KOMPARASI SISTEM KOMUNIKASI SERIAL *MULTIPOINT* PADA ROBOT *MANAGEMENT* SAMPAH MENGGUNAKAN I2C DAN SPI

SKRIPSI

KONSENTRASI TEKNIK ELEKTRONIKA

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:
TAUFIQ YUDI SULISTIYONO
NIM. 125060309111004-63

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
MALANG
2014

LEMBAR PERSETUJUAN

KOMPARASI SISTEM KOMUNIKASI SERIAL *MULTIPOINT* PADA ROBOT *MANAGEMENT* SAMPAH MENGGUNAKAN I2C DAN SPI

SKRIPSI

KONSENTRASI TEKNIK ELEKTRONIKA

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:

TAUFIQ YUDI SULISTIYONO
NIM. 125060309111004-63

Telah diperiksa dan disetujui oleh :

Dosen Pembimbing I	Dosen Pembimbing II
Ir. Nurussa'adah, MT	**Eka Maulana, ST., MT., M.Eng**
NIP. 19680706 199203 2 001	**NIK. 841130 06 1 1 0280**

LEMBAR PENGESAHAN

KOMPARASI SISTEM KOMUNIKASI SERIAL *MULTIPOINT* PADA ROBOT *MANAGEMENT* SAMPAH MENGGUNAKAN I2C DAN SPI

SKRIPSI

KONSENTRASI TEKNIK ELEKTRONIKA

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:

TAUFIQ YUDI SULISTIYONO

NIM. 125060309111004-63

Skripsi ini telah diuji dan dinyatakan lulus pada
tanggal 11 Juni 2014

Dosen Penguji

Ir. Nanang Sulistiyanto, MT	**Ir. Ponco Siwindarto, M.Eng.Sc**
NIP. 19700113 199403 1 002	**NIP. 19590304 198903 1 001**

Ir. M. Julius St, MS
NIP. 19540720 198203 1 002

Mengetahui,
Ketua Jurusan Teknik Elektro

M. Aziz Muslim, ST., MT., Ph.D
NIP. 19741203 200012 1 001

PENGANTAR

Alhamdulillah, puji dan syukur penulis panjatkan kepada Allah SWT, karena atas segala petunjuk dan nikmat-Nya lah skripsi ini dapat diselesaikan.

Skripsi berjudul "Komparasi Sistem Komunikasi Serial *Multipoint* pada Robot *Management* Sampah menggunakan I2C dan SPI" ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Universitas Brawijaya.

Penulis menyadari bahwa dalam penyusunan skripsi ini tidak terlepas dari bantuan berbagai pihak. Oleh karena itu, dengan ketulusan dan kerendahan hati penulis menyampaikan terima kasih kepada:

- Allah SWT atas rahmat dan hidayah yang telah diberikan,
- Rosulullah Muhammad SAW, semoga shalawat serta salam selalu tercurah kepada beliau,
- Ayah dan Ibu atas segala nasehat, kasih sayang, perhatian dan kesabarannya didalam membesarkan dan mendidik penulis, serta telah banyak mendoakan kelancaran penulis hingga terselesaikannya skripsi ini,
- Bapak M. Aziz Muslim, ST., MT., Ph.D selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya,
- Bapak Hadi Suyono, ST., MT., Ph.D selaku Sekretaris Jurusan Teknik Elektro Universitas Brawijaya,
- Bapak Moch. Rif'an, ST., MT. selaku Ketua Prodi Strata Satu Jurusan Teknik Elektro Universitas Brawijaya,
- Bapak Teguh Utomo, ST., MT. selaku dosen Penasehat Akademik,
- Ibu Ir. Nurussa'adah, MT. selaku Ketua Kelompok Dosen Keahlian Elektronika Jurusan Teknik Elektro Universitas Brawijaya serta sebagai Dosen Pembimbing I atas segala bimbingan, pengarahan, ide, saran, motivasi, dan masukan yang diberikan,
- Bapak Eka Maulana, ST., MT., M.Eng sebagai Dosen Pembimbing II atas segala bimbingan, pengarahan, ide, saran, motivasi, dan masukan yang diberikan,
- Seluruh dosen pengajar Teknik Elektro Universitas Brawijaya,

- Staff Recording Jurusan Teknik Elektro,
- Teman – teman SAP angkatan 2012,
- Rekan seperjuangan dalam skripsi, Imam, Bambang, Deaz, Ebay, Ridho terima kasih atas segala bantuan yang telah diberikan,
- Seluruh teman-teman serta semua pihak yang tidak mungkin bagi penulis untuk mencantumkan satu-persatu, terimakasih banyak atas bantuan dan dukungannya.

Pada akhirnya, penulis menyadari bahwa skripsi ini masih belum sempurna. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang membangun. Penulis berharap semoga skripsi ini dapat bermanfaat bagi pengembangan ilmu pengetahuan dan teknologi serta bagi masyarakat.

Malang, Mei 2014

Penulis

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

ABSTRAK

Sampah merupakan masalah yang sering kita jumpai dalam kehidupan sehari-hari. Hal ini perlu diatasi karena setiap manusia pasti memproduksi sampah, disisi lain masyarakat tidak ingin berdekatan dengan sampah. Oleh karena itu dalam makalah ini penulis akan membahas bagian dari robot *management* sampah.

Pada robot ini menggunakan 4 buah mikrokontroller sehingga diperlukan protokol komunikasi serial *multipoint*. Komunikasi ini harus dirancang dengan baik agar tidak terjadi kesalahan dalam pengiriman maupun penerimaan data serta kecepatan transfer data dapat maksimal sehingga informasi dari sensor selalu terbaca oleh robot.

Empat buah mikrokontroller tersebut terdiri dari 1 *Master* dan 3 *Slave*. Fungsi masing-masing dari 3 mikrokontroller *slave* tersebut adalah untuk membaca sensor, mengendalikan motor kiri, dan mengendalikan motor kanan. Sedangkan fungsi dari mikrokontroller *master* adalah sebagai pemroses data dari *slave* pembaca sensor dan hasilnya di kirim ke *slave* pengendali motor kanan dan *slave* pengendali motor kiri. Dengan sistem seperti ini diharapkan waktu pemrosesan lebih cepat dibandingkan hanya menggunakan 1 mikrokontroller.

Kata kunci : robot, mikrokontroler, serial, *multipoint*

BAB I
PENDAHULUAN

1.1 Latar Belakang

Penggunaan jasa petugas kebersihan untuk membersihkan sampah dari setiap tempat sampah di dalam suatu gedung merupakan hal yang sangat umum dilakukan demi terciptanya keadaan gedung yang bersih. Penggunaan jasa ini memiliki kekurangan dalam hal tenaga dan penjadwalan apalagi kebiasaan masyarakat yang kurang disiplin menyebabkan terjadi penumpukan sampah berlebih di setiap tempat sampah (Nuraini *et al*, 2011).

Sistem robot telah menyebabkan kemajuan penting dalam bidang otomatisasi. Bahkan sejak robot industri diberlakukan, kinerja robot mempunyai kelebihan yang signifikan dibandingkan kinerja manusia seperti tahan lama, presisi, dan tidak mudah lelah. Robot *Cleaning Services* dimaksudkan untuk melakukan berbagai aktivitas, termasuk tugas-tugas rumah tangga. Hal ini telah membawa kontribusi yang cukup besar dalam dua dekade terakhir (Alexandrescu *et al*, 2010).

Dalam sistem robot biasanya terdapat beberapa mikrokontroler yang mempunyai fungsi bermacam-macam. Mikrokontroler tersebut saling berkomunikasi satu sama lain dengan protokol tertentu. Jenis protokol yang sering digunakan adalah sistem komunikasi serial *multipoint*. Sistem komunikasi serial *multipoint* mempunyai beberapa keuntungan yaitu dapat menghubungkan lebih dari 2 *device* dan mudah di *upgrade* (penambahan *device*).

Dewasa ini sistem komunikasi serial *multipoint* yang sering digunakan pada robot adalah protokol I2C dan SPI. Kedua protokol tersebut mempunyai kelebihan dan kekurangan masing-masing. Pengkabelan pada protokol I2C lebih sederhana dari pada protokol SPI. Namun kecepatan transmisi data pada protokol I2C lebih lambat dari pada protokol SPI.

Agar sistem dapat bekerja optimal biasanya dipilih protokol SPI karena mempunyai kecepatan transmisi data yang lebih cepat. Namun kita belum mengetahui bagaimana respons sistem jika dibandingkan dengan protokol I2C. Berdasarkan latar belakang tersebut penulis mengangkat judul "Komparasi Sistem

Komunikasi Serial *Multipoint* pada Robot *Management* Sampah menggunakan I2C dan SPI".

