Semester Project Report: Ball-beam PID Control

CS 513 Winter 2024

Sebastian E. Zapata
dept. of Electrical and Computer Engineering
Brigham Young University
Provo, Utah, USA
szapata8@byu.edu

***Abstract*—Proportional-Integral-Derivative (PID) Control is a flexible and widely used control strategy for single-input- single-output linear systems. This work goes over the theory behind this control strategy and also the details of implementation for the control of a ball-beam system. A general treatment of hardware and software integration concepts, employed tools, and unexpected shortcomings in the development process are also included.**

***Index Terms*—PID control, ball-beam system**

I. Introduction

A big focus of this course has been to understand how to deal with instabilities and perturbations in a given system, by applying control techniques that result in being able to stabilize the behavior of it towards a specific target. Some of those techniques were LQR and LQG controllers, which implement robust control. However, these techniques require more analysis, mathematical tools, and information about the system dynamics. A specific control strategy not covered in this course is Proportional-Integral-Derivative control (PID) which though not as sophisticated, provides more flexibility and requires minimal knowledge about the system dynamics to provide acceptable performance.

The objective of this project is to explore in more detail both the theory and implementation of the PID control algorithm. This will be done by both showing the general principles behind PID and then implementing them in hardware with a custom-made ball-beam system. A section will be dedicated to describing the theory of the PID algorithm with variations of the tuning process, followed by a description of the demo built to test the algorithm. Afterwards, the results of the tuning process will be described.

A significant portion of the work for this project went into the building, assembly, coding, and testing of the demo, which cannot be included in the report, however a reference to the code and pertinent files for fabrication, assembly, and operation can be found in the "Supplemental Resources" section at the end of this work.

II. Background: PID Control

The PID (Proportional-Integral-Derivative) control algorithm relies on being able to measure the state of a target parameter in the system and receiving a target/desired value for such parameter. This allows the algorithm to calculate the mismatch between them (error) and adjust the input to the system so as to modify the state of the given parameter and bring it closer to the target value. Fig. 1 shows a block diagram of this strategy, and as it can be noted, the error is passed through a proportional block, an integral block, and a derivative block, each of which contributes to the input signal that will be passed into the "process" block that encapsulates the behavior of the system.

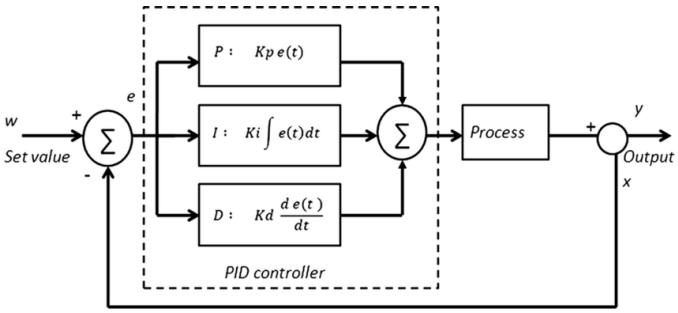

Fig. 1. PID block diagram, highlighting the proportional, integral, and derivative blocks that take the error signal and provide an input signal to the system

Each of the controller blocks has a specific functionality in this algorithm, and they are specified below:

- Proportional (P): a constant K_p multiplies the error signal, which means that the bigger the error signal, the bigger the proportional response to it. Controlling a system with just a K_p gain is called P control.
- Integral (I): by using P control and depending on the dynamics of the system and the kind of input provided, error in the state can be perpetuated over time in what is called "steady-state error". To solve this problem, the value of the error can be tracked over time, integrated, and multiplied by K_i, so the larger the "accumulated" error, the larger the compensation for it.
- Derivative (D): depending on whether the error signal changes slowly or rapidly, ideally, we would want out controller to react correspondingly as well, so in this block we calculate the derivative of the error signal by tracking past values and observing how fast they are changing. The derivative of the error signal is then

multiplied by K_d . This allows us to account for the speed at which the error signal changes and deal with oscillations introduced by variability introduced from the K_p or K_i gains.

In the sense described above, each of the constants or "gains" (K_p, K_i, and K_d) become independent knobs we can use to modify the response of our controller to the errors in the system. Since every system behaves differently, finding the right combination of "gains" that will ensure good performance given a desired value for the states is a process that can take significant amount of work. This process is called "tuning" and even though there are many guidelines and algorithms that have been developed to efficiently perform it (one of them treated in more detail in the section below), it is by nature a trial-and-error process until a "good enough" combination is found. In this sense, there's no perfect tuning, only better or worse combinations of gains; this decision is therefore under the prerogative of the control engineer in charge of the controller design.

