

MODUL 7 Sistem Kendali PID : Kasus P,I, dan D

1. JUDUL PRAKTIKUM

Sistem Kendali PID Kasus P, I, dan D dengan EEPROM

2. MAKSUD DAN TUJUAN

Maksud dan tujuan dari praktikum ini adalah :

1. Mahasiswa dapat memahami fungsi dan cara kerja PID pada motor DC
2. Mahasiswa dapat membuat program berbasis timer untuk melakukan algoritma PID.
3. Mahasiswa dapat menggunakan peripheral berupa *push button* untuk menambah konstanta K_p , K_i , dan K_d .

3. PARAMETER PENILAIAN

No.	Parameter	Persentase (%)
1.	Lembar Penilaian Praktikum	40%
2.	Jurnal/Laporan Praktikum	60%

4. PERALATAN DAN BAHAN

Alat dan Bahan :

1. Robot Kit Line Follower
2. Baterai LiPo 2-Cell 1300 mAh
3. Kabel Mini-USB
4. Arduino Nano
5. Battery Checker
6. Battery Balancer

Perangkat Lunak :

1. Software IDE Arduino
2. Software Proteus (untuk simulasi)

5. TEORI DASAR

5.1. Sistem Kendali PD

Jika dengan kasus P dan D telah mengendalikan pergerakan robot dengancukup smooth, maka penambahan Integratif menjadi opsional. Komponen Integratif (I) digunakan untuk mengakumulasi error dan mengetahui durasi error. Dengan menjumlahkan error disetiap pembacaan *process value* (PV) akan memberikan akumulasi offset yang harus diperbaiki sebelumnya. Saat robot bergerak menjauhi garis, maka nilai error akan bertambah. Semakin lama tidak mendapatkan *set point* (SP), maka semakin besar nilai I. Dengan mendapatkan nilai K_i yang

tepat, imbas dari Integratif bisa dikurangi. Nilai akumulasi error didapat dari: $\text{error} + \text{last_error}$. Proses perhitungan integral secara intuitif melibatkan komponen waktu, sehingga implementasinya dibutuhkan suatu fungsi timer pada mikrokontroler. Dalam Arduino untuk menghitung waktu atau fungsi timer digunakan fungsi `millis()` menghasilkan jumlah milidetik semenjak program berjalan. Fungsi ini memiliki tipe data Unsigned Long. Jumlah milidetik yang dihasilkan ini harus dicatat dalam setiap perulangan program.

Nilai konstanta perhitungan PID di-*tuning* secara *trial and error* dan proses ini dilakukan dengan metode eksperimental. Nilai proporsional, derivative, dan integratif pada algoritma PID diujicoba hingga ditemukan hasil sistem yang stabil. Adapun cara yang dilakukan untuk men-*tuning* PID pada robot line follower adalah sebagai berikut:

1. Langkah awal gunakan kontrol proporsional terlebih dahulu, abaikan konstanta integratif dan derivatifnya dengan memberikan nilai nol pada integratif dan derivatif.
2. Tambahkan terus konstanta proporsional maksimum hingga keadaan stabil namun robot masih berosilasi.
3. Untuk meredam osilasi, tambahkan konstanta derivatif dengan membagi dua nilai proporsional, amati keadaan sistem robot hingga stabil dan lebih responsif.
4. Jika sistem robot telah stabil, kontrol integratif dapat menjadi opsional, dalam artian jika ingin mencoba-coba tambahkan kontrol integratif tersebut, namun pemberian nilai integratif yang tidak tepat dapat membuat sistem robot menjadi tidak stabil.
5. Nilai set point kecepatan dan nilai batas bawah/atas memberikan patokan kecepatan robot.
6. Nilai time sampling (waktu cuplik) juga mempengaruhi perhitungan PID, tentunya saat penggunaan kontrol integratif dan derivatif.
7. Periksa kembali perfoma sistem hingga mendapatkan hasil yang memuaskan.