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, dapat disusun rumusan masalah sebagai berikut :

1) Bagaimana merancang dan membuat rangkaian antarmuka serial *multipoint* mikrokontroler dengan protokol I2C dan SPI.
2) Bagaimana merancang dan membuat perangkat lunak sistem mikrokontroler untuk berkomunikasi serial *multipoint* dengan protokol I2C dan SPI.
3) Bagaimana menganalisis perbandingan antara protokol I2C dan SPI pada performa robot *management* sampah.

1.3 Batasan Masalah

Dengan mengacu pada permasalahan yang telah dirumuskan, maka hal-hal yang berkaitan dengan alat akan diberi batasan sebagai berikut :

1) Jumlah mikrokontroler adalah 1 Master dan 3 Slave. Agar waktu pemrosesan lebih cepat.
2) Protokol komunikasi yang dikomparasi adalah I2C dan SPI.
3) Mikrokontroler Master yang digunakan adalah AVR ATMega16. Sedangkan 3 mikrokontroler Slave adalah AVR ATMega8.
4) Bahasa program yang digunakan adalah bahasa C.
5) Software tool yang digunakan adalah AVR Studio 4.18, WinAVR/AVR-GCC 1.6.

1.4 Tujuan

Penelitian ini bertujuan untuk merancang robot yang di dalamnya terdapat beberapa mikrokontroler yang saling terhubung dengan protokol komunikasi serial *multipoint* I2C dan SPI. Robot tersebut bergerak mengikuti garis dengan pembacaan jalur hitam di lantai. Kemudian menganalisis perbandingan respon robot ketika menggunakan sistem komunikasi serial *multipoint* I2C dan SPI.

1.5 Sistematika Penulisan

Sistematika penulisan dalam penelitian ini sebagai berikut:

BAB I Pendahuluan

Memuat latar belakang, rumusan masalah, ruang lingkup, tujuan, dan sistematika pembahasan.

BAB II Tinjauan Pustaka

Membahas teori-teori yang mendukung dalam perencanaan dan pembuatan alat.

BAB III Metodologi

Berisi tentang metode-metode yang dipakai dalam melakukan perancangan, pengujian, dan analisis data.

BAB IV Perancangan

Perancangan dan perealisasian alat yang meliputi spesifikasi, perencanaan blok diagram, prinsip kerja dan realisasi alat.

BAB V Pengujian dan Analisis

Memuat aspek pengujian meliputi penjelasan tentang cara pengujian dan hasil pengujian. Aspek analisis meliputi penilaian atau komentar terhadap hasil-hasil pengujian. Pengujian dan analisis ini terhadap alat yang telah direalisasikan berdasarkan masing-masing blok dan sistem secara keseluruhan.

BAB VI Kesimpulan dan Saran

Memuat intisari hasil pengujian dan menjawab rumusan masalah serta memberikan rekomendasi untuk perbaikan kualitas penelitian dimasa yang akan datang.

BAB II
TINJAUAN PUSTAKA

2.1.Mikrokontroler ATMega16

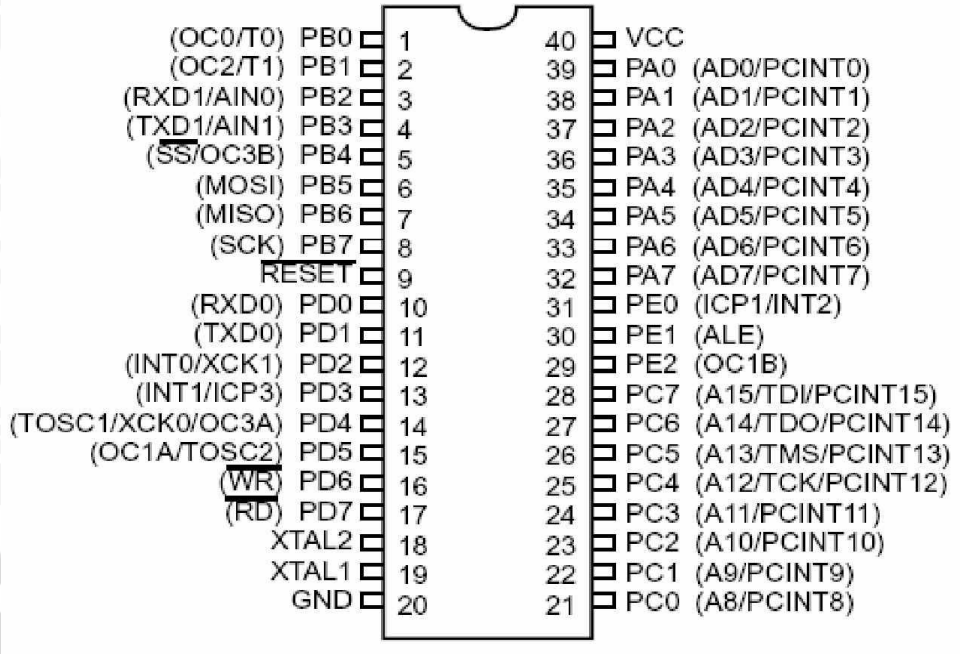
ATMega16 merupakan seri mikrokontroler CMOS 8-bit buatan Atmel, berbasis arsitektur RISC (*Reduced Instruction Set Computer*). Hampir semua instruksi dieksekusi dalam satu siklus clock. ATMega16 mempunyai 32 register *general-purpose*, *timer*/*counter* fleksibel dengan mode *compare*, *interrupt* internal dan eksternal, *serial* UART, *programmable Watchdog Timer*, dan *mode power saving*. ATMega16 mempunyai ADC dan PWM internal. ATMega16 juga mempunyai *In-System Programmable Flash on-chip* yang mengijinkan memori program untuk diprogram ulang dalam sistem menggunakan hubungan serial SPI. Atmega16 mempunyai *throughput* mendekati 1 MIPS per MHz membuat disainer sistem dapat mengoptimasi konsumsi daya versus kecepatan proses. Beberapa keistimewaan dari AVR ATmega16 antara lain:

1) *Advanced* RISC *Architecture*
   - *Most Single-clock Cycle Execution* - 131 *Powerful Instructions*
   - *General Purpose Working Registers* (32 x 8)
   - *Up* to 16 MIPS *Throughput at* 16 MHz
   - *On-chip* 2-*cycle Multiplier*
2) *Non-volatile Program and Data Memories*
   - *In-System Self-programmable Flash Endurance* (16K *Bytes*): 1,000 *Write*/*Erase Cycles*
   - *Optional Boot Code Section with Independent Lock Bits In-System Programming by On*-chip *Boot Program True Rea*d-*While-Write Operation*
   - EEPROM *Endurance* (512 *Bytes*): 100,000 *Write*/*Erase Cycles*
   - *Internal* SRAM : 1K *Bytes*
   - *Up to* 64K *Bytes Optional External Memory Space*
   - *Programming Lock for Software Security*
3) *Peripheral Features*
   - *Two* 8-*bit Timer*/*Counters with Separate Prescalers and Compare Modes*

- *Two* 16-*bit Timer*/*Counters with Separate Prescalers, Compare Modes, and Capture Modes*
- *Real Time Counter with Separate Oscillator*
- *Six* PWM *Channels*
- *Dual Programmable Serial* USARTs
- *Master*/*Slave* SPI *Serial Interface*
- *Programmable Watchdog Timer with Separate On-chip Oscillator*
- *On-chip Analog Comparator*

4) *Special Microcontroller Features*
   - *Power-on Reset and Programmable Brown-out Detection*
   - *Internal Calibrated* RC *Oscillator*
   - *External and Internal Interrupt Sources*
   - *Five Sleep Modes*: *Idle, Power-save, Power-down, Standby, and Extended Standby*
5) I/O *and Packages*
   - *Programmable* I/O *Lines* (35)
   - PDIP (40-*pin*), 44-*lead* TQFP, *and* 44-*pad* MLF
6) *Operating Voltages*
   - ATmega16V : 1.8 - 3.6V
   - ATmega16L : 2.7 - 5.5V
   - ATmega16 : 4.5 - 5.5V

Untuk memaksimalkan performa dan paralelisme, ATMega16 menggunakan arsitektur Harvard (memori dan bus terpisah untuk program dan data). Penyemat ATMega16 kemasan 40-pin DIP (*Dual In-Line Package)* ditunjukkan dalam Gambar 2.1.