It is worth noting that this diagram very abstractly denotes the "process" block as a simple black box that takes the input signal and provides an output signal but does not specify any kind of structure or requirement. This is one of the big advantages of PID with respect to other control techniques: prior knowledge of the system dynamics is optional. Knowledge of the dynamics of the system enriches the control performance as the behavior becomes even more predictable, however, this is not strictly needed as a good tuning process will still offer a good response to a desired behavior in the most general case. It is, however, plausible that certain systems will still perform poorly despite a decent amount of tuning work being performed, so by adding dynamics to the algorithm or even considering other control techniques, more robustness can be introduced.

Finally, the diagram in Fig. 1 shows each of the operation blocks as if the error signal was in continuous time, however, since this algorithm will be implemented in software, the integral and derivative blocks will instead be changed to their corresponding discrete time counterparts and evaluated at every iteration of the firmware of the micro-controller.

III. Tuning Methods

The tuning methodology used for this project consisted of setting all the gains to 0 and sequentially adjusting the values of each until a good-enough combination was found. The proportional gain was first set to attain a fast enough response that did not introduce oscillations. Following this, the integral gain was set to a small value to account for the steady state error present after finding an acceptable proportional gain. And finally, I tried some relatively small values for the derivative gain, however they did not seem to help much and after having a PI controller (proportional and integral gains only), the angle control was good enough in terms of response time and steady state step response. With this realization, the derivative term was left in 0.

Though this sequential methodology is very simple, it's not the accepted practice in industry as many algorithms have been developed that have a more formal procedure. The most popular methods being the Ziegler-Nichols, the Cohen Coon, and the Internal Model Control. Part of my exploration of the topic included learning the basics of the Ziegler-Nichols method, which seems to be the most prevalent in industry applications. An exhaustive treatment of this technique is not included for simplicity and because I did not use it in my project, but it was something worth noting, nonetheless.

IV. Demo Description

The tool built to implement the algorithm is a combination of a 3D printed ball-beam system with an actuator, sensors, a micro-controller and supporting power system integrated with a web app for manual control, feedback control manipulation, and live visualization of the sensor readings. Fig. 2 shows the project organization and the integration of hardware and software required to perform tests of the PID algorithm. It also shows the communication protocol (or controlling signal scheme) used for the interaction of all the connected blocks.

The hardware realization of the system is shown in Fig. 3, and the main software tools employed in the development process of the GUI and communication of the individual components of the system are shown in Fig.4. An exhaustive list of components used in the physical system go as follow:

- 3D Printed parts for component mounting
- NEMA 17 stepper motor
- DRV8834 stepper motor driver
- AS5600 magnetic encoder
- 11.1V LiPo battery
- 30W buck converter
- Breadboard
- ESP32 micro-controller
- GP2Y0E03 infrared distance sensor
- VL53L0X time of flight laser distance sensor

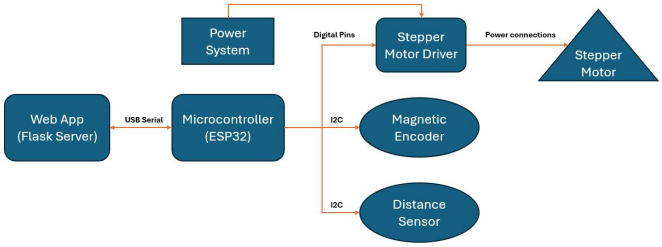

Fig. 2. Block diagram describing the integration of hardware and software of the project

It should be mentioned that a significant amount of work went into the characterization and calibration of the sensors in the system. Both a running average was performed on the readings to reduce noisy measurements as well as a custom offset calculation from sampling at different intervals and averaging the error at specific ranges. For the magnetic encoder, a good understanding of the mechanics of the process for angle

