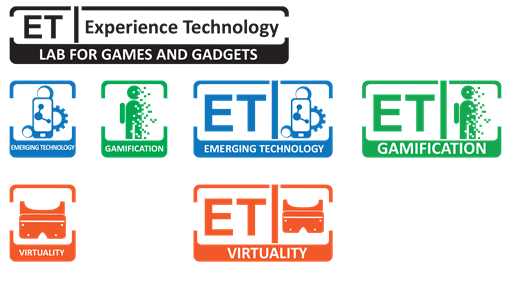
****

**Technical Report**

|  |  |
| --- | --- |
| Title | CooperativeArduinoMazeSolverProject |
| Authors | Sharokh Aria |
| Jesse Koekkoek |
|  |
|  |
| Date | November 2016 |
| Language | English |

|  |  |
| --- | --- |
|  | |
| **Project** | Cooperative Arduino Maze Solver Project |
| Keywords | PID; Arduino; Line Tracking |

W

Project Phase

Paper Grade

TECHNICAL REPORT

CooperativeArduinoMazeSolverProject

| Sharokh Aria  Haagse Hogeschool  12059250@student.hhs.nl |  | Jesse Koekkoek  Haagse Hogeschool  13090437@student.hhs.nl |
| --- | --- | --- |

Abstract / Samenvatting—Hier een samenvatting in maximaal 150 woorden

## Keywords

# PID; Arduino; Line Tracking;

# Introduction

This white paper describes the theory behind a Proportional Integral Derivative controller (PID controller) and its application in the Cooperative Arduino Maze Solver Project. In this project the development of smart artificial intelligence for autonomous robots is key and therefore we have looked into the advancement of the line tracking algorithm. In our research we came across several implementations of PID controllers into line tracking robots. These robots showed the advantages of having a PID controller to regulate the line tracking and the developers behind them showed several ways to build your own. The PID controller provides a relatively easy and accurate way to make a robot track a line. The only possible issue could be ninety degree corners.

# Background

# This project is housed and facilitated by the Experience Technology lab at The Hague University of Applied Sciences. The ET lab combines the search for innovation with practical challenges and research. This particular study which is about cooperatively solving a maze with multiple autonomous Arduino agents, is part of a larger project. The end goal of the project is to have multiple agents explore a maze while simultaneously sharing the information they gather with each other. This information can in turn be used to avoid that the robots' paths overlap each other and to tell the robots how to solve the maze the quickest.

# Research design

To solve a maze the Arduino robots have to give information about their position. We use this information to build a maze in our system. If the robots don’t give enough correct information then the whole maze solving wouldn’t work, therefore we had to modify the robots to follow the line and give correct information at all times. The implementation of the PID controller into the line tracking robot could achieve this.

## Proportional Integral Derivative Controller

A proportional-integral-derivative controller is a control algorithm that's commonly used in industrial control systems. It continuously calculates an error value which represents the difference between a desired setpoint and a measured process variable. It then uses that error to apply a correction based on its proportional, integral, and derivative terms. By continuously applying new corrections the controller attempts to minimize the error over time. The algorithm behind the controller is described by:

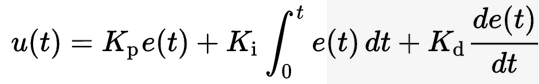


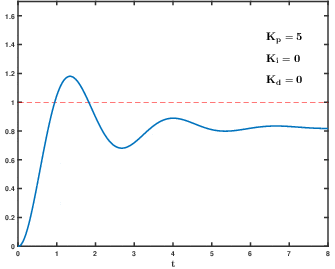
Figure 1: Description of the PID algorithm

where the ,  and  indicate the proportional, integral and derivative terms respectively. The P-term is proportional to the error and represents the present value. The I-term is proportional to the integral of the error and represents the past value. The D-term is proportional to the derivative of the error and represents the future value.

## Effects of the Proportional, Integral and Derivative Terms

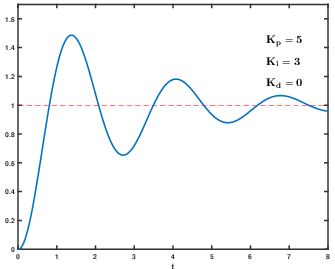
The minimal error that the controller reaches over time is referred to as the steady state error. As the name suggests this error will appear when the error doesn't reduce anymore and keeps the algorithm oscillating around the setpoint. In practice a slight oscillation in the values of the process variable is desired to let a fast system constantly adjust itself slightly so it reacts to changes more quickly.

The aforementioned terms are used to create this oscillating effect around the setpoint. Increasing the gain on the P-term will cause the oscillation of the error to increase. When tuning a PID controller the P-term is applied first to create the required oscillation of the error.



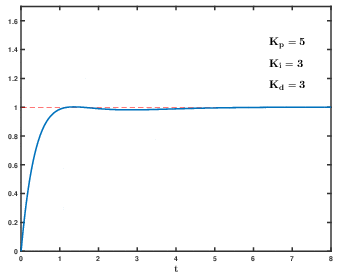
**Figure 2: Example of the resulting graph when only applying the P-term**

Next the I-term is applied. The gain on the I-term causes the overshoot of the error to increase. This results in the constant oscillation around the setpoint. In practice the combination of the P- and I-term is often already sufficient to create an acceptable steady state error in the process variable.