Gambar 2.1. Pin-Pin pada ATMega16 dengan Kemasan 40-Pin DIP (*Dual In-Line Package*)

Sumber: Atmel, 2010

Adapun fungsi dari pin-pin yang terdapat pada Mikrokontroler ATMega16 dijelaskan sebagai berikut :

a) VCC, suplai tegangan digital
b) GND, pin *ground*
c) PORT A (PA0 – PA7). PORT A merupakan port I/O 8-*bit bidirectional* yang dilengkapi dengan resistor *pull-up* internal (dapat dipilih untuk tiap *bit*). Selain sebagai port I/O, PORT A juga mempunyai fungsi lain seperti antarmuka memori eksternal dan pin *change interrupt*.
d) PORT B (PB0 – PB7). PORT B merupakan port I/O 8-*bit bidirectional* dengan resistor *pull-up* internal (dapat dipilih untuk tiap *bit*). Selain itu, PORT B juga mempunyai fungsi lain yaitu *Serial Peripheral Interface* (SPI), *Analog Comparator*, *input/output Timer/Counter*.
e) PORT C (PC0 – PC7). PORT C merupakan port I/O 8-*bit bidirectional* dengan resistor *pull-up* internal (dapat dipilih untuk tiap *bit*). Selain itu PORT C juga mempunyai fungsi lain yaitu JTAG, antarmuka memori eksternal, dan pin *change interrupt*.
f) PORT D (PD0 – PD7). PORT D merupakan port I/O 8-*bit bidirectional* dengan resistor *pull-up* internal (dapat dipilih untuk tiap *bit*). Selain itu PORT D juga

mempunyai fungsi lain yaitu USART, *interupsi eksternal, strobe memori eksternal, timer/counter.*

g) PORT E (PE0 – PE2) PORT E merupakan port I/O 3-*bit bidirectional* dengan resistor *pull-up* internal (dapat dipilih untuk tiap *bit*). Selain itu PORT E juga mempunyai fungsi lain yaitu *timer/counter, latch enable memori eksternal, interupsi eksternal.*
h) RESET, berfungsi untuk mereset mikrokontroler jika diberikan sinyal *active low* dalam selang waktu tertentu.
i) XTAL1, input ke inverting *oscillator* amplifier dan ke rangkaian detak internal.
j) XTAL2, output dari inverting *oscillator amplifier*.

Universal synchronous dan *asynchronous* pemancar dan penerima *serial* adalah suatu alat komunikasi serial sangat fleksibel. Mikrokontroller ATMega16 memiliki dua buah *port* USART untuk komunikasi *serial*, yaitu USART0 dan USART1. Fasilitas komunikasi serial USART mikrokontroler ini memiliki fitur sebagai berikut:

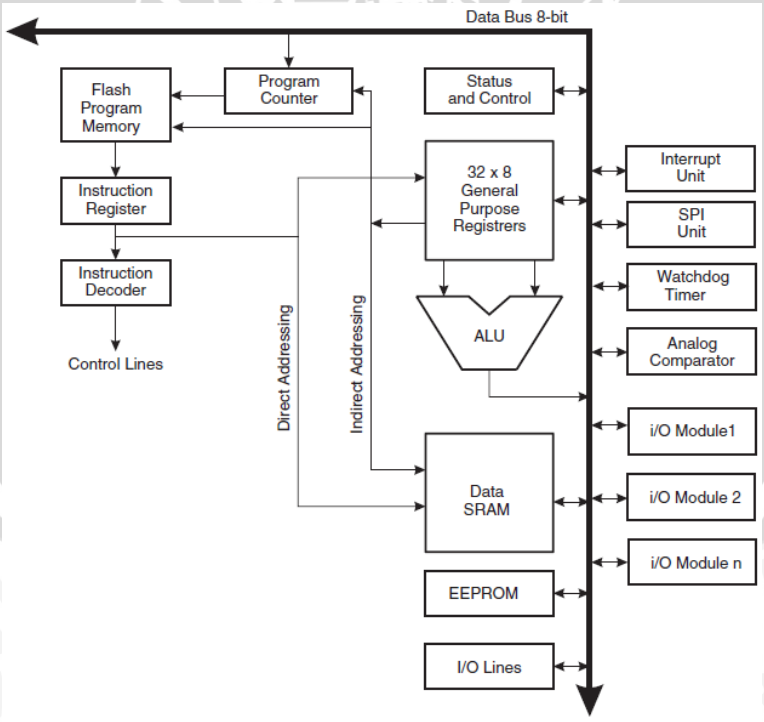
1) *Operasi full duplex* (register penerima dan pengirim *serial* dapat berdiri sendiri)
2) *Operasi Asychronous* atau *synchronous*
3) *Master* atau *slave* mendapat *clock* dengan operasi *synchronous*
4) Pembangkit *baud rate* dengan resolusi tinggi
5) Dukung *frames serial* dengan 5, 6, 7, 8 atau 9 *data bit* dan 1 atau 2 *stop bit*
6) Tahap *odd* atau *even parity* dan *parity check* didukung oleh *hardware*
7) Pendeteksian data *overrun*
8) Pendeteksi *framing error*
9) Pemfilteran gangguan (*noise*) meliputi pendeteksian *bit false start* dan pendeteksian *low pass filter* digital
10) Tiga *interrupt* terdiri atas TX *complete*, TX *data register empty*, dan RX *complete*
11) Mode komunikasi *multi-processor*
12) Mode komunikasi *double speed asynchronous*

2.2.ATMega8

ATMega8 adalah mikrokontroler CMOS 8-bit berarsitektur AVR RISC yang memiliki 8Kbytes *In-System Programmable Flash*. Mikrokontroler dengan konsumsi daya rendah ini mampu mengeksekusi instruksi dengan kecepatan maksimum 16 MIPS pada frekuensi 16MHz. Selain itu ATMega8 juga memiliki beberapa spesifikasi lain, yaitu;

- Memori *Flash* 8 *Kbytes* dan dapat diprogram atau hapus sebanyak 10.000 siklus
- Memori EEPROM 512 *bytes* dan dapat diprogram atau hapus sebanyak 100.000 siklus
- Memori SRAM internal 1 *Kbytes*
- Maksimal 32 pin I/O dan 6 *channel* ADC 10/8 bit pada tipe PDIP
- *External interrupt*, analog komparator dan 3 I/O PWM
- Satu 16-bit timer dan dua 8-bit timer
- Komunikasi serial melalui SPI dan USART
- Fasilitas *In System Programming* (ISP)

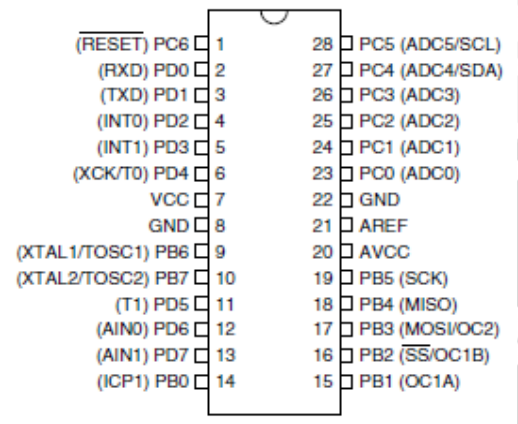
Blok sistem ATMega8 ditunjukkan dalam Gambar 2.2.



Gambar 2.2. Blok Sistem ATMega8

Sumber : Atmel, 2011

Bentuk dari ATMega8 ada yang bertipe PDIP, QFN/MLF dan TQFP, tetapi yang sering digunakan adalah yang tipe PDIP karena mudah dalam pemasangan dan penyolderan. Penyemat ATMega8 (tipe PDIP) ditunjukkan dalam Gambar 2.3.



Gambar 2.3. Pin-pin ATMega8

Sumber : Atmel, 2011

Dari gambar diatas dapat dilihat bahwa ATMega8 tipe PDIP memiliki 28 pin dengan fungsi yang berbeda-beda. Berikut ini adalah penjelasan dari masing-masing pin ATMega8.

