ d_0 &= x\,, \\ \theta_0 &= 0\,, \end{aligned} \tag{5.34}$$

  where p is the distance between the beam and the ball, and s is a non-zero threshold. Calculating a minimum distance between a line segment and a point represents the distance p between the ball and the beam. The line segment corresponds to the flat part of the beam, and the point is the ball position. The calculation of this distance is done by using vectors [11].

  When the ball lands back on the beam, one can presuppose that the beam angle would be close to zero, which signifies

$$\theta \approx 0 \qquad \Longrightarrow \qquad d \approx x\,. \tag{5.35}$$

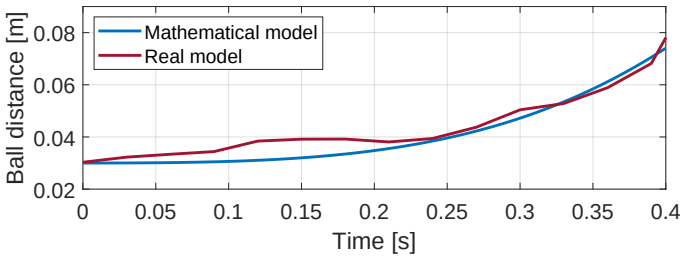
5.7 Modes Simulation

To simulate the individual modes of the hybrid system and verify the hybrid system's correctness, I intended to use Runge-Kutta 4th and 5th order method. Beginning with the balancing mode, I tried to simulate the nonlinear model of the mode described with the state-space vector.

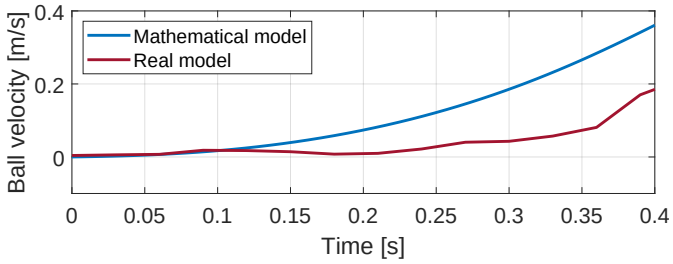
To simulate the model, I used `ode45` with a time span of the system's sampling time. The simulation, however, proved to be unsuccessful as the simulation did not finish. To simulate the model, I then utilized *Euler-Lagrange tool* package[1], which enables one to generate a state-space vector

[1]The package can be downloaded from MathWorks File Exchange site: `https://www.mathworks.com/matlabcentral/fileexchange/49796-euler-lagrange-tool-package`.

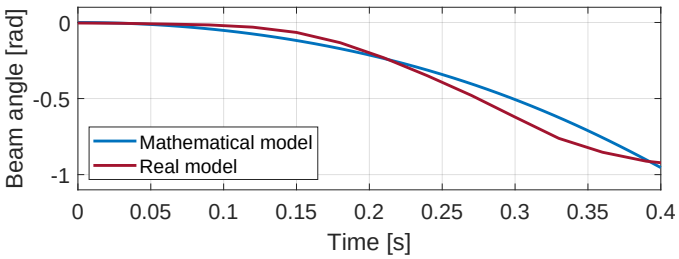
of the model from a kinetic co-energy, potential energy, and the dissipation function. Using this tool, the simulation of the model emerged to be successful; for the mathematical model of the balancing mode, both frictions b_1 and b_2 were fine-tuned to match the real model. A comparison of the simulated mathematical model and the real model responses to an initial state $\mathbf{x}_{\mathrm{BM},0} = \begin{pmatrix} 0 & 0 & d & 0 \end{pmatrix}^\top$, where $d = 30\,\mathrm{mm}$, is shown in Figure 5.6. At the beginning of the simulation, the ball is placed on the beam to the distance d. The beam overbalances because of zero input torque, and the ball rolls down the groove.



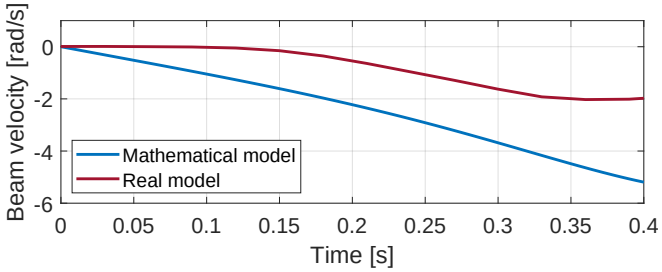
(a) : Ball distance d.



