California State University, Northridge

Department of Electrical & Computer Engineering

ECE 595R - Robotics & Embedded Systems

Fall 2023

Final Project

PID Control of TI-RSLK MAX

December 18, 2023

Cristian Robles & David Torres

Shahnam Mirzaei, Ph.D.

Table of Contents

Links:

https://github.com/csun-ece-robotics/ece595rl-fall-2023-final-project-group-7
https://drive.google.com/drive/folders/1DPvFXepZspW4EGPUMr3B1krpwpLK1wPy

Introduction:

In the ECE595R course, we were tasked with either selecting a final project from a list or coming up with our own idea. Our group opted to create a PID controller that would take specified distance targets and drive the TI-RSLK max robot to meet these goals. This idea was sparked by the ongoing developments of the CleanBot3000 senior design project, which is currently in the prototype phase. This semester, the CleanBot team has implemented the SLAM algorithm using ROS2 with the RPLiDAR A1, a step that allowed for the creation of environmental maps. The upcoming phase involves traversing the mapped area for effective sanitization. The intended method to accomplish this involves a path planning algorithm that will provide the robot with distance and angle targets, which the CleanBot will follow. Our group is undertaking this project to delve into the intricacies of PID control, including its development and tuning, to prepare for the coming tasks of the CleanBot project.

Theory:

1. PID Control

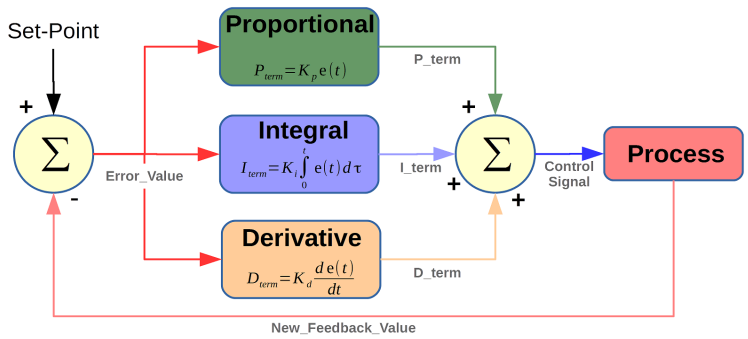

Figure 1 - PID controller block diagram

As seen in Figure 1, a PID controller is a control loop mechanism that uses continuous feedback to calculate an error value as the difference between a setpoint and a measured process variable. The controller then applies a correction based on proportional, integral, and derivative terms, denoted as P, I, and D. The proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by K_P, which is the proportional gain constant. The integral term handles the accumulation of past error values and looks to eliminate the residual steady-state error that occurs with a proportional only controller. It accumulates the error over time and is then scaled by the integral gain K_I. The derivative term is a prediction of future error, based on the system's rate of change. It is calculated by

determining the slope of the error over time and multiplying this rate of change by the derivative gain K_D.

By fine tuning the weights of the three components, the PID controller can refine its output for precise control, surpassing the capabilities of many other controller types. As illustrated in Figure 2 different component weights create various PID responses.