Tabel 2.1 Deskripsi pin ATMega8

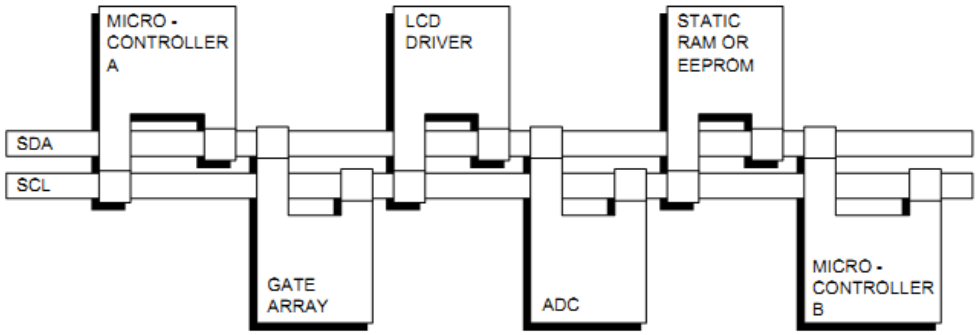
Nama Pin	Deskripsi
PORTB (PB7..PB0)	Port I/O 8-bit dengan resistor *pull-up internal* tiap pin. Khusus PB6 dan PB7 dapat digunakan sebagai *input* dan *output* kristal (*input/output inverting oscillator amplifier*) bergantung pada pengaturan *fuse bit* yang digunakan untuk memilih sumber *clock*. Fungsi lain port B: PB5 → SCK PB2 → SS/OC1B PB4 → MISO PB1 → OC1A PB3 → MOSI/OC2 PB0 → ICP1
PORTC (PC6..PC0)	Port I/O 7-bit dengan resistor pull-up internal tiap pin. Khusus untuk PC6 jika *fuse bit* RSTDISBL di *"programmed"* maka digunakan sebagai pin I/O. Jika *fuse bit* RSTDISBL di *"unprogrammed"*, maka PC6 digunakan

	sebagai pin RESET (aktif *low*). Untuk PC5..PC0 juga bisa digunakan untuk ADC. PC5 dan PC4 juga dapat digunakan untuk komunikasi I2C.
PORTD (PD7..PD0)	Port I/O 8-bit dengan resistor *pull-up internal* tiap pin. Di PD1 dan PD0 dapat digunakan untuk komunikasi USART.
AVCC	Tegangan catu untuk ADC.
AREFF	Untuk pin tegangan referensi *analog* untuk ADC
VCC	Tegangan *Supply*
GND	*Groud*

Sumber : Winoto, 2008

2.3.Sistem Bus I2C

Sistem Bus I2C pertamakali diperkenalkan oleh Philips Semikonduktor (sekarang NXP Semikonduktor) pada tahun 1979. Sistem Bus ini hanya memerlukan dua jalur yaitu SCL dan SDA.



Gambar 2.4. Contoh Konfigurasi Bus I2C menggunakan Dua Mikrokontroler

Sumber : NXP, 2014

Karakter I2C adalah:

1. Data dikirim serial secara per-bit.
2. Menggunakan dua penghantar koneksi dengan ground bersama. Dua penghantar tersebut adalah SCL (Serial Clock Line) untuk menghantarkan sinyal clock dan SDA (Serial Data) untuk mentransaksikan data
3. Jumlah slave maximal 127. Slave dialamatkan melalui 7-bit-alamat.
4. Setiap transaksi data terjadi antara pengirim (*Transmitter*) dan penerima (*Receiver*).

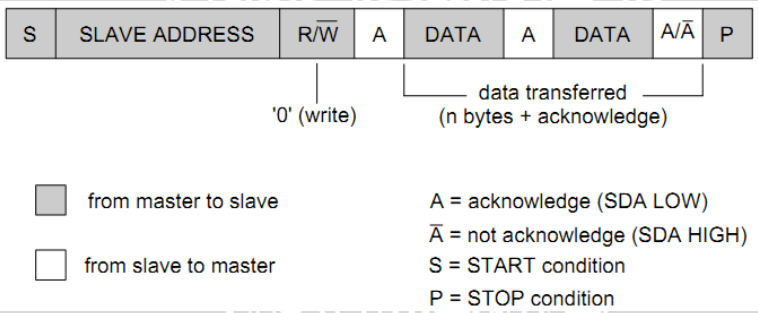
Aturan Komunikasi I2C:

1. Device atau komponen yang mengirim data disebut *transmitter*, sedangkan device yang menerimanya disebut *receiver*.
2. Device yang mengendalikan operasi transfer data disebut master, sedangkan device lainnya yang dikendalikan oleh master disebut slave.
3. Master harus menghasilkan serial clock melalui pin SCL, mengendalikan akses ke BUS serial, dan menghasilkan sinyal kendali START dan STOP.

Mode pengoperasian transfer data ada 2 jenis, yaitu :

1. Transfer data dari master(*transmitter)* ke slave(*receiver).*

   Byte pertama yang dikirimkan oleh master adalah alamat slave, setelah itu master mengirimkan sejumlah *byte* data. Slave atau *receiver* mengirimkan sinyal *acknowledge* setiap kali menerima 1-*byte* data. Pada tiap *byte*, *bit* pertama yang dikirim adalah MSB.

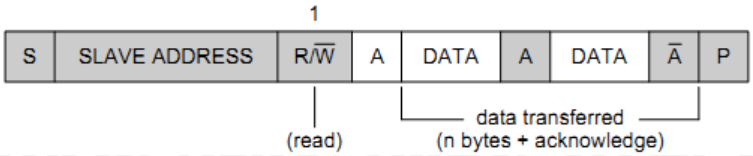


Gambar 2.5. Transfer data dari master ke slave

Sumber : NXP, 2014

2. Transfer data dari slave(*transmitter)* ke master(*receiver).*

   Meskipun master berperan sebagai *receiver*, *byte* pertama dikirimkan oleh master berupa alamat slave. Setelah itu slave mengirimkan bit *acknowledge*, dilanjutkan dengan pengiriman sejumlah *byte* dari slave ke master. Master mengirimkan bit *acknowledge* untuk setiap byte yang diterimanya, kecuali *byte* terakhir. Pada akhir *byte*, master mengirimkan sinyal '*not acknowledge'*, setelah itu master mengirimkan sinyal STOP.



Gambar 2.6. Transfer data dari slave ke master

Sumber : NXP, 2014

Beberapa perangkat yang lebih baru dari seri ATmega berisi modul dukungan untuk antarmuka mikrokontroler ke bus dengan dua jalur, yang disebut TWI. TWI pada dasarnya adalah sama dengan I2C yang diperkenalkan oleh Philips, tetapi istilah I2C dihindari dalam dokumentasi Atmel karena masalah paten.

2.4.Sistem Bus SPI

SPI (*serial peripheral interface*) merupakan salah satu metode pengiriman data dari suatu devais ke devais lainnya. Metode ini merupakan metode yang bekerja pada metode *full duplex* dan merupakan standar sinkronasi serial data link yang dikembangkan oleh Motorola. Pada SPI, devais dibagi menjadi dua bagian yaitu master dan slave dengan master sebagai devais yang menginisiasi pengiriman data. Dalam aplikasinya, sebuah master dapat digunakan untuk mengatur pengiriman data dari atau ke beberapa slave(*Multipoint)*. SPI disebut juga dengan "*four wire"* serial bus untuk membedakannya dengan bus serial tiga, dua, dan satu kebel.

Pin – Pin Penghubung pada SPI

Komunikasi serial data antara master dan slave pada SPI diatur melalui 4 buah pin yang terdiri dari SCLK, MOSI, MISO, dan SS. Berikut ini adalah penjelasan singkat mengenai ke 4 pin tersebut:

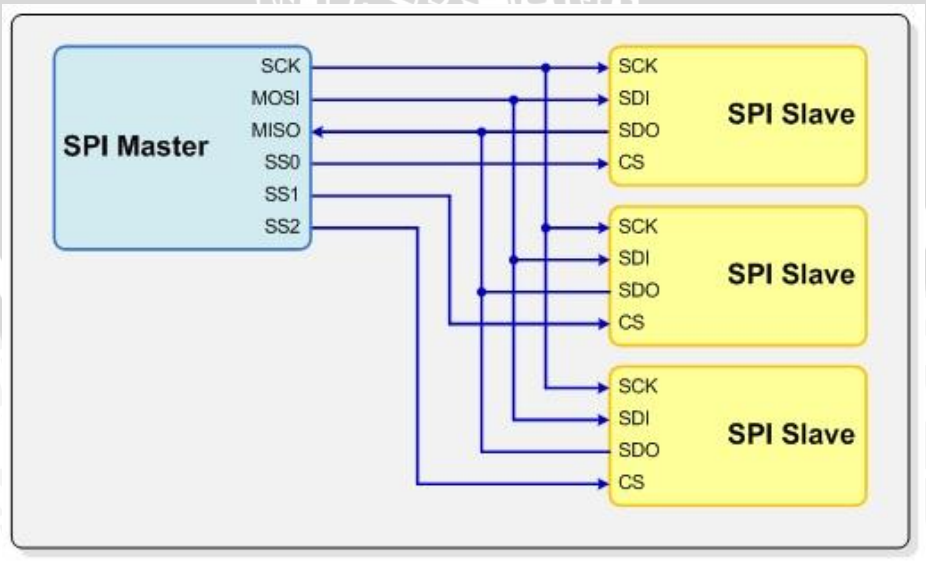
SCLK (serial clock) merupakan data biner yang keluar dari master ke slave yang berfungsi sebagai clock dengan frekuensi tertentu. Clock merupakan salah satu komponen prosedur komunikasi data SPI. Dalam beberapa devais, istilah yang digunakan untuk pin ini adalah SCK.

MOSI (master *out* slave *input*) merupakan pin yang berfungsi sebagai jalur data pada saat data keluar dari master dan masuk ke dalam slave. Istilah lain untuk pin ini antara lain SIMO, SDI, DI, dan SI.