(b) : Ball velocity $\dot{d}$.



(c) : Beam angle θ.



(d) : Beam angular velocity $\dot{\theta}$.

Figure 5.6: Comparison of the simulated mathematical model and the real model responses to the initial state $\mathbf{x}_{\mathrm{BM},0}$.

The simulation verifies the mathematical model of the balancing mode compared to the real model. While the ball distance d and beam angle θ correspond, the difference in the ball velocity $\dot{d}$ and the beam angular velocity $\dot{\theta}$ of the mathematical and real model appears to be more significant than I initially assumed. This might be because the mathematical model is not entirely accurate in describing the ball and beam motion as it relies on simplifications and approximations in defining the exact mode's dynamical behavior. The second reason for this difference could be an improperly chosen simulation. Using the input torque in the simulation and simulating the ball motion on the beam could demonstrate the model's behavior more accurately.

Simulating the revolving mode, I wanted to verify the model's behavior for the initial state $\mathbf{x}_{\mathrm{RM},0} = \begin{pmatrix} \frac{\pi}{2} & 0 & 0 & 0 \end{pmatrix}^\top$ with no input torque as this state represents an unstable equilibrium of the model. First, the angles and angular velocities of the ball and the beam are zero and remain constant. In $t = 2.5\,\mathrm{s}$, the ball and the beam leave the equilibrium, ending up in a chaotic motion. This behavior was expected, as the model reflects only the ball's motion with the ball angle $\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. Because of this uncertainty in the model's correctness, the graphs are not presented.

The simulation of the projectile mode is more straightforward as the model is linear. For the simulation of the projectile mode, the initial state is $\mathbf{x}_{\mathrm{PM},0} = \begin{pmatrix} 0 & 0 & y & 0 & 0 & 0 \end{pmatrix}^\top$, where $y = 0.1\,\mathrm{m}$. Omitting the ball's air resistance, the model only depends on the gravitational acceleration g. The mathematical mode simulation shows the ball's free-fall motion in Figure 5.7.

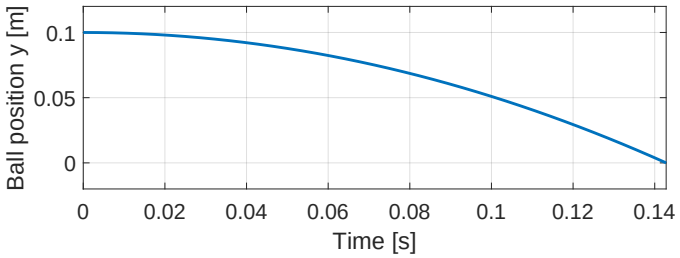


Figure 5.7: Simulated ball projectile motion response to the initial state $\mathbf{x}_{\mathrm{PM},0}$.

Chapter 6

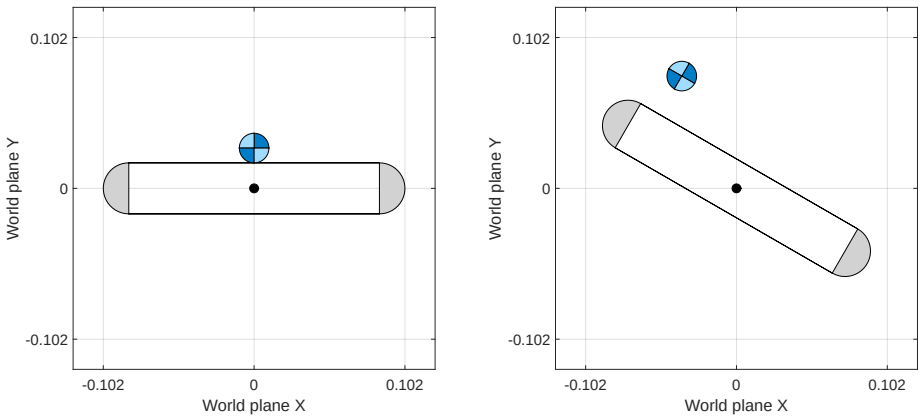
System Visualization

The particular modes of the system are visualized using Matlab. The ball and the beam, portrayed using different geometric shapes with `linspace`, are initially drawn by `fill` function. The ball comprises four quarter-circle arcs, and two semicircles with two line segments are used to portray the beam. A small circle in the beam's center marks the beam's center pivot point.