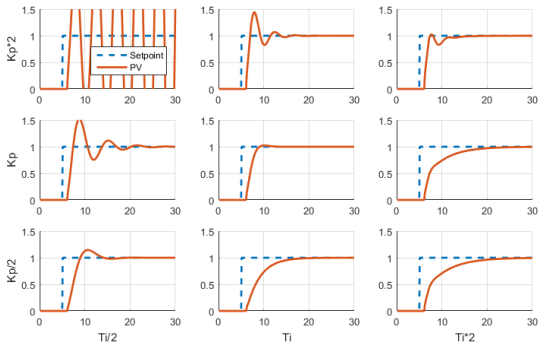

Figure 2 - PID controller responses

2. Tuning a PID Controller

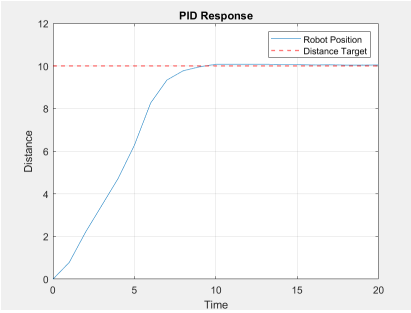

Figure 3 - Tuned PID controller

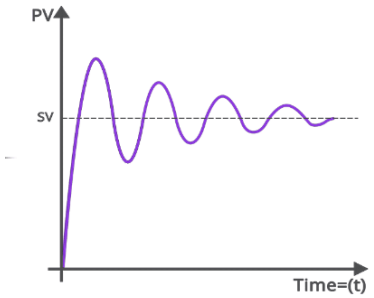

Figure 4 - Improperly tuned PID controller

Figure 3 illustrates a well-tuned PID controller, where the red dotted line indicates the target position and the blue line marks the robot's actual movement. This graph shows the controller smoothly ramping up and then stopping precisely at the target. Contrastingly, Figure 4 depicts an improperly tuned controller, evident by the robot's overshoot of the target followed by oscillations as it attempts to stabilize at the target position. In the following sections, we will explore various tuning methods for PID controllers.

A. Heuristic Tuning

In control systems, heuristic tuning methods apply empirical techniques to adjust the parameters of a PID controller with the goal of reaching stability in a system. These methods rely on loosely set procedures and hands on adjustments rather than mathematical modeling to guide the tuning process. Within this method, the Ziegler-Nichols method emerges as a prominent heuristic approach. It provides a protocol for setting the PID controller to induce constant amplitude oscillations from which the critical gain and oscillation period are measured. These measurements are then utilized to calculate the controller's gains according to formulas. While the Ziegler-Nichols method is a classic heuristic technique, it is just one of many. Another thing to mention is that this

method is not always safe. For instance, in systems like industrial robotic arms, induced oscillations can cause injuries or damage to the machine.

B. Model Based Tuning

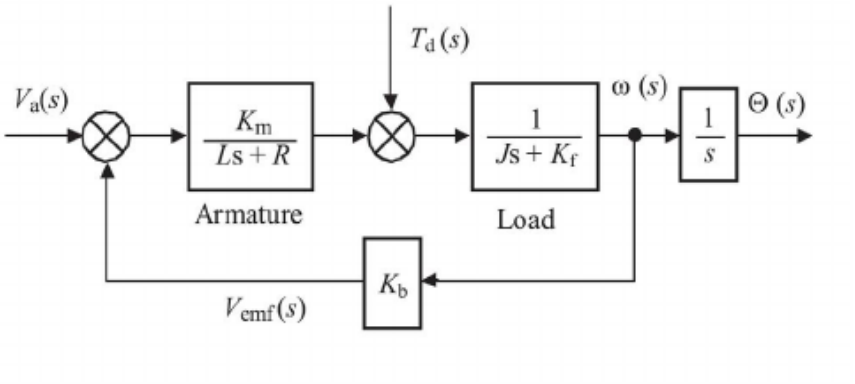

Figure 5 - Mathematical model of a DC motor

Model based tuning is another approach which utilizes a mathematical model of the system for parameter adjustment. This method typically starts with fundamental principles, like deriving kinematic equations, to develop a model that simulates the system. For more complex systems, tools such as system identification in MATLAB can create an accurate model by processing the system's response to inputs like a step signal. Tuning can be conducted through control system theory, employing techniques like pole placement to ensure a stable model. With a model heuristic tuning methods can also be applied in a simulated environment, this is especially useful in cases where hardware based tuning poses safety risks.

C. Automatic Tuning Method

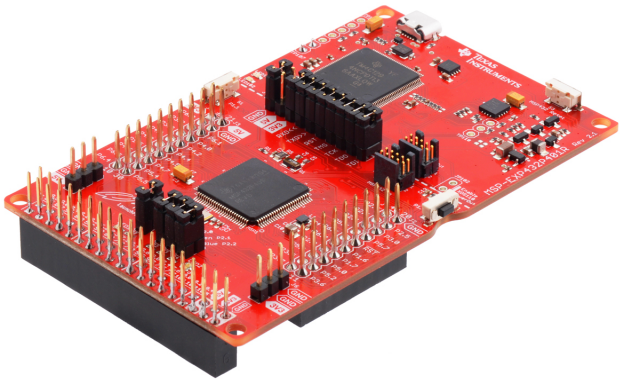

Figure 6 - MSP432 Microcontroller

Automatic software tuning, blending heuristic and model-based methods, is a valuable tool when there's a PID-controlled system with a test bed or control environment available. As seen with the MSP432 microcontroller we are using in Figure 6, you can deploy an autotuner block to your hardware enabling you to automatically tune the gains of the PID controller embedded in your system, effectively harnessing both the theoretical and practical aspects of system optimization.