MISO (master *input* slave *output*) merupakan pin yang berfungsi sebagai jalur data yang keluar dari slave dan mesuk ke dalam master. Istilah lain untuk pin ini adalah SOMI, SDO, DO, dan SO.

SS (slave *select*) merupakan pin yang berfungsi untuk mengaktifkan slave sehingga pengiriman data hanya dapat dilakukan jika slave dalam keadaan aktif (active low). Istilah lain untuk SS antara lain CS (chip *select*), nCS, nSS, dan STE (slave *transmit enable*) .

Pin SCLK, MOSI, dan SS merupakan pin dengan arah pengiriman data dari master ke slave. Sebaliknya, MISO mempunyai arah komunikasi data dari slave ke master. Pengaturan hubungan antara pin SDO dan SDI harus sesuai dengan ketentuan. Pin SDO pada master harus dihubungkan dengan pin SDI pada slave, begitu juga sebaliknya. Hal ini penting untuk diperhatikan untuk menghindari terjadinya kesalahan prosedur pada pengiriman data. Istilah pin -pin SPI untuk berbagai devais mungkin saja mempunyai istilah yang berbeda dengan istilah di atas tergantung produsen yang membuatnya.



Gambar 2.7. Konfigurasi Bus SPI dengan *Multiple Slave*

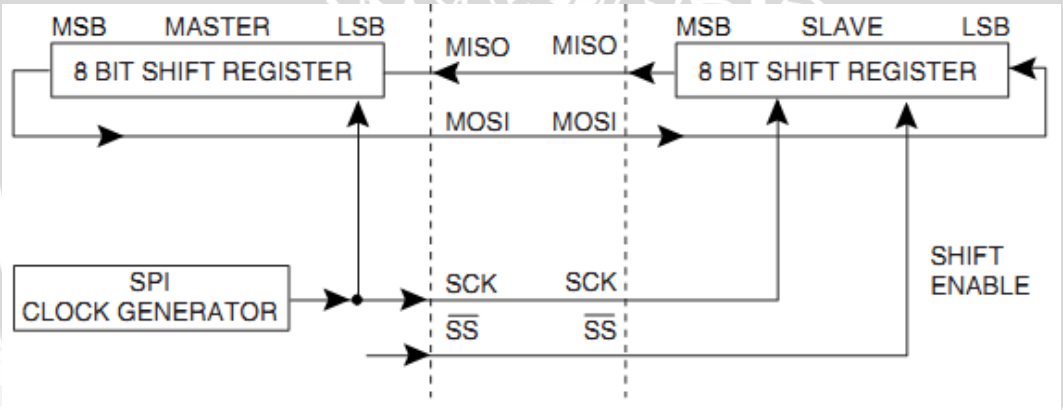
Sumber : Corelis, 2013

Prosedur Operasi SPI

Komunikasi data SPI dimulai pada saat master mengirimkan clock melalui SCK dengan frekuensi lebih kecil atau sama dengan frekuensi maksimum pada slave. Kemudian, master memberi logika nol pada SS untuk mengaktifkan slave sehingga pengiriman data (berupa siklus clock) siap untuk dilakukan. Pada saat siklus clock terjadi, transmisi data *full duplex* terjadi dengan dua keadaan sebagai berikut:

- Master mengirim sebuah bit pada jalur MOSI, slave membacanya pada jalur yang sama.
- Slave mengirim sebuah bit pada jalur MISO, master membacanya pada jalur yang sama.

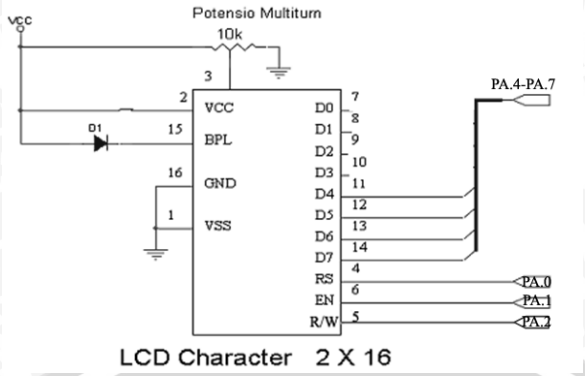
Transmisi dapat menghasilkan beberapa siklus clock. Jika tidak ada data yang dikirim lagi maka master menghentikan clock tersebut dan kemudian menon-aktifkan slave.



Gambar 2.8. SPI Master-Slave Interconnection
Sumber : Atmel, 2011

2.5.*Liquid Crystal Display* (LCD)

Liquid Crystal Display (LCD) merupakan komponen elektronika yang digunakan untuk menampilkan karakter baik berupa karakter angka, huruf, atau karakter lainnya, sehingga tampilan tersebut dapat dilihat secara visual. Gambar 2.9 menunjukkan rangkaian *interface* ke LCD Karakter 2x16 dan Tabel 2.2 menunjukkan Pin-Pin I/O LCD.



Gambar 2.9. Rangkaian *Interface* ke LCD Karakter 2X16

Sumber: Nalwan, 2004

Tabel 2.2. Pin-Pin I/O LCD

No	Simbol	Level	Fungsi
1	Vss		GND
2	Vcc		*Power supply* 5 Volt
3	Vee		LCD Drive
4	RS	H/L	H:*data input* L:*ins input*
5	R/W	H/L	H:*Read* L:*Write*
6	E	H	*Enable signal*
7	DB0	H/L	
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	*Data Bus*
11	DB4	H/L	
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	
15	V-BL		*Power supply* 4 – 4,2 Volt
16	V-BL		GND

Sumber: Nalwan, 2004

Pada perancangan sistem ini memakai LCD modul M1632 yang merupakan sebuah modul LCD dot matrik yang membutuhkan daya kecil. LCD modul M1632 dilengkapi panel LCD dengan tingkat kontras yang cukup tinggi serta pengendali LCD CMOS yang telah terpasang dalam modul tersebut. LCD modul M1632 mempunyai spesifikasi sebagai berikut :

- Memiliki 16 karakter dan 2 baris tampilan yang terdiri atas 5 x 7 dot matrik ditambah dengan kursor.
- Memerlukan catu daya DC 5 V.
- Otomatis *reset* saat catu daya dinyalakan.
- Memiliki data RAM (max 80 karakter) dengan 80 x 8 *display*.
- Menggunakan 4 bit data dan 3 bit kontrol.

2.6.Kontroler

Sistem pengendalian dirancang untuk melakukan dan menyelesaikan tugas tertentu. Syarat utama sistem pengendalian adalah harus stabil. Disamping kestabilan mutlak, maka sistem harus memiliki kestabilan secara relatif, yakni tolok ukur kualitas kestabilan sistem dengan menganalisis sampai sejauh mana batas-batas kestabilan sistem tersebut jika dikenai gangguan (Ogata, 1997). Selain itu analisis juga dilakukan untuk mengetahui bagaimana kecepatan sistem dalam merespons *input*, dan bagaimana peredaman terhadap adanya lonjakan *(over shoot).*

Suatu sistem dikatakan stabil jika diberi gangguan maka sistem tersebut akan kembali ke keadaan *steady state* di mana *output* berada dalam keadaan tetap seperti tidak ada gangguan. Sistem dikatakan tidak stabil jika *output*nya berosilasi terus menerus ketika dikenai suatu gangguan. Karena suatu sistem pengendalian biasanya melibatkan penyimpanan energi maka *output* sistem ketika diberi suatu *input*, tidak dapat mengikuti *input* secara serentak, tapi menunjukkan respons transien berupa suatu osilasi teredam sebelum mencapai *steady state*.

Prinsip kerja kontroler adalah membandingkan nilai aktual keluaran plan dengan nilai referensi. Kemudian kontroler menentukan nilai kesalahan dan akhirnya menghasilkan sinyal kontrol untuk meminimalkan kesalahan (Ogata, 1997).

2.6.1. Kontroler Proporsional

Untuk kontroler dengan aksi kontrol proporsional, hubungan antara keluaran kontroler *m(t)* dan sinyal kesalahan penggerak *e(t)* adalah:

$$m(t)=K_p\, e(t) \quad \text{.....(2-1)}$$

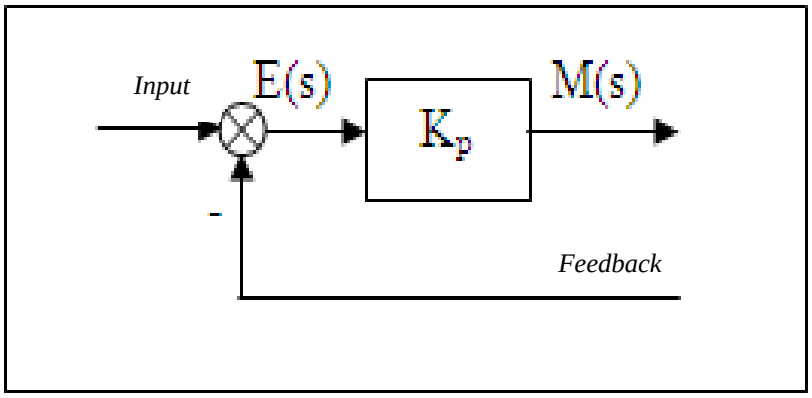
atau, dalam besaran transformasi Laplace,

$$\frac{M(s)}{E(s)} = Kp \quad \text{..... (2-2)}$$

di mana K_p adalah kepekaan proporsional atau penguatan.