The portrayal of the two system's objects is not deleted and redrawn. Still, the properties of the objects are changed using the `set` function to increase the visualization's performance. The changes in their properties are then updated with `drawnow`. The visualization of the ball with the beam both in the initial position on the beam's flat sides and in the projectile motion is shown in Figure 6.1.

Both objects are updated on every iteration with a transformation matrix. The beam's parameter is the angle θ, and the ball's parameters are the Cartesian coordinates x and y and the angle φ.

To visualize the Flying Ball and Beam system, a class named `Trajectory` prepares the input data of the visualization. The class is initialized with the system's generalized coordinates θ and d, the input torque, and the time vector.



(a) : The ball and the beam in a steady state position with $\theta = 0$. **(b) :** The ball and the beam in a projectile mode motion with $\theta = -\pi/6$.

Figure 6.1: Visualization of the Flying Ball and Beam performed in Matlab.

For the balancing mode, the angle φ is calculated from the generalized coordinates d and θ using the equation 5.4. The time is sampled with the system's sampling time, and the coordinates and the input torque are interpolated with `interp1`. Lastly, the ball coordinates x and y are calculated using a rotation matrix based on the two generalized coordinates.

Chapter 7

Control Tasks

7.1 Cascade Motor Control

As the subsection 3.1.1 mentions, the motor can be controlled either in a positional or a current mode. Considering the positional mode, the input to the motor's Matlab System Object in Simulink is the desired angle θ of the beam, whereas, for the current control, one sets the input torque τ.

The positional mode is, however, not convenient for control tasks such as balancing the ball. With this motor setting, I did not manage to control and balance the ball efficiently in any desired position on the beam. Therefore, I decided to switch to the current control mode. To control the motor in the current mode, one has to set the motor PID constants to zero during the Moteus controller configuration. The configuration is performed using the Moteus gui named `tview`, which lets one configure and inspect the state of the controller.

The current control requires setting a *cascade control* structure to control the motor's torque, i.e., the moment of the motor, by choosing a reference position. The reference position is the beam angle θ. The structure comprises two closed loops, both controlled by a PD controller, where the inner loop is a moment control loop, and the outer one is a speed control loop.

Initially, I individually implemented the inner loop with one controller and tuned its proportional and derivative constant. As reference velocities, I used sine and pulse waveforms with various amplitudes. After I set the inner loop controller, the outer loop with the second controller was added to the scheme and tuned likewise.

7.2 Flat Part Balancing

The first task of the system control is balancing the ball in any position on the flat part of the beam so that $\theta = 0$. The system's architecture implemented in Simulink is depicted in Figure 7.1, and all function blocks used in the control scheme are described in detail below. Balancing the ball in a desired position on the beam by following a reference position is displayed in Figure 7.3.

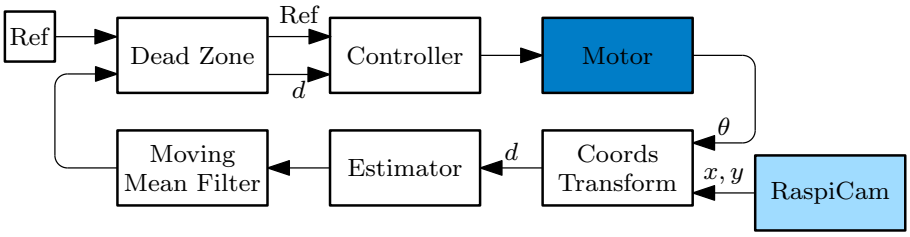


Figure 7.1: Control loop for the flat part balancing.

- **Coords Transform** – The ball's Cartesian coordinates x and y are firstly transformed to a signed distance d using the equation (5.27).