3. **TI-RSLK Max**

   The TI-RSLK Max robot, shown in Figure 7, serves as an educational tool aimed at teaching embedded systems, control, and robotics. Equipped with a microcontroller, the robot is designed to facilitate learning through practical projects. It comes with a suite of sensors and motors, allowing for experimentation with real world robotics applications, including the implementation and testing of control systems like PID controllers. This hands-on platform is well suited for projects that require an understanding of mechanical, electronic, and software integration.

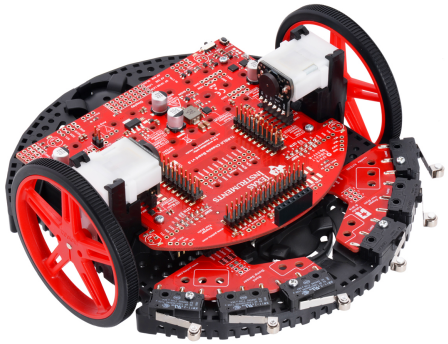

Figure 7 - TI-RSLK Max robot

Procedure:

1. Hardware

A. Motor and Encoder

For our PID controller to function, it requires a feedback mechanism, which is provided to us by the encoder within the gear and encoder assembly depicted in Figure 8. This encoder is important for acquiring precise wheel rotation measurements, enabling the robot to be controlled with accuracy. It's a magnetic quadrature shaft encoder, utilizing the Hall effect sensor to generate two output signals, channels A and B, as illustrated in Figure 9. These signals are phase shifted by 90 degrees relative to each other. By analyzing these channels, we can get both the velocity and direction of the robot's wheel rotation, which are critical data points for our PID controller to adjust the motor's input and achieve the desired motion control. The motor is a brushed 4.5 volt DC motor.

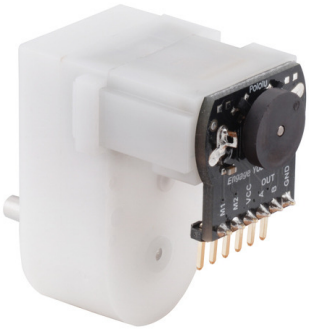

Figure 8 - Gear motor and encoder assembly

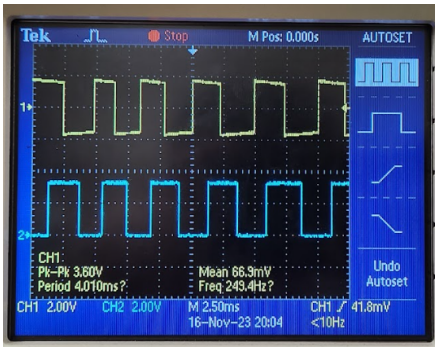

Figure 9 - Quadrature encoder signals

B. Wheels

To accurately program our PID controller with distance targets, it was necessary to measure the diameter of our robot's wheel, as shown in Figure 10, which is 2.75 inches. Knowing the diameter allows us to calculate the wheel's circumference, which is essential for converting the rotation counts from the encoder into linear travel distance. The circumference is determined by the formula $C = \pi * d$, where C is the circumference and d is the diameter of the wheel. Thus, each full rotation of the wheel will correspond to the robot moving forward by the length of this circumference. This measurement is crucial for the PID controller to accurately control the robot's movement to meet the set distance targets.