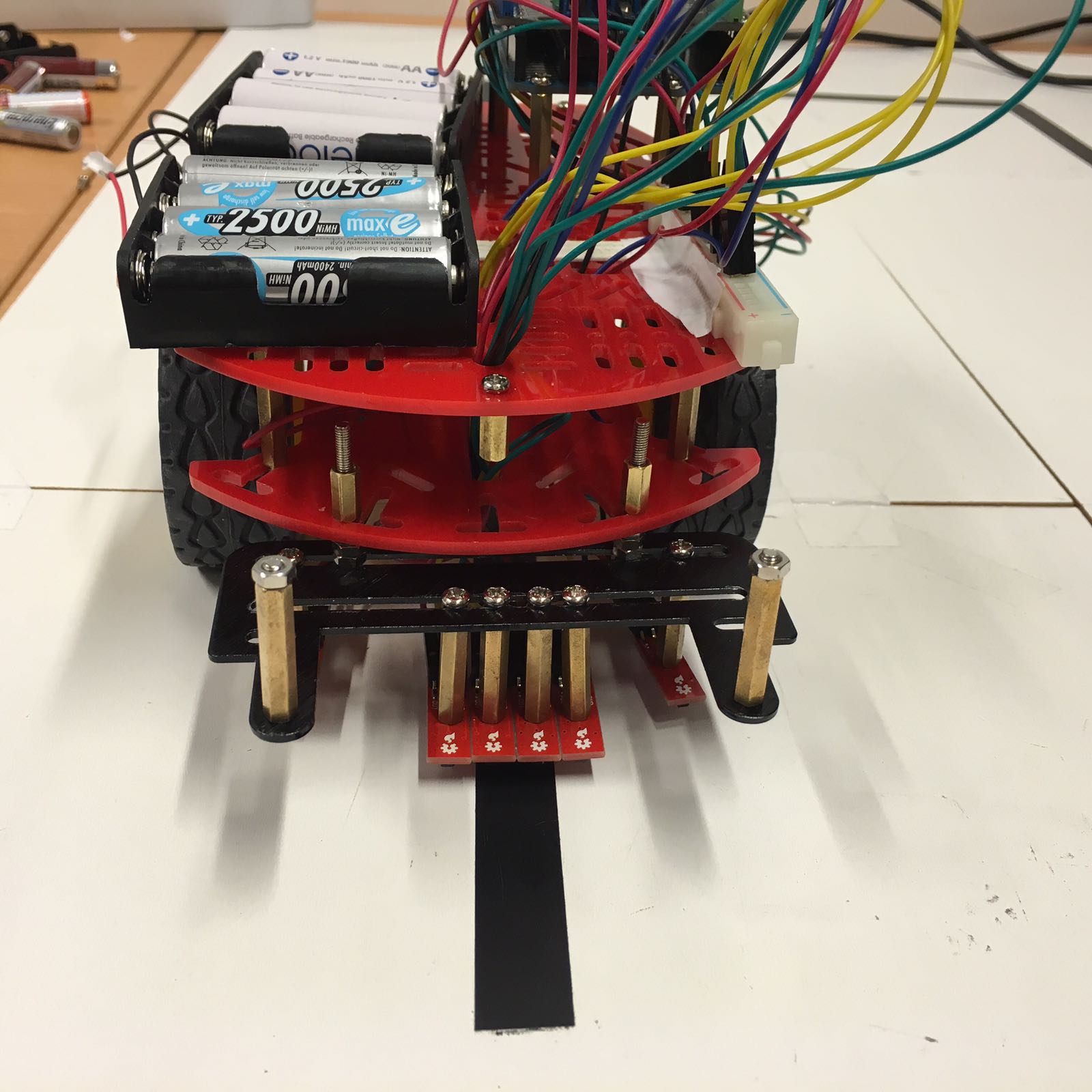
**Figure 3: Example of the resulting graph when applying the P-and the I-term**

If the system is required to stabilize more quickly the gain on the D-term can be increased. A larger D-term causes the overshoot to decrease, which results the steady state error to be reached more quickly and thus makes the system stabilize faster.

 **Figure 4: Example of the resulting graph when applying all three terms**

## Applying a PID Controller to the line tracking robot

To apply the PID algorithm to the line tracking of the robot several conditions had to be taken into account. Firstly the setpoint had to be defined. The goal was to keep the line under the middle of the robot. Attached to the front of the robot were the infrared sensors that reported where the line was every loop iteration. Since we had four sensors available to be used solely for line tracking we set our setpoint to when the line was visible by the middle two of the four sensors.



**Figure 4: Picture showing the line tracker robot when the setpoint is reached**

Secondly we had to define our process variable. The Arduino robots' engines took an integer indicating the amount of power went into them. By adjusting the power to the engines we could influence the course of the robot. By making this integer the process variable of our PID controller we could directly change the course of the robot when necessary.