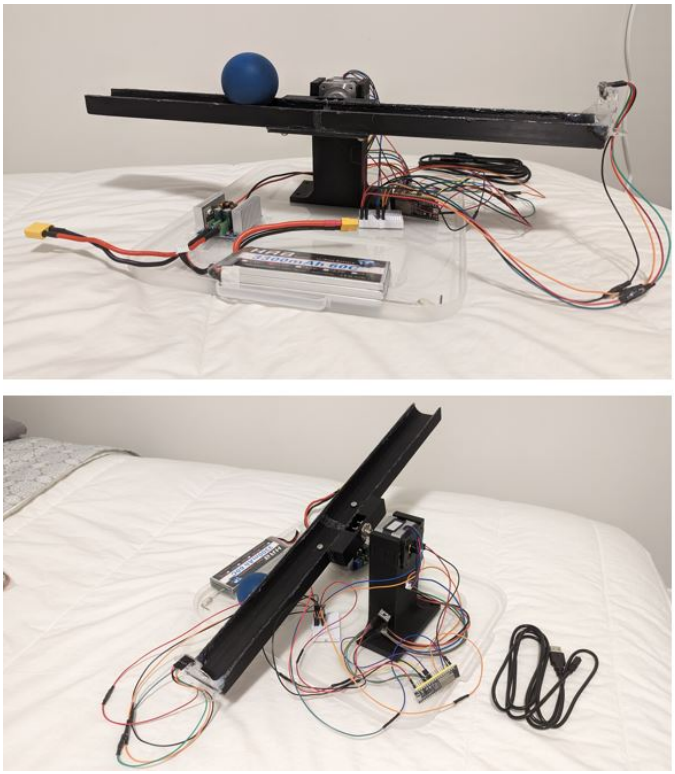

Fig. 3. Hardware realization of the system, showing the base holding the stepper motor which has a platform attached to the shaft upon which a ball is placed

Fig. 4. Main software tools employed for communication and control of the hardware

recognition was needed to effectively include the sensor as depending on the orientation of the magnet on the motor shaft, different initial angles are read. It also should be noted that due to unexpected behavior at initialization, different angles are read every time, so a "zero out" functionality was built in so that the operator can define the zero position. After this, the sensor performs as expected indicating angles relative to the user-defined zero position.

For the distance sensor, different positions and orientations had to be tested (not to mention different kinds of sensors had to be tested) to make sure an optimal and least error-prone measurement could be guaranteed within the range of interest for the positioning of the ball. Upon completion of the project, an extra sensor was mounted on the other end of the beam since a single sensor would be very noisy and error prone past the nearest half to the beam. Thus, each sensor was assigned a portion of the beam to provide more accurate measurements and the position of the ball was measured by considering both measurements and picking the sensor that would provide readings within the assigned range.

V. Performance Assesment

After assembly, manual testing, and calibrations were performed, the algorithm was loaded on the microcontroller to perform control of the beam angle and ball position, with certain target values coming from the web app over serial communication as well as the ability to provide different values for the PID gains. A qualitative assessment was initially conducted to make sure that both the angle and ball position followed the desired value and then a more quantitative assessment was conducted to measure the response accuracy and settle time for the system to reach the desired value.

Upon carrying out tests, the system seemed to behave properly in terms of angle control, showing fast response to step inputs with little to no overshoot, even with the beam mounted on the stepper motor shaft. However, when trying to implement ball position control, the distance sensors proved to be faulty since the i2c communication with one of the sensors would periodically fail and an error state would be triggered for a few seconds until communication would be reestablished. This prevented any kind of reliable position tracking. It should be noted that the errors most likely came from an issue internal to the sensor circuitry since even when the system would be kept still without any kind of motion on the wires connecting the sensor to the microcontroller, the communication would still periodically be interrupted. This ruled out the option that a poor wiring scheme was used, though it should be acknowledged that a breadboard was used as an intermediate routing point since there was not enough time during the design and assembly of this project to build a proper and more professional PCB to mount all the electronics of the system. This could be a potential improvement for the future to increase reliable performance, not just for the distance sensors, but for all the electronics.

VI. Conclusions

PID is an excellent control technique for simple and even complex systems and provides significant flexibility with respect to other control techniques such as LQR or LQG. However, it suffers from a lack of accurate and/or prompt response depending on the quality of the tuning performed, and a lack of guaranteed performance as the noise/disturbances on the sensors are not accounted for in the mathematical structure of the controller. For applications where robust guarantees of performance are not required, PID is an excellent way to offer control capabilities with low modelling and computing requirements.

However, implementation of the PID algorithm in software combined with the integration of accompanying hardware is not a trivial problem and several issues arise from variability in sensor operation, system dynamics, and other physical limitations not considered in a theoretical framework. Valuable

insights and experience were gained from this hardware-software integration process, which was the motivation for this project.

VII. Supplemental Resources - GitHub Repository

- Link: https://github.com/szapata8/Ball_beam_control
- Contents:
  - Flask web app project
  - Micro-controller firmware
  - Individual drivers for modules and sensors
  - STL files for ball-beam system
  - Pertinent data sheets
  - Project deliverables

References

[1] C. Peterson, R. Beard, and T. McLain, "Introduction to Feedback Control", 2019.