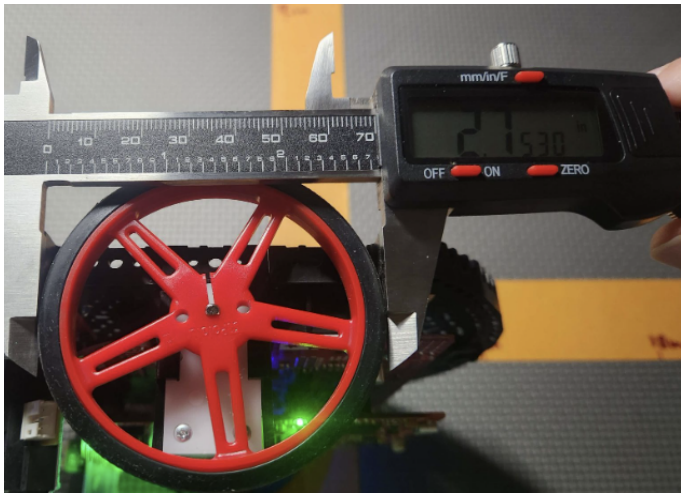

Figure 10 - Wheel diameter, 2.75 inches

C. Bumper Switches

To input distance targets into the robot, we used the bumper switch array on the robot, depicted in Figure 11. These switches enable the robot to move to predefined distances providing a simple and interactive way to control the robot's navigation and ensure it reaches the intended target points.

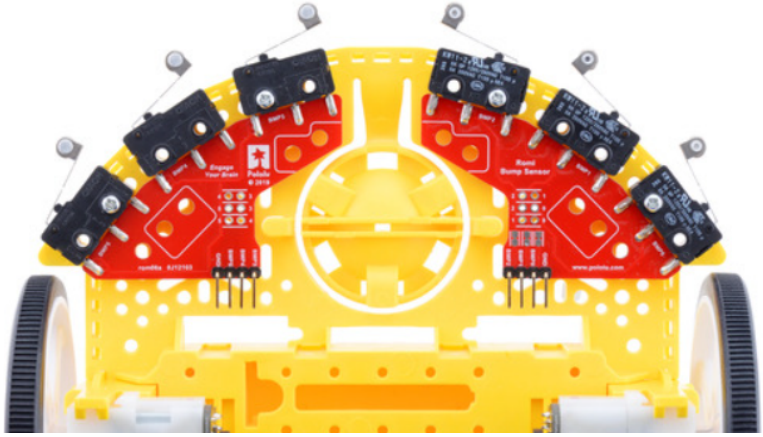

Figure 11 - Bumper switch assembly

2. Software

As depicted in Figure 12, the software controlling our robot follows a straightforward logic flow. When in idle mode, the robot remains stationary, with its current position set as the target. Upon activation by a bumper switch, the software's switch handler updates the target position. This triggers the PID control mechanism to engage, driving the robot toward the new target position. This process ensures the robot can respond promptly to inputs, adjusting its course as needed to reach the designated locations.

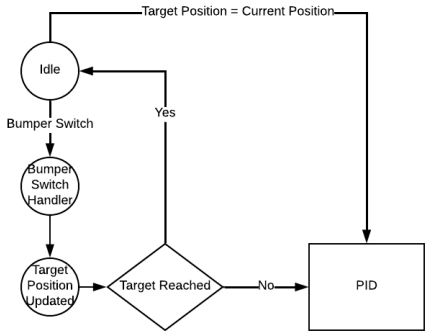

Figure 12 - Software Diagram

A. Encoder Interrupts

As illustrated in Figure 9, the encoder outputs channels A and B are utilized to generate interrupts, allowing the robot to monitor the encoder counts that occur. This count, along with the circumference data from Figure 10, enables the robot to accurately track its linear movement and is used to calculate the current position. To ensure precise control, each wheel has its own PID controller instance. This setup allows both wheels to independently reach their respective targets, avoiding the potential pitfalls of relying on a single encoder for movement guidance.