Apapun wujud mekanisme yang sebenarnya dan apapun bentuk daya penggeraknya, kontroler proporsional pada dasarnya merupakan penguat dengan penguatan yang dapat diatur (Ogata, 1997).

Diagram blok kontroler proporsional ditunjukkan dalam Gambar 2.10.



Gambar 2.10 Diagram Blok Kontroler Proposional
Sumber: Ogata, 1997

2.6.2. Kontroler Integral

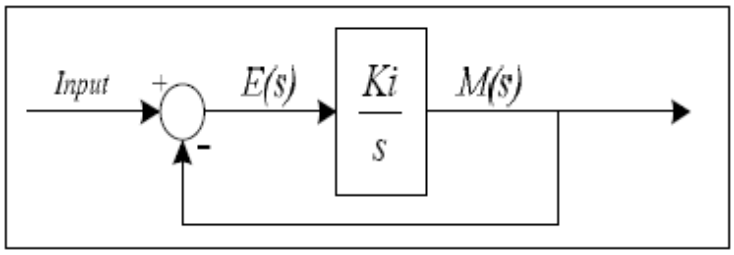
Pada kontroler dengan aksi integral, harga keluaran kontroler m(t) diubah dengan laju yang sebanding dengan sinyal kesalahan penggerak *e (t)*.

Jadi,

$$\frac{dm(t)}{dt} = Kie(t) \quad \text{..... (2-3)}$$

dengan Ki adalah konstanta integral. Jika harga *e(t)* diduakalikan, maka harga m(t) berubah dengan laju perubahan menjadi dua kali semula. Jika kesalahan penggerak nol, maka harga m(t) tetap stasioner. Aksi kontrol integral seringkali disebut kontrol *reset* yang digunakan untuk menghilangkan *error steady state* (Ogata, 1997).

Diagram blok kontroler integral ditunjukkan dalam Gambar 2.11.



Gambar 2.11 Diagram Blok Kontroler Integral
Sumber: Ogata, 1997

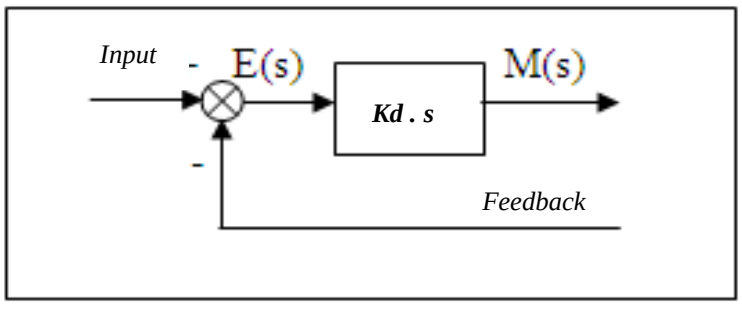
2.6.3. Kontroler Diferensial

Kontroler ini digunakan untuk memperbaiki atau mempercepat respons transien sebuah sistem kontrol dengan cara memperbesar *phase lead* terhadap penguatan kontrol dan mengurangi *phase lag* penguatan tersebut (Ogata, 1997). Kontroler diferensial tidak dapat mengeluarkan *output* bila tidak ada perubahan *input*, selain itu kontroler differensial tidak dapat digunakan untuk proses yang mengandung *noise*. Hubungan antara keluaran kontroler *m(t)* dan sinyal kesalahan penggerak *e(t)* adalah :

$$\frac{M(s)}{E(s)} = Kd.s \quad \text{..... (2-4)}$$

dengan Kd adalah konstanta diferensial yang digunakan untuk memperbaiki atau mempercepat respons transien sebuah sistem serta dapat meredam osilasi.

Gambar 2.12 menunjukkan diagram blok kontroler diferensial.



Gambar 2.12 Diagram Blok Kontroler Diferensial
Sumber: Ogata, 1997

2.6.4. Kontroler Proporsional Integral Diferensial (PID)

Gabungan aksi kontrol proporsional, integral, dan diferensial mempunyai keunggulan dapat saling menutupi kekurangan dan kelebihan dari masing-masing kontroler. Persamaan kontroler PID ini dapat dinyatakan sebagai berikut :

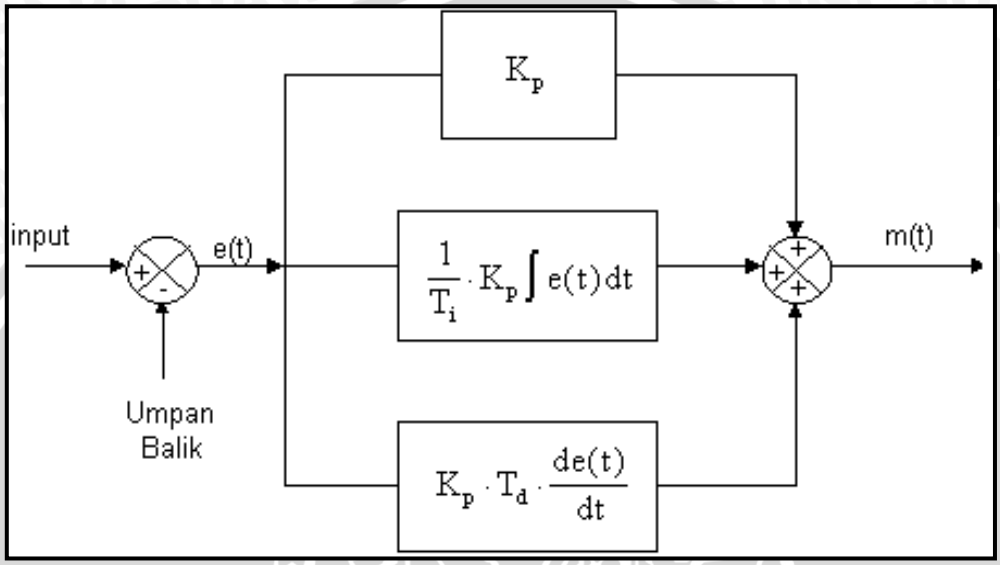
$$m(t) = Kp\,.e(t) + \frac{Kp}{Ti}.\int_0^t e(t)dt + Kp.Td\frac{de(t)}{dt} \quad \text{..... (2-5)}$$

Dalam transformasi Laplace dinyatakan sebagai berikut :

$$\frac{M(s)}{E(s)} = Kp\,(\,1 + \frac{1}{Ti\,.s} + Td.s\,) \quad \text{..... (2-6)}$$

dengan Ti adalah waktu integral, dan Td adalah waktu derivatif.

Gambar 2.13 menunjukkan diagram blok kontroler PID.



Gambar 2.13 Diagram Blok Kontroler PID

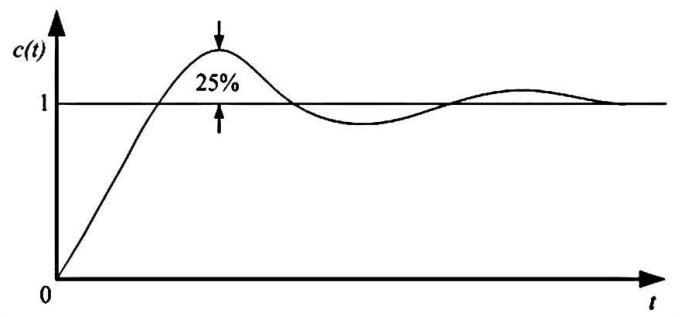
Sumber: Ogata, 1997

2.6.5. Metode Perancangan Kontroler Proporsional Integral Diferensial (PID) Menggunakan Metode Ziegler-Nichols

Ziegler dan Nichols mengemukakan aturan-aturan untuk menentukan nilai dari gain proporsional Kp, waktu integral Ti, dan waktu derivatif Td berdasarkan karakteristik respon transien dari *plant* yang diberikan. Penentuan parameter kontroler PID atau penalaan kontroler PID tersebut dapat dilakukan dengan bereksperimen dengan plan (Ogata, 1997).