Finally the definition of the error was left. For this we numbered the sensors from 1 to 4. If a sensor detected the line, it became active and its number was taken into the sum of the total reading. Then the total reading was divided by the amount of active sensors and 2.5 was subtracted from that amount. This resulted in a error with a value between   
-1.5 and 1.5 where 0 indicated that the line was detected by the two middle sensors. If the error had a positive value, the resulting process variable also became positive. In turn if the process variable had a positive result the power of the right motor was lowered. If on the other hand the error was negative and thus the process variable was also negative, then the power of the left motor was lowered. This resulted in the robot constantly adjusting its course according to its PID controller.

void readLine() {

for (int i = 0; i < 4; i++) {

sensorReadings[i] = readSensor(sensor[i]);

if (sensorReadings[i] == 1) {

activeSensors += 1;

}

totalReading += sensorReadings[i] \* (i + 1);

}

avgReading = totalReading / activeSensors;

lastReading = avgReading;

totalReading = 0; activeSensors = 0;

previousError = error;

error = avgReading - 2.5;

totalError += error;

**Figure 5: Code snippet of the retrieval of the error values**

pV = (kp \* error) + (kd \* (error - previousError)) + (ki \* totalError);

if (pV > maxMotorSpeed) {

pV = maxMotorSpeed;

}

if (pV < -maxMotorSpeed ) {

pV = -maxMotorSpeed;

}

if (pV < 0) //Turn Left

{

rightMotorSpeed = maxMotorSpeed;

leftMotorSpeed = maxMotorSpeed - abs(pV);

}

else

{

rightMotorSpeed = maxMotorSpeed - pV;

leftMotorSpeed = maxMotorSpeed;

}

}

**Figure 6: Code snippet of calculating the process variable and determining which motor to apply it to**

Zorg ervoor dat alle tekst in één van de voor gedefinieerde stijlen is. Er is in principe geen enkele tekst met eigen opmaak (hooguit een keertje vet of schuin binnen de standaardtekst).

## Nog een subkopje

Zorg er ook voor dat het titelblad op de juiste manier is ingevuld. Alleen het vakje Grade blijft leeg. Dat vullen de docenten in.

## En nog één

Elk plaatje en elke tabel heeft en onderschrift. Tegenwoordig staan die allemaal onder het object. Verwijs naar alle plaatjes / tabellen. Gebruik een kruisverwijzing. Zie bijvoorbeeld: Tabel 1 en Afbeelding 1

Plaatjes en tabellen beslaan of 1 of 2 kolommen. Het liefst staan ze bovenaan of onderaan een kolom. Gebruik minimaal 300dpi.

## Een laatste…

Zo ziet sourceode eruit

# Results

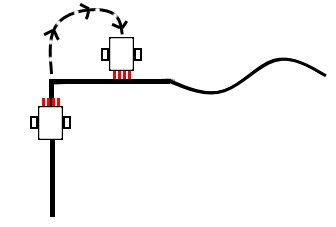
The result of applying a PID controller to the line tracking of the Arduino robot was a more reliable and a more accurate course. The robot no longer kept zigzagging across the line and more easily gained speed on courses without sharp turns. As shown in the table of test results below the average efficiency of the line tracking increased by approximately 250%.

| Code conditions | **Average Time (s)** |
| --- | --- |
| Without PID | 34,47 |
| With PID | 14,30 |

Hou je bij het gebruik van referenties aan de standaard van de IEEE. Maak gebruik van de citation manager van Word. De referenties zien er dan zo [[1](#Coc05)] [[2](#Bro20)] of zo [[3](#Pat98)] uit.

# Conclusion & Discussion

We can conclude that the application of a PID controller to the line tracking of Arduino robot was a success. Line tracking became more reliable and more accurate. The robot kept a stable course and could pick up speed when the line went straight. Overall the robot's line tracking performance greatly improved. The drawback this system still had was that the robot still couldn't make very sharp turns. When the line took a close to ninety degree turn the robot would respond with turning smoothly in the direction of the turn as shown in figure 7. This resulted in a wide turning arc after which the sensors would sometimes all see the line at once when the robot found the line again. It meant a difference in millimetres if the robot would decide to turn back to the ninety degree corner again or continue following the line down the right path. The robot could be tuned to make sharper turns by modifying the numbers of the error calculation. For example the result could be multiplied to increase the effect on the motor speeds when the process variable is applied.



**Figure 7: Example of the robot making a ninety degree turn**

**References**

x

|  |  |
| --- | --- |
| [1] | Karl Johan Aström. 2002. *Control System Design*[Online]. Available FTP: cds.caltech.edu Directory: ~murray/courses/cds101/fa02/caltech/ File: astrom-ch6.pdf |
| [2] | Vincent Broeren, "Eindelijk mijn Scriptie," vol. I, no. 12, pp. 1-98, December 2020. |
| [3] | Henk Patat, "Eet Friet, of eet het niet," *Snackbar Magazine*, vol. II, no. 3, pp. 1-4, Februari 1998. |

x