- **Controller** – For controlling the ball's position on the beam, I employed a PID controller with the following constants

$$K_p = 0.0031\,, \qquad K_i = 0.0021\,, \qquad K_d = 0.0029\,. \tag{7.1}$$

- **Estimator** – As the Section 3.3 states, the sampling time of the RaspiCam is $T_c = 30\,\text{ms}$, while the system sampling time is $T_m = 5\,\text{ms}$. The difference between the two sampling times means the ball position is updated in every sixth sample of the simulated system. With this setting, the beam could not effectively control the ball's position, as the beam tilting was not sufficiently smooth.

  This issue was solved by introducing a linear estimator to the scheme 7.1. For its functionality, I approximated the ball's motion on the beam with a linear motion. The velocity of the ball is calculated using the current position of the ball d_c and the previous position d_{c-1} from the camera

$$v = \frac{d_c - d_{c-1}}{T_c}\,. \tag{7.2}$$

  The five unknown ball positions between every two camera samples (position d_0 is the only known) are then calculated using an equation

$$d_i = d_c + ivT_m, \qquad i = 0, \ldots, 5\,. \tag{7.3}$$

  The index i is determined by a counter, which always counts to six with the system's sampling time and then resets back to zero.

- **Moving Mean Filter** – Considering the current and previous values of the ball distance d_c and d_{c-1}, I discovered during the control design that some differences between the two values turned out to be too distinct. In some cases, velocity v calculated from such a significant change in the ball position was overly large. The velocity used for the following five estimated distances did not then resemble the real velocity of the ball as the ball could be, in reality, slowed down by the beam. The inappropriately obtained velocity thus resulted in the wrong estimation of the ball position.

On that account, I added a moving mean to the control loop, which limits the vast distances of the ball estimated with the algorithm. This way, it uses the trend of the ball position and prevents the ball from moving too fast on the beam.

The moving mean is calculated from the last six distances corresponding to the length of one sample from the camera. The choice of the moving mean proved to be an efficient solution for an appropriate estimation of the ball position and was achieved by using delay blocks in Simulink. The equation of the moving mean is represented by

$$u_{i+1} = \frac{1}{6} \sum_{j=0}^{5} u_{i-j} \,. \tag{7.4}$$

The comparison of the position from the RaspiCam and the estimated position with the estimator and moving mean is shown in Figure 7.2.

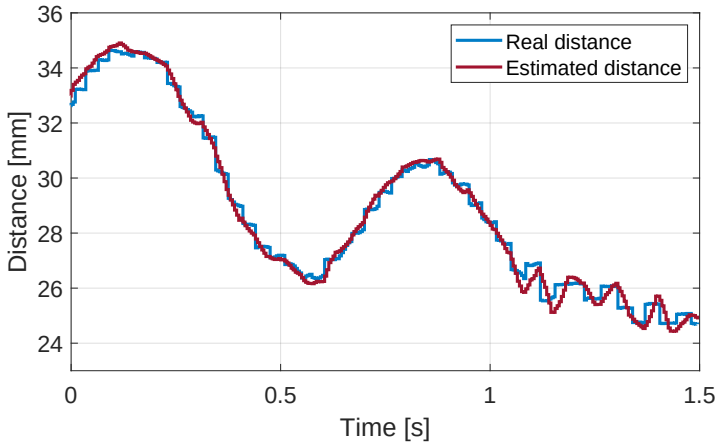


Figure 7.2: Comparison of the real ball position captured directly by the RaspiCam and the estimated position obtained using the estimator and the moving mean.

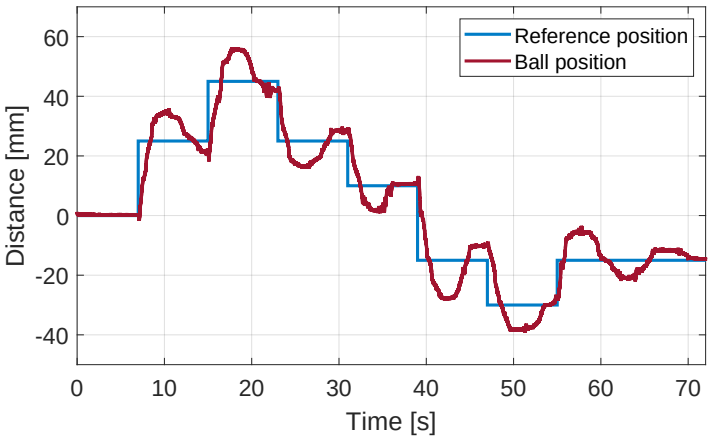


Figure 7.3: Balancing the ball in various positions on the flat part of the beam by following a reference position.