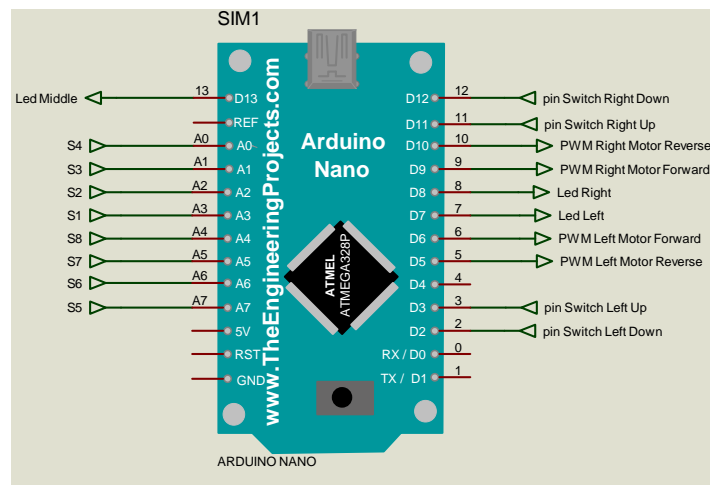
PROSEDUR PRAKTIKUM

A. Percobaan dalam praktikum

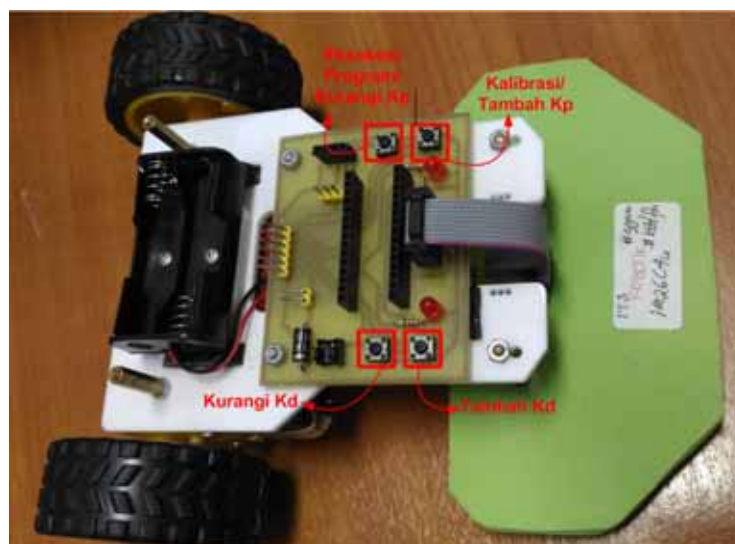
1. Kasus Percobaan



Gambar 1 Contoh susunan dan urutan sensor pada robot line follower.



Gambar 2 Pin Layout Arduino pada Robot Line Follower.



Gambar 3 Posisi dan fungsi push button pada robot line follower.

- a. Modifikasi program pada praktikum sebelumnya dengan variable dan fungsi berikut ini :
- Tambahkan variabel integer dengan nama '**state**' dengan nilai awal adalah 0.
 - Tambahkan variabel integer dengan nama '**setting**' dengan nilai awal adalah 0.
 - Tambahkan variabel integer berupa array dengan nama '**peka**' dari 0 hingga 7 dengan nilai awal adalah 500.
 - Tambahkan variabel integer **errSum**
 - Tambahkan variabel unsigned long **sampleTime** dengan nilai awal 1000. Angka 1000 menunjukkan nilai 1 detik.
 - Tambahkan variabel **now**, **lastTime**, dan **timeChange** dengan tipe data **unsigned long**.
 - Tambahkan variabel integer dengan nama '**Kp**' dengan nilai awal adalah 20.
 - Tambahkan variabel integer dengan nama '**Ki**' dengan nilai awal adalah 20.
 - Tambahkan variabel integer dengan nama '**Kd**' dengan nilai awal adalah 5.

 - Di dalam **void setup** tambahkan :
 - Baca nilai eeprom yang terdapat di EEPROM dengan alamat 0 hingga 7 (dengan ketentuan alamat 0 adalah untuk data nilai tengah kalibrasi sensor 1, alamat 1 untuk nilai tengah kalibrasi sensor 2, dan seterusnya hingga sensor 8) dengan perintah **EEPROM.read(0)** hingga **EEPROM.read(7)**, kemudian simpan didalam variabel **peka[0]** hingga **peka[7]** dengan mengkalikan dengan angka 4 (contoh : **peka[0]=EEPROM.read[0]*4**).
 - Baca nilai eeprom yang terdapat di EEPROM dengan alamat 8 (ketentuan alamat 8 adalah data nilai Kp), kemudian simpan didalam variabel **Kp**.
 - Baca nilai eeprom yang terdapat di EEPROM dengan alamat 9 (ketentuan alamat 9 adalah data nilai Kd), kemudian simpan didalam variabel **Kp**.
 - Tampilkan data **peka[0]** hingga **peka[7]**, **Kp** dan **Kd** yang berasal dari data EEPROM ke Serial Monitor.

 - Di dalam **void loop** ditambahkan *conditional* sebagai berikut :

Referensi posisi push button dapat dilihat pada

 - Jika tombol kiri belakang ditekan dan setting bernilai 0 maka '**state**' akan bernilai 1.
 - Jika tombol kiri depan ditekan dan setting bernilai 0 maka '**state**' akan bernilai 2 dan '**setting**' akan bernilai 1.
 - Jika tombol kiri belakang ditekan dan setting bernilai 1 maka '**state**' akan bernilai 3.
 - Jika tombol kiri depan ditekan dan setting bernilai 1 maka '**state**' akan bernilai 4.
 - Jika tombol kanan belakang ditekan dan setting bernilai 1 maka '**state**' akan bernilai 5.