B. PID Function

As depicted in Figure 13, the PID function implements the three critical components: proportional, integral, and derivative, based on the error variable. The error, computed as the difference between the setpoint and the actual value, feeds into the proportional term. The integral term accumulates the error over time, and the derivative term calculates the rate of error change between the current and previous cycle. These components are then multiplied by their respective constants.

```
 * @brief PID Control Math function.
 *
 * @return error * Kp + integral * Ki + derivative * Kd
 */
float PID_Controller(PID_Controller_t *pid, float setpoint, float actual_value) {
    float error = setpoint - actual_value;
    pid->integral += error;
    float derivative = error - pid->prev_error;
    pid->prev_error = error;

    return (pid->Kp * error) + (pid->Ki * pid->integral) + (pid->Kd * derivative);
}
```

Figure 13 - PID function

C. Target Positions

For setting target positions in forward movement, the process is straightforward, we directly assign the target distance value. For instance, to move forward 10 inches, the target position is set to 10. However, for turning without a gyroscope, we performed simple calculations to determine the target position, as demonstrated in Figure 14. The blue line in the figure represents the necessary rotation distance for both wheels to achieve a 90 degree turn. To find this distance, we measured the space between the wheels to find the diameter of the robot's turning circle, calculated the circumference of that circle, and then divided it by four. This quarter circumference gives us a length of 4.31 inches, which is then programmed into the robot to execute a 90 degree turn.

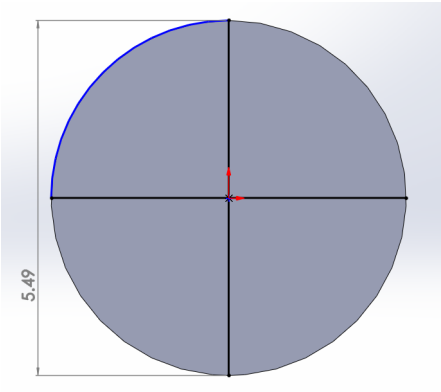

Figure 14 - Calculation of 90 degree turn, blue line is 4.31 inches

Results:

Figure 15 illustrates the PID response of our robot following the tuning process. The robot exhibits a stable response, starting from a position of zero and smoothly reaching a distance of 10 inches. This level of control was achieved through the application of heuristic tuning methods, where we began with zero values for both the integral and derivative gains. From this baseline, we employed a systematic approach to reach a general area of stability. After establishing this stable baseline, we fine-tuned the PID controller manually to achieve the precise output depicted in the graph. Demonstration of this can be found in the Google Drive link above.

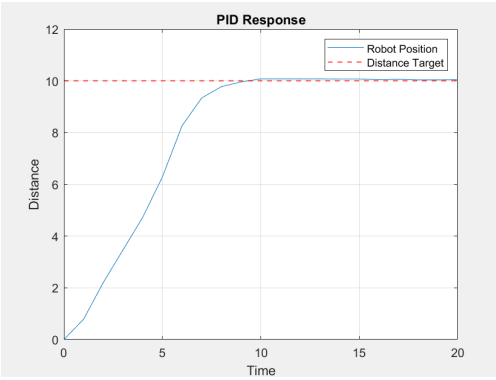

Figure 15 - PID response of our robot after tuning

Conclusion:

This project served as a practical lesson in the integration of software, hardware, and mechanical components within a robotic system. Through the development and tuning of a PID controller, it enhanced understanding of software algorithms and their real world applications. Working with the MSP432 microcontroller and the TI-RSLK Max robot provided insights into embedded system design and the complexities of electronic control. Mechanically, the project emphasized the importance of precise measurements and physical configurations in achieving accurate robot movement. In conclusion, this project showed how robotics blends different areas of engineering together, and how important it is to combine these fields to make complex systems work.

References:

- *Texas Instruments Robotics System Learning Kit User Guide*. Texas Instruments. https://www.ti.com/lit/ml/sekp166/sekp166.pdf
- *A heuristic approach to PID design and implementation*. MathWorks. https://www.mathworks.com/matlabcentral/fileexchange/91340-a-heuristic-approach-to-pid-design-and-implementation
- *Linear Model Identification*. Linear Model Identification - MATLAB & Simulink. https://www.mathworks.com/help/ident/linear-model-identification.html
- *Pololu - Gearmotor and encoder assembly for Romi/Ti-RSLK Max*. Pololu Robotics & Electronics. https://www.pololu.com/product/3675
- *Ziegler Nichols method*. Ziegler Nichols Method - an overview | ScienceDirect Topics. https://www.sciencedirect.com/topics/computer-science/ziegler-nichols-method
- *Closed-loop PID Autotuner*. MathWorks. https://www.mathworks.com/help/slcontrol/ug/when-to-use-pid-autotuning.html