- **Dead Zone** – Although the ball remains steady on the beam and does not move, the distance d fluctuates $\pm 0.5\,\text{mm}$ around the reference position. The added dead zone prevents the motor from compensating for such small fluctuations as well as minor position differences if the ball does not exactly reach the reference position. The dead zone is thus given by

$$|d_{\text{ref}} - d| < 1.5\,\text{mm}\,. \tag{7.5}$$

7.3 Rounded Part Balancing

Balancing the ball on the beam's rounded part signifies that $\theta = \frac{\pi}{2}$ with the aim to control the system so that $\alpha = 0$, which is shown in Figure 5.4. The ball's coordinates x and y are transformed with (5.29) to control the angle α.

Barriers were added to the sides of the beam in the vertical position preventing the ball from falling. To control the α deviation, I designed a PID controller with a similar scheme architecture to the one used to balance the ball in the horizontal position.

However, the control was never efficient and fast enough to steadily hold the ball in the desired position. I wanted to ascertain thus that controlling the ball on the rounded part was feasible with this hardware.

At first, I measured the system's delay, which is the time it takes for the controller to react to the α deviation. To measure correctly the delay between capturing and processing the image from the RaspiCam to computing the regulator's control input, I switched the simulation sampling time back to the original value 30 ms and measured two significant times.

I used the motor in the positional mode with a reference position $d = 0\,\text{mm}$ and balanced the ball on the beam in its horizontal position. I then deflected the beam by hand and captured the time of the first angle change $\text{d}\theta$, which is the motor block output. To get the second time, I measured the time of the first control input from the controller, corresponding to the motor input. By subtracting these two times, I got the system's delay, which is 90 ms.

After that, I measured the time it took the ball to move from the tip of the beam's rounded part to a position where the controller could no longer control it. For that position, I chose an angle $\alpha \approx \frac{\pi}{3}$. This revolving motion was captured by a super slow-motion mode on a mobile phone camera. Using the OpenCV library in Python, I calculated the number of frames in the recorded video and divided it by the slow-motion mode frame rate of the mobile phone's camera, which is 960 Hz [12]. The resulting time was 145 ms. This means that the system has to balance the ball on the beam's rounded part with only one sample of the control input provided by the controller.

This signifies that the system's hardware is unsuitable for such control task. Considering the system's delay, one has to take into account the Raspberry camera's frame rate, the time it takes to process the image, and the delay in the communication between the Raspberry and the Moteus controller and the motor. The most significant improvement would be to use a higher frame-rate camera and a faster image-processing algorithm.

7.4 Ball Projecting

The third control task is the projectile mode, where the ball is projected by tilting the beam. The ball performs a free-fall motion and lands back on the beam. For this task, I assume that the ball's motion can only be directly controlled while the ball is in contact with the beam. When the ball starts the free-fall motion, the motion of the ball is only affected by the gravity force.

The variable parameters of this task are three different positions of the beam. The first is the ball's initial position, and the second is the point where the ball is released from the beam and projected. The third parameter is the point where the ball is balanced after landing back on the beam. With these preconditions, I divided the projectile mode into five sequential motions.

The ball can be projected from any position on the beam; for the following description of the projectile mode, I assume that the ball is projected from the left half of the world plane, which implies a negative distance d. The five motions of the projectile mode are as follows:

- **Release point approach** – The entire mode starts with the ball moving from the initial position toward the release point. This maneuver is executed by the same control algorithm for balancing the ball on the beam's flat part.

- **Ball projecting** – Once the ball approaches the desired release point, the beam tilts slightly in the direction of the forthcoming projectile motion so that $\mathrm{d}\theta$ is negative. Such a tilt should prevent the ball from flying in the wrong direction and landing away from the beam because of the centrifugal force applied to the ball. After that, the motor rapidly tilts the beam, causing the beam to project the ball, which starts the free-fall motion.