Terdapat dua metode yang disebut dengan aturan penalaan Ziegler-Nichols, pada kedua metode tersebut memiliki tujuan yang sama yaitu untuk mencapai 25% *maximum overshoot* pada respon unit step.

Kurva respon unit step yang menunjukkan 25% *maximum overshoot* terlihat dalam Gambar 2.14.

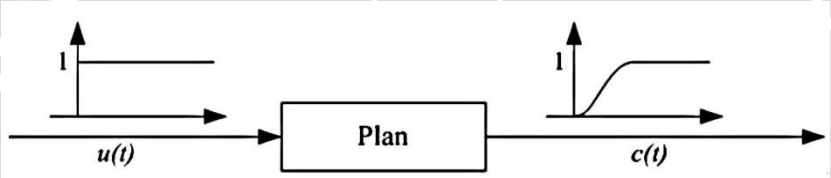


Gambar 2.14 Kurva Respon Unit Step Menunjukkan 25% *Maximum Overshoot*

Sumber: Ogata, 1997

a) Metode Pertama

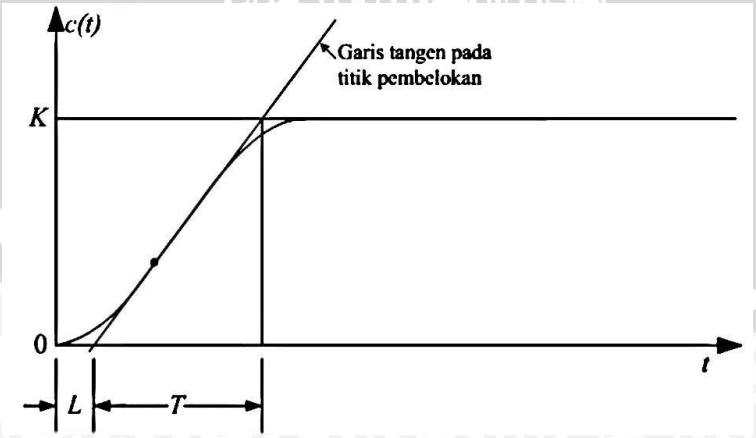
Metode pertama atau sering disebut metode kurva reaksi, respon dari plan dapat dapat diperoleh secara eksperimental dengan masukan berupa unit step, seperti ditunjukkan dalam Gambar 2.15.



Gambar 2.15 Respon Plant Terhadap Masukan Berupa Unit Step

Sumber: Ogata, 1997

Jika dalam plan tersebut terdapat integrator atau *dominan complex-conjugate poles*, maka kurva respon unit step berbentuk seperti huruf S, seperti dalam Gambar 2.16, jika respon tidak memberikan bentuk kurva S, maka metode ini tidak berlaku.(Ogata, 1997).



Gambar 2.16 Kurva yang Berbentuk S

Sumber: Ogata, 1997

Kurva berbentuk S tersebut dapat dikarakteristikkan menjadi dua konstanta yaitu waktu tunda *L* dan konstanta waktu *T*. Waktu tunda dan konstanta

waktu ditentukan dengan menggambar sebuah garis tangen pada titik pembelokan dari kurva S, dan menentukan perpotongan antara garis tangen dengan sumbu waktu *t* dan sumbu *c(t) =K*, seperti yang telah ditunjukkan dalam Gambar 2.16. Fungsi alih *C(s)/U(s)* dapat dilakukan pendekatan dengan sistem orde satu dengan persamaan sebagai berikut:

$$\frac{C(s)}{U(s)} = \frac{Ke^{-Ls}}{Ts+1} \quad \text{(2-7)}$$

Ziegler dan Nichols menyarankan untuk menentukan nilai-nilai dari Kp, Ti dan Td berdasarkan pada formula yang ditunjukkan dalam Tabel 2.3 (Ogata, 1997).

Tabel 2.3 Aturan Penalaran Ziegler-Nichols Berdasarkan Respon Unit Step dari Plan

Tipe Kontroler	K_p	T_i	T_d
P	$\frac{T}{L}$	∞	0
PI	$0.9\frac{T}{L}$	$\frac{L}{0.3}$	0
PID	$1.2\frac{T}{L}$	$2L$	$0.5L$

Sumber: Ogata, 1997

Aturan untuk metode pertama dengan persamaan sebagai berikut:

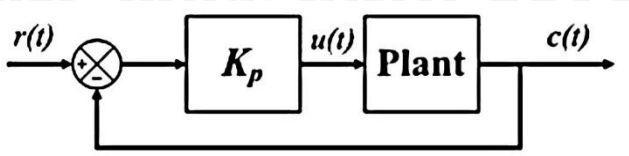
$$G_c(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$$
$$= 1.2\frac{T}{L}\left(1 + \frac{1}{2Ls} + 0.5Ls\right) \quad \text{(2-8)}$$

Gc(s) merupakan penguatan sistem yang memiliki nilai gain proporsional Kp, waktu integral Ti, dan waktu derivatif Td.

b) Metode Kedua

Dalam metode kedua Ziegler-Nichols, mula-mula yang dilakukan adalah membuat Ti = ∞ dan Td = 0. Kemudian hanya dengan menggunakan tindakan kontrol proporsional, harga ditingkatkan dari nol ke suatu nilai kritis *K*cr, disini mula-mula keluaran memiliki osilasi yang berkesinambungan (Jika keluaran tidak memiliki osilasi berkesinambungan untuk nilai Kp manapun yang telah diambil, maka metode ini tidak berlaku). Dari keluaran yang berosilasi secara berkesinambungan, penguatan kritis *K*cr dan periode *P*cr dapat ditentukan.

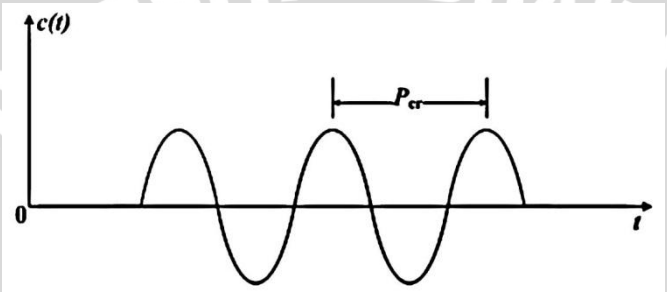
Diagram blok sistem loop tertutup dengan kontroler proporsional dapat dilihat dalam Gambar 2.17.



Gambar 2.17 Sistem Loop Tertutup dengan Kontroler Proporsional

Sumber: Ogata, 1997

Osilasi berkesinambungan dengan periode Pcr dapat dilihat dalam Gambar 2.18.



Gambar 2.18 Osilasi Berkesinambungan dengan Periode Pcr

Sumber: Ogata, 1997

Ziegler dan Nichols menyarankan penyetelan nilai parameter Kp, Ti, dan Td berdasarkan rumus yang diperlihatkan dalam Tabel 2.4 (Ogata, 1997).

Tabel 2.4 Aturan Dasar Ziegler-Nichols Berdasarkan *Critical Gain* Kcr dan *Critical Period* Pcr

Tipe Kontroler	K_p	T_i	T_d
P	$0.5\ Kcr$	∞	0
PI	$0.45\ Kcr$	$\frac{1}{1.2} Pcr$	0
PID	$0.6\ Kcr$	$0.5\ Pcr$	$0.125\ Pcr$

Sumber: Ogata, 1997

Aturan untuk metode kedua dengan persamaan sebagai berikut:

$$G_c(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right)$$
$$= 0.6K_{cr}\left(1 + \frac{1}{0.5P_{cr}s} + 0.125P_{cr}s\right) \text{.....(2-9)}$$

Gc(s) merupakan penguatan sistem yang memiliki nilai gain proporsional Kp, waktu integral Ti, dan waktu derivatif Td.

BAB III
METODOLOGI PENELITIAN

Penyusunan laporan ini didasarkan pada masalah yang bersifat aplikatif, yaitu perencanaan dan perealisasian alat agar dapat bekerja sesuai dengan yang direncanakan dengan mengacu pada rumusan masalah. Langkah-langkah yang perlu dilakukan untuk merealisasikan alat yang dirancang adalah studi literatur, penentuan spesifikasi alat, perancangan dan pembuatan alat, pengujian alat, dan pengambilan kesimpulan.

3.1. Studi Literatur

Studi literatur dilakukan untuk mempelajari teori penunjang sistem yang dibutuhkan dalam perencanaan dan pembuatan alat. Teori yang diperlukan antara lain berkaitan dengan protokol komunikasi serial I2C dan SPI, ATMega16, ATMega8, LCD.

