


# Technical Notebook for Programming

Made by AndreiDani





# Content

1. Introduction to the Problem
  2. Proportional - Integral - Derivative Controllers
    - a. Integral Windup (limiting the integral)
    - b. Speeding up the calculations
  3. Object Stall Detection (to see if the robot's stuck)
  4. Speed Controllers. Accelerating Functions
  5. Gyro Forwards / Backwards (Moving)
  6. Turn Left / Right -> Arc Turns (Formulas)
  7. Calibrating the constants and parameters
  8. Links (Github, Wikipedia, Desmos, ...)
  9. Further reading (Wikipedia, Github, ...)
- 

# 1. Introduction to the Problem

We have four main functions that we use for moving:

- Move Forwards / Backwards
- Turn Left / Right

But how do we implement them?

## a. The first approach

For all functions we can apply constant speeds and check if the distance or the targeted angle has been reached. Then we stop the robot.

Problems: This approach is correct in theory, but when it is put in practice, the robot may deviate from its actual trajectory because of external factors. (ex. worn - out tires, wobbly table, slippery part of the map, etc.)

## b. A new concept

For all functions we can use a ***Proportional-Controller***. Its role is to fix the current error and make the robot return to its initial path. The error is set to ***(SETPOINT - THE READ VALUE)***. Then the proportional - ***(ERROR \* KP)*** is applied, along-side the speed given. This concept is used by a lot of FLL teams.

Problems: This approach fixes the previous problem, but what if the center of mass of the robot with the system is not in the center, but rather on the one side? This will make the robot lean to that side, and therefore go in that direction because this type of controller can't fix these kind of errors.

## c. Our approach to this problem

For all functions we use a **Speed Controller**, but for moving forwards / backwards we also use a **PID Controller** that runs in parallel (at the same time as the **Speed Controller**).

## 2. PID Controllers

**Proportional - Integral - Derivative Controllers** (or **Three Term Controllers**) are used when there is an input value (like from a gyroscopic sensor) and a value that it needs to reach.

The first step is getting the error, which is the difference between **A SETPOINT** and **THE INPUT VALUE**. The correction / output is then calculated by the **P**, **I** and **D terms**.

1. **The Proportional** is responsible for fixing the current error. Without this the response time is slow and will create offsets from the robot's actual path.

2. **The Integral** is responsible for fixing errors that may have accumulated over the past. For example if the robot is always going to the right side, because of the center of mass, the integral will become higher and higher, until they cancel out.

3. **The Derivative** is used for predicting the next error, using the difference between the current and last error (rate of exchange). This concept is used to minimize the next error.

4. **The Output** is the sum of these values times the constants.  
**Output = Proportional \* Kp + Integral \* Ki + Derivative \* Kd.**

## a. Integral Windup

**Integral Windup** is a modification done to the PID Controller. This makes the integral always be in the range of  $[x, -x]$ , where  $x$  is a constant value (parameter - integralLimit).

**Fix:** Let's say that the robot is moving forwards in a straight line, but then it collides with a mission and gets stuck. This will generate an error that can't be fixed and the integral will reach infinity as time passes. If this happens the program crashes. To make sure this doesn't happen we limit the integral, which also makes the output always be in bounds.

## b. Speeding up the calculations

The output is always rounded to the nearest integer, but the constants ***Kp, Ki, Kd*** are written as rational numbers and the error is an integer. This means that we can skip doing equations with real numbers and just approximate them to the nearest integer to calculate the output to any desired precision with a common scale for the constants.

***Kp, Ki, Kd*** are written as fractions of  $x / \text{scale}$ , where  $x$  is the constant multiplied by *the common scale* and *scale* is a parameter in the form of  $10^{\text{the number of decimals}}$ .

***Output = (Proportional \* Kp + Integral \* Ki + Derivative \* Kd)*** divided by ***Scale \* sign***. Because ***the terms P, I, D*** are always integers the final division will approximate the ***Output*** to the nearest integer without the need of *the extra decimals*.

**Example:** Let's say we have **Kp** = 2.5, **Ki** = 0.002 and **Kd** = 0.75. If we want to write them in the form of fractions they become **Kp** = 25 / 10, **Ki** = 2 / 1000 and **Kd** = 75 / 100. The common scale becomes 1000, **Kp** = 2500 / 1000, **Ki** = 2 / 1000 and **Kd** = 750 / 1000. When we pass them through the parameters they become **Kp** = 2500, **Ki** = 2, **Kd** = 750 and **Scale** = 1000. *The Correction becomes = (2500 \* P + 2 \* I + 750 \* D) / 1000.*

This optimization makes the output be calculated in less time, due to the fact that the hub doesn't have to do floating - point arithmetics, which are unoptimised.