- **Free-fall** – After projecting the ball, the beam returns to the horizontal position so that $\theta = 0$, and the ball follows a parabolic trajectory.

- **Fall dampening** – When the ball approaches the beam, it tilts in the direction of the ball's fall to dampen its impact. The function that calculates the distance between the ball and the beam was introduced in Section 5.6 and is implemented in `PLS_dist.m`. The fall dampening is controlled using the equation (5.34).

- **Ball balancing** – After the ball lands back on the beam, the ball is balanced in the desired position in the same way as in the balancing mode.

The beam tilt, which projects the ball and the beam's subsequent return to the horizontal position, is set directly with a reference position waveform controlling the beam's position. The waveform is created as a fractional function composed of several linear functions corresponding to ramp functions.

The waveform is designed and can be adjusted by changing parameter k in each linear function's equation $y = kx + q$ and the duration of every ramp. The script for generating such a waveform can be found in `create_FF_wav.m` and is depicted in Figure 7.5.

The projectile task is implemented in Simulink. The control scheme consists of the motor block, camera block, and the particular motions of the projectile mode described above. The elements of the projectile mode are activated successively by three switches depending on the ball position. The logic behind the control of the mode is clarified with a diagram in Figure 7.4. In the diagram, `Leave pos` and `End pos` is the mode's second and third parameters of the mode. Block `Switch2FF` controls switching between the *Release point approach* motion and the *Ball projecting* motion, as well as between the *Free-fall*, *Fall dampening* and *Ball balancing* motion. For the diagram's simplicity, signals from the block `Switch2FF` triggering the switches are not shown in the figure.

Completing the task within the prescribed period proved unattainable in light of the time limitation, which resulted in being unable to test and debug the implemented projectile mode introduced and described above.

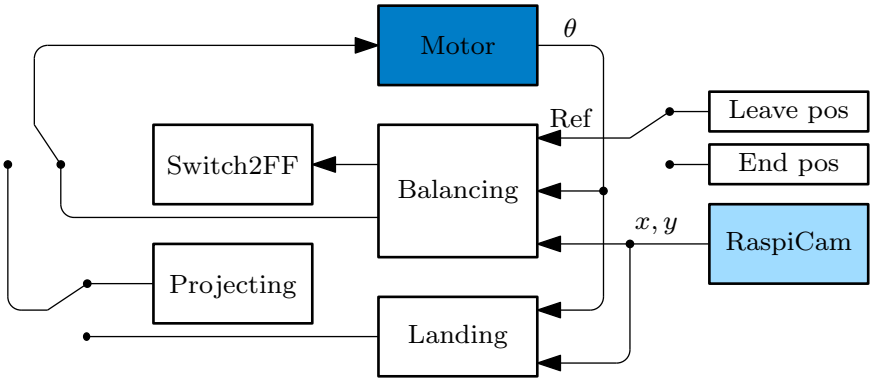


Figure 7.4: Diagram of the projectile mode control.

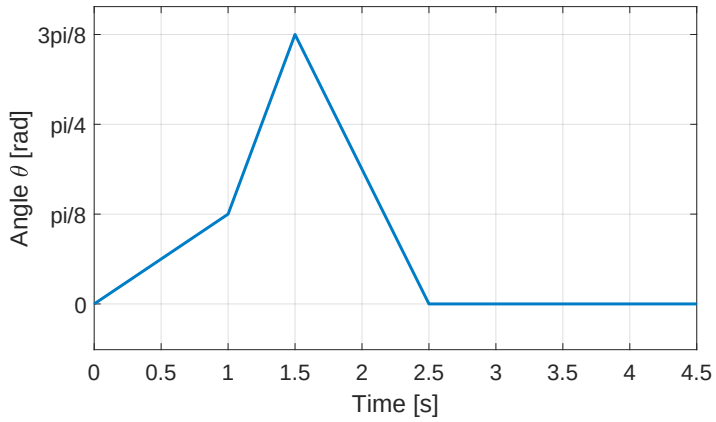


Figure 7.5: Reference beam's position used to project the ball and level the beam in its horizontal position.

