Quinn Maloney
200431628
Dr. Karim Naqvi
April 11, 2025

Ball Height Controller

Design

Overview

The ball height controller is a system that uses a fan and a valve to change the airflow inside a tube with a ping pong ball. It is connected to both a 5V and 12V power supply (12V for fan, 5V for everything else). It is controlled with the MCBSTM32EXL board that uses a STM32F103ZG processor. Depending on the airflow inside the tube, the ping pong ball will rise or sink inside the tube. The goal is to design an algorithm to balance the ball in a specific location inside the tube. The ball height controller has two main controllable components, an infra-red sensor and a servo motor. The infra-red sensor is responsible for reading the ping pong ball's location by taking a proximity reading. The servo controls the valve at the bottom of the tube to control the airflow by opening and closing it. The last used element in this project is the USART port on the board. This is used for communication from the user to the board via terminal input/output. The system will use a PID algorithm to balance the ping pong ball without user input.

Implementation

The infra-red sensor is connected to the board on pin PC0 and uses channel 10 from ADC1. This then reads the infra-red sensor value so it can be used by various functions in the program. The servo motor is driven by a pulse-width modulation signal on pin PB7 with Timer4. Servos are generally run at around 50Hz with around a 5% to 10% duty cycle. There were some calculations involved in determining the proper values for the timer. The prescaler is set to 71 making the clock frequency calculation 72MHz / (71 + 1) = 1Mhz. After bringing the frequency down with the prescaler the auto reload register can be set to change the period of the timer. To find this value I took the period I want (20ms) and divided it by the current period 20ms / 0.001ms = 20000. This then becomes the auto reload register value resulting in the end frequency being 50hz. The period length on the servo motor specifications says 600us to 2400us. This translates directly to 600 to 2400 as values inside the CCR2 register to control the duty cycle. The COM port of the board is attached to USART2 which uses pin PA2 for transmission and PA3 for receiving. It uses a baud rate of 9600 for its communications.

The program is structured into three primary sections: CLI_Control, Ball_Control, and PID_Control. CLI_Control is responsible for taking input from the user as well as providing output to the user. Ball_Control has functions directly relating to the servo and infra-red sensor to interact with the ping pong ball. PID_Control is the PID algorithm implementation which takes input from the Ball_Control section and provides output to it as well. The CLI_Control uses a USART receive interrupt to get the inputted character from the user whenever a key is pressed. This is then processed accordingly and can possibly call another function if needed. The cli interface is updated via a SysTick interrupt that fires about twice a second updating the display with any new duty cycle or infra-red readings.

The user can initiate or stop self-balancing through a command either ‘start’ or ‘stop’ depending on the desired action. This will cause the ball height controller to begin self balancing where it passes in the current infra-red reading and compares it to the goal to output the next change to balance the ball. In PID, P stands for proportional where it is compared directly to the current error. I stands for integral where it takes the sum of previous errors into account in its calculation. Lastly, D stands for derivative where it compares the previous error to the current error. These three values are added together to provide output of the next duty cycle to use based on the current infra-red reading.

Testing

Tube Calibration

Before being able to implement any algorithm to balance the ball, a graph of the infra-red readings at each distance was required to map out the values at each distance before choosing a suitable point to balance at. The measurements are not very uniform at all and very difficult to work with since it is not a linear or exponential graph but instead it fluctuates up and down. Through this calibration, the best location to balance at is where the infra-red reading is at 1300 or between 330 and 340 mm from the top of the tube. The calibration was done by taping the ping pong ball to the end of a tape measure and measuring the distance. The bottom of the tube was recorded as 601mm from the top.

PID Tuning

The method I used to approach this was to set I and D to zero and to find a P value that oscillates up and down at the setpoint. After that was successful, I reduced P to half its value and started incrementing I until it reached its setpoint at a reasonable time. Last, I then incremented D until it got to a reasonable point and was not over shooting the setpoint. The final values ended up being P = 2.0, I = 2.5, and D= 3.0. This gave the best results. I had initially said in my pitch that I would aim for around one centimeter of error. This proved to be extremely difficult due to the noise generated by the infra-red sensor as well as the inconsistencies in the infra-red output at different distances. This means that there was a very small workable area for the PID algorithm to work at all. In the end it is about an eight centimeter range or four centimeter error. This is much larger than I had hoped but I am still satisfied with the result.

Component Testing

To test the components I did not create a formal testing suite. I did, however, conduct testing through trial and error and debugging. One of the major tests was to ensure the duty cycle of the servo was never outside of its bounds. There are constants that set the maximum and minimum duty cycles so that they will not be crossed. USART had no specifics in testing other than what is typed is what is entered. The ADC was tested to ensure it was always reading and reading correctly. This was a hard one to quantify since the readings can vary quite widely even from the same position but through the calibration it did not have and problems reading and returning the value.

Usage

The system can be used in two modes: Manual or self balancing. The system initializes into manual mode where the duty cycle can be changed with the '[' and ']' keys (lower and raise). The duty cycle moves by ten per input and is about the right amount to control it manually. To begin self balancing mode, the user can enter 'start' to begin self balancing where the PID algorithm will begin changing the duty cycle without any user input and try to balance the ball at the 1300 point. To return to manual mode the user can enter 'stop' and it will revert back to manual mode. The user can also enter 'help' for a list of commands at any time or use -v to display the git commit version. Keil uvision 5 is very inconsistent at least on my home computer about executing the pre-build steps which is where this command is configured. If it is unsuccessful in your environment the git command can be run in the git bash separately and generate the new hash. This command is documented on the homepage of the doxygen documentation if it is required.