- Jika tombol kanan depan ditekan dan setting bernilai 1 maka '**state**' akan bernilai 6.
- Jika tombol kanan dan kiri depan ditekan secara bersamaan dan setting bernilai 1 maka '**state**' akan bernilai 7.
- Jika tombol kanan dan kiri belakang ditekan secara bersamaan dan setting bernilai 1 maka '**state**' akan bernilai 8.

Perubahan state tersebut akan mengaktifkan sub program berikut :

- Jika '**state**' bernilai 1 maka proses *auto calibration* berlangsung.
 - Jika '**state**' bernilai 2 maka nilai terakhir variabel '**peka[0]**' hingga '**peka[7]**' disimpan ke dalam EEPROM dengan alamat 0 hingga 7 dan menjalankan perintah proses *line follower* dengan sistem kendali PID dengan nilai **peka[0]** hingga **peka[7]**, **Kp** dan **Kd** yang diperoleh dari data EEPROM dan atau berasal dari hasil autocallibration.
 - Perintah **EEPROM.write(alamat,value)** dengan value yang dibagi dengan 4. Jelaskan mengapa value harus dikali dan dibagi 4!
Contoh : **EEPROM.write(0, peka[0]/4)**
 - Jika '**state**' bernilai 3 maka LED kiri akan menyala dan setelah 500 milidetik LED kiri akan mati, mengurangi nilai Kp(contoh : $Kp=Kp-1$);, menyimpan nilai Kp kedalam EEPROM pada alamat 8.
 - Jika '**state**' bernilai 4 maka LED kiri akan menyala dan setelah 500 milidetik LED kiri akan mati, menambah nilai Kp(contoh : $Kp=Kp+1$);, menyimpan nilai Kp kedalam EEPROM pada alamat 8.
 - Jika '**state**' bernilai 5 maka LED kanan akan menyala dan setelah 500 milidetik LED kiri akan mati, mengurangi nilai Kd(contoh : $Kd=Kd-1$);, menyimpan nilai Kd kedalam EEPROM pada alamat 9
 - Jika '**state**' bernilai 6 maka LED kanan akan menyala dan setelah 500 milidetik LED kiri akan mati, menambah nilai Kd(contoh : $Kd=Kd+1$);, menyimpan nilai Kd kedalam EEPROM pada alamat 10
 - Jika '**state**' bernilai 7 maka LED kiri dan kanan akan menyala dan setelah 500 milidetik LED kiri dan kanan akan mati, menambah nilai Ki(contoh : $Ki=Ki+1$);, menyimpan nilai Ki kedalam EEPROM pada alamat 11.
 - Jika '**state**' bernilai 8 maka LED kiri dan kanan akan menyala dan setelah 500 milidetik LED kiri dan kanan akan mati, mengurangi nilai Ki(contoh : $Ki=Ki-1$);, menyimpan nilai Ki kedalam EEPROM pada alamat 12.
- b. Gunakan fungsi **millis()** untuk menghitung jumlah milidetik semenjak program berjalan. Output dari fungsi ini memiliki tipe data **Unsigned Long**. Simpan output fungsi ini dalam variable **now**. Hitung selisih antara variabel waktu **now** dengan **lastTime** dalam variable **timeChange**.

- Simpan nilai error saat ini dengan variabel **error** dan nilai error sebelumnya dengan variable **lastError**.
- Simpan nilai selisih antara **lastError** dengan **error** dalam variabel **rate**.
- Simpan nilai penjumlahan antara hasil perkalian **error** dan **timeChange** dengan **errSum** dalam variable **errSum**. Hak ini menunjukkan bahwa nilai variabel **errSum** selalu di-update setiap saat.
- Simpan nilai **moveControl** $= (K_p \times \text{Error}) + (K_i \times \text{errSum}) + (K_d \times \text{rate})$.
- Simpan nilai **kecepatanMotorKanan** = **kecepatanSetPoint** dikurang **moveControl**
- Simpan nilai **kecepatanMotorKiri** = **kecepatanSetPoint** ditambah **moveControl**
- Kecepatan Motor Kiri dengan nilai analog sebesar **kecepatanMotorKiri**
- Kecepatan Motor Kanan dengan nilai analog sebesar **kecepatanMotorKanan**

e. *Screenshot* keluaran serial monitor untuk setiap kondisi. Cetak dan tempelkan pada buku jurnal praktikum.

e. Isi tabel kebenaran dari sistem pada Tabel 1 dan tuliskan pada buku jurnal praktikum.

[illegible]

6. Jurnal Praktikum

- a. Jurnal pada Buku Praktikum harus memuat konten sebagai berikut :
 - Judul Praktikum :
 - Maksud dan Tujuan Praktikum :
 - Peralatan dan Bahan Praktikum :
 - Dasar Teori
 - Foto Peralatan dan Bahan Praktikum :
 - Hasil Praktikum (Tulis tangan kode program yang telah diberi komentar/penjelasan beserta foto hasil percobaan yang telah diberi nama dan NIM anggota kelompok)
 - Kesimpulan Praktikum