Chapter 8

Conclusion and Outlook

8.1 Summary

The objective of this bachelor's thesis was to introduce a Flying Ball and Beam system, which extends the established ball and beam control system by incorporating two new modes, resulting in a hybrid system. The hybrid system comprises three modes: balancing, revolving, and projectile, each representing distinct motions of the ball and beam. The modes were described with mathematical models derived in Chapter 5. In terms of hardware, the Flying Ball and Beam makes use of a laboratory model called Flying Ball in Hoop, which was adjusted and modified. The hardware and software setup of the model is detailed in Chapter 3.

Subsequently, two balancing tasks were conducted on the real model. The first task involved controlling a ball at any desired position on the beam balanced in its horizontal position, as analyzed in Section 7.2. The second task aimed to balance the ball on the beam in its vertical position, which proved unsuccessful due to limitations in the real model's hardware.

The third task focused on demonstrating ball projection on the beam. The implementation of the task is described in Section 7.4. Given the limited time available, testing and verifying the last task on the real model was not feasible.

Documentation for the setup and usage of the Flying Ball and Beam system is available in the system's GitHub repository, which also contains the source codes used in this work.

Videos documenting the ball balancing on the horizontal beam position are provided. The first video, named `pos_dev.mp4`, showcases balancing the ball at a desired position and demonstrates the system's response to disturbances in the form of ball position deviations. The second video, named `ref_flw.mp4`, illustrates balancing at various positions along the beam by following a reference position waveform. This video corresponds to the ball balancing on the beam depicted in Figure 7.3.

8.2 System Use

While working on the Flying Ball and Beam system, I used a PC in the office labeled KN:E-8 in the Department of Control Engineering. As mentioned in Chapter 3, the Raspberry Pi and the computer communicate thanks to SSH communication.

I have discovered that a connection to the same network and setting the SSH communication for such a system can be made with two methods. The first and easiest option is connecting the Raspberry Pi and the user's computer with two Ethernet cables to the same router.

The second option is to connect the Raspberry Pi wirelessly to a Wi-Fi network. In this case, the user's computer must also be connected wirelessly to the network. The advantage of this option is that the computer does not have to be connected physically to the system. On the other hand, only some PCs have a Wi-Fi card enabling a wireless connection. One can use either a Wi-Fi USB adapter or a notebook instead of a PC. I used the first option for my bachelor's thesis and connected both devices to the same router.

To use the Flying Ball and Beam system with another computer, one has to take into consideration choosing the suitable connection option. Installing and setting all required software on the computer in the office to control the system turned out to be heavily time-consuming. Following the system's documentation will facilitate the installation and help set the system up on another computer.

8.3 Future work

Future work on the Flying Ball and Beam could focus on designing control for the three modes of the implemented hybrid system. The derived models could be used to design, e.g., LQR stabilizing controllers and utilize them on the real model. Although the model's camera proved to be a slight drawback when controlling the ball in the beam's vertical position, executing the revolving mode and then balancing the ball on the beam in the horizontal position should be feasible. The hybrid system could be first tested using the visualization and then performed on the real model to demonstrate switching between the modes.

Appendix A
Additional information

A.1 Source Codes

This section lists the software packages' source codes used in this thesis and their source links. The first link is the GitHub repository of the Flying Ball in Hoop system, which served as a master model for this thesis. The Flying Ball and Beam repository contains all files I created for the system identification, hybrid system implementation, and system control. Considering the system's hardware, links to source codes for Matlab System Objects, namely the RaspiBallPos and the Moteus API, are listed below, too. The last source code link is the ERT Linux used for the Simulink–Raspberry communication.

The repositories also include installation instructions on installing and running these software packages.

- **Flying Ball in Hoop** – `https://github.com/aa4cc/flying-ball-in-hoop`
- **Flying Ball and Beam** – `https://gitlab.fel.cvut.cz/lehkysim/flying-ball-and-beam`
- **ERT Linux** – `https://github.com/aa4cc/ert_linux`
- **Moteus API** – `https://github.com/doloi456/moteusapi_ert`
- **RaspiBallPos** – `https://github.com/aa4cc/raspi-ballpos`

A.2 Documentation

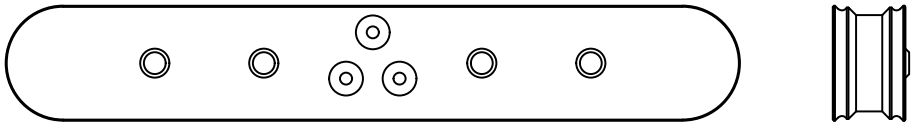
The system documentation describes installing the software packages and other software libraries required for the system's functionality. The documentation also includes a description of how to run and control the system.

One can find the documentation[1] in the system's GitHub repository.

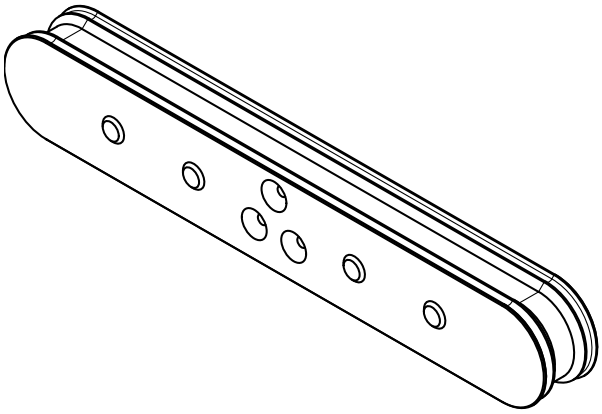
[1]The documentation of the Flying Ball and Beam system is available at `https://gitlab.fel.cvut.cz/lehkysim/flying-ball-and-beam/-/tree/main/docs`.

A.3 Beam Illustration

An illustration of the 3D beam model created in Fusion 360 can be seen from three different view angles in Figure A.1.



(a) : Front and left side view.



(b) : Top right view.

Figure A.1: Illustration of the beam model in Fusion 360 with the four calibration points and three holes used to mount the beam on the motor.

Appendix B

Bibliography

[1] Peter E. Wellstead et al. "The Ball and Beam Control Experiment". In: *International Journal of Electrical Engineering & Education* 15.1 (1978), pp. 21–39. DOI: 10.1177/002072097801500107.

[2] Peter E. Wellstead. *Introduction to physical system modelling*. Academic Press, 1979. ISBN: 0127443800.

[3] Peter E. Wellstead. "The Ball and Hoop System". In: *Automatica* 19.4 (1983), pp. 401–406. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(83)90054-7.

[4] Martin Gurtner and Jiří Zemánek. "Ball in double hoop: demonstration model for numerical optimal control". In: *IFAC-PapersOnLine* 50 (July 2017). ISSN: 2379-2384. DOI: 10.1016/j.ifacol.2017.08.429.

[5] Daniel Ortíz Morales, Pedro X. La Hera, and Shafiq-Ur Rehman. "Generating periodic motions for the butterfly robot". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Nov. 2013). ISSN: 2153-0866. DOI: 10.1109/IROS.2013.6696712.

[6] Hongrui Wang et al. "Output regulation of the ball and plate system with a nonlinear velocity observer". In: *2008 7th World Congress on Intelligent Control and Automation*. 2008, pp. 2164–2169. DOI: 10.1109/WCICA.2008.4593259.

[7] Antonio Criminisi, Iain Reid, and Andrew Zisserman. "A plane measuring device". In: *Image and Vision Computing* 17.8 (Aug. 1999), pp. 625–634. ISSN: 0262-8856. DOI: 10.1016/S0262-8856(98)00183-8.

[8] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, July 2017, p. 349. ISBN: 9781107016880.

[9] Carlos G. Bolívar-Vincenty and Gerson Beauchamp-Báez. "Modelling the Ball-and-Beam System From Newtonian Mechanics and from Lagrange Methods". In: *Twelfth LACCEI Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2014)*. July 2014.

[10] The MathWorks Inc. *Stateflow*. 2023. URL: https://www.mathworks.com/products/stateflow.html.

[11] rrlinus. *Minimum distance from a point to the line segment using Vectors*. Geeks for Geeks. Version: 2022-12-14. URL: https://www.geeksforgeeks.org/minimum-distance-from-a-point-to-the-line-segment-using-vectors.

[12] phev8. *How to know total number of Frame in a file with cv2 in python*. Stack Overflow. Version: 2016-05-24. URL: https://stackoverflow.com/questions/25359288/how-to-know-total-number-of-frame-in-a-file-with-cv2-in-python.