3.2. Penentuan Spesifikasi Alat

Spesifikasi alat secara global ditetapkan terlebih dahulu sebagai acuan dalam perancangan selanjutnya. Spesifikasi alat yang direncanakan yaitu :

1) Jumlah mikrokontroller adalah 1 Master dan 3 Slave.
2) Protokol komunikasi yang dikomparasi adalah I2C dan SPI..
3) Mikrokontroler Master yang digunakan adalah AVR ATMega16. Sedangkan 3 mikrokontroller Slave adalah AVR ATMega8.
4) LCD digunakan untuk menampilkan data pembacaan sensor.
5) Catu daya menggunakan AKI DC 12 V.
6) Alat gerak atau aktuator robot adalah motor DC NISCA N4775 12V 1930rpm.

3.3. Perancangan dan Pembuatan Alat

Perancangan dan pembuatan alat dalam penelitian ini dibagi menjadi empat bagian, yaitu mekanik, *hardware*, sistem kontrol, dan *software*.

3.3.1. Perancangan dan Pembuatan Mekanik

Perancangan dan pembuatan mekanik robot *management* sampah meliputi tahapan-tahapan sebagai berikut:

1) Penentuan ukuran dimensi robot.
2) Penentuan peletakan posisi motor, roda, dan sensor garis.
3) Pembuatan gambar perspektif 3 dimensi robot.
4) Pembuatan mekanik robot berdasarkan pada perancangan.

3.3.2. Perancangan dan Pembuatan Perangkat Keras (*Hardware*)

Secara garis besar perancangan perangkat keras (*Hardware*) dibagi dalam beberapa tahap sebagai berikut:

1) Penentuan spesifikasi alat.
2) Pembuatan blok diagram keseluruhan sistem.
3) Penentuan komponen yang akan digunakan.
4) Merakit perangkat keras masing-masing blok.

3.3.3. Perancangan Sistem Kontroler PID pada Robot *Management* Sampah

Perancangan sistem kontroler PID pada robot management sampah meliputi tahapan-tahapan sebagai berikut:

1) Pembuatan blok diagram kontrol sistem.
2) Penentuan set point posisi robot.
3) Penentuan nilai Kcr dan Pcr.
4) Perhitungan nilai Kp, Ki, Kd.

3.3.4. Perancangan dan Pembuatan Perangkat Lunak (*Software*)

Penyusunan perangkat lunak (*software*) digunakan untuk mengendalikan dan mengatur kinerja dari robot ini. Desain dan parameter yang telah dirancang kemudian diterapkan ke dalam mikrokontroller.

3.4. Pengujian dan Analisis

Untuk menganalisis kinerja alat apakah sesuai dengan yang direncanakan maka dilakukan pengujian sistem. Pengujian dilakukan pada timer

mikrokontroller, waktu eksekusi, serta performa robot saat menggunakan sistem I2C maupun SPI.

3.3.1. Pengujian Timer Mikrokontroller

Untuk pengujian timer mikrokontroller, pengujian dilakukan dengan interupsi *mode normal overflow*. Mikrokontroller diprogram agar setiap terjadi interupsi timer overflow maka PORTA akan di togle. Pengujian ini bertujuan untuk menganalisis apakah timer mikrokontroller tersebut akurat.

3.3.2. Pengujian Waktu Eksekusi

Pengujian waktu eksekusi dilakukan dengan cara melihat nilai register TCNT setelah mengeksekusi suatu perintah. Pengujian ini bertujuan untuk menganalisis waktu yang dibutuhkan untuk mengirim maupun menerima data pada protokol I2C dan SPI.

3.3.3. Pengujian Performansi Robot

Pengujian performansi robot dilakukan dengan membandingkan antara sistem yang menggunakan protokol I2C dan sistem yang menggunakan protokol SPI terhadap perfroma robot apabila diberi gangguan pada bentuk lintasan dan variasi beban robot. Data yang di ambil adalah nilai posisi sensor garis. Data tersebut diambil dengan cara komunikasi serial UART antara mikrokontroller master dengan PC/Laptop.

3.5. Pengambilan Kesimpulan dan Saran

Kesimpulan didapat berdasarkan hasil perealisasian komparasi sistem komunikasi serial multipoint pada robot management sampah menggunakan I2C dan SPI. Beberapa hal hasil pengujian disampaikan dalam kesimpulan disertai realita yang disusun secara berurutan.

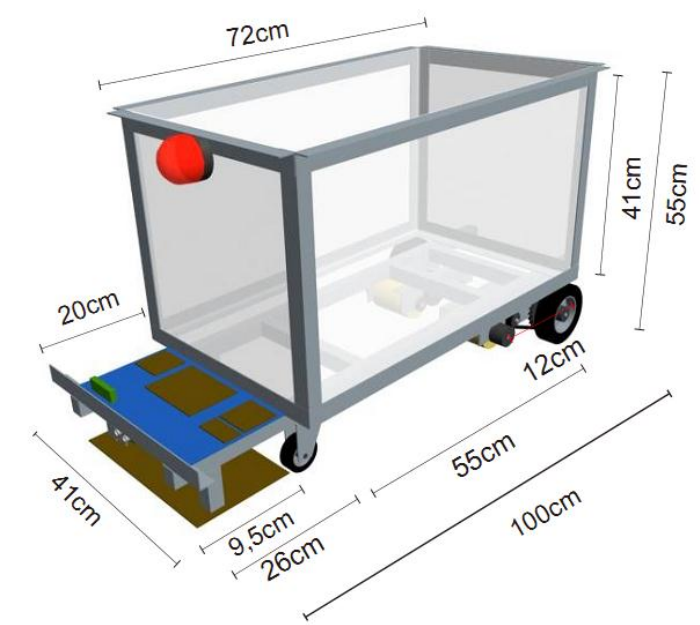
BAB IV
PERANCANGAN DAN PEMBUATAN ALAT

Perancangan robot dilakukan secara bertahap sehingga akan memudahkan dalam analisis pada setiap bloknya maupun secara keseluruhan. Perancangan ini terdiri dari:

- Perancangan mekanik robot.
- Perancangan perangkat keras (*hardware*).
- Perancangan sistem kontroler PID.
- Perancangan perangkat lunak.

4.1.Perancangan Mekanik Robot

Sistem mekanik yang baik, mendukung pergerakan robot menjadi lebih baik, oleh karena itu perancangan mekanik dalam hal ini bodi dan rangka robot haruslah proporsional dengan panjang dan lebar serta tinggi dari robot. Dimensi robot adalah sebagai berikut :

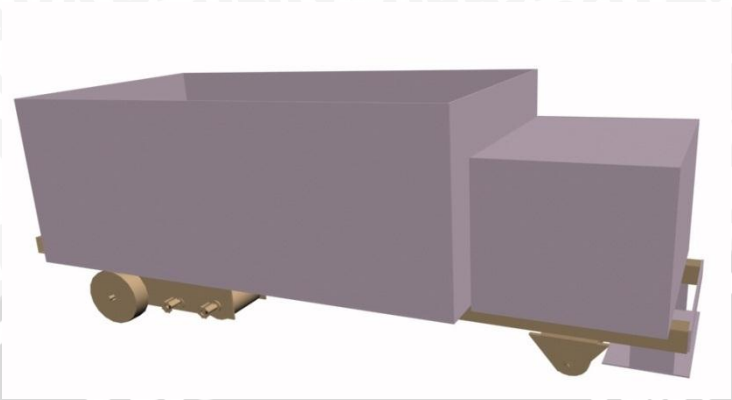


Gambar 4.1. Perancangan Mekanik Robot Management Sampah

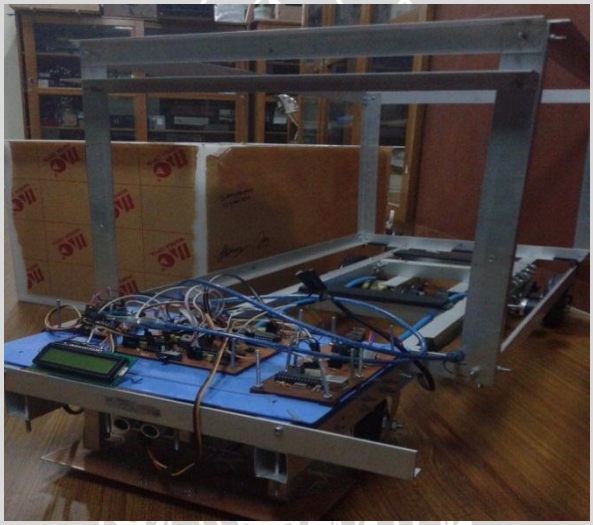
Keterangan:

- Sensor diletakan di depan.
- Sisa space di belakang sebagai pemicu ke tempat sampah.
- Baterai dan komponen elektrik diletakkan di kepala robot.

Gambar perspektif Robot ditunjukkan dalam Gambar 4.2.



Gambar 4.2. Perspektif robot tampak samping



Gambar 4.3. Foto robot *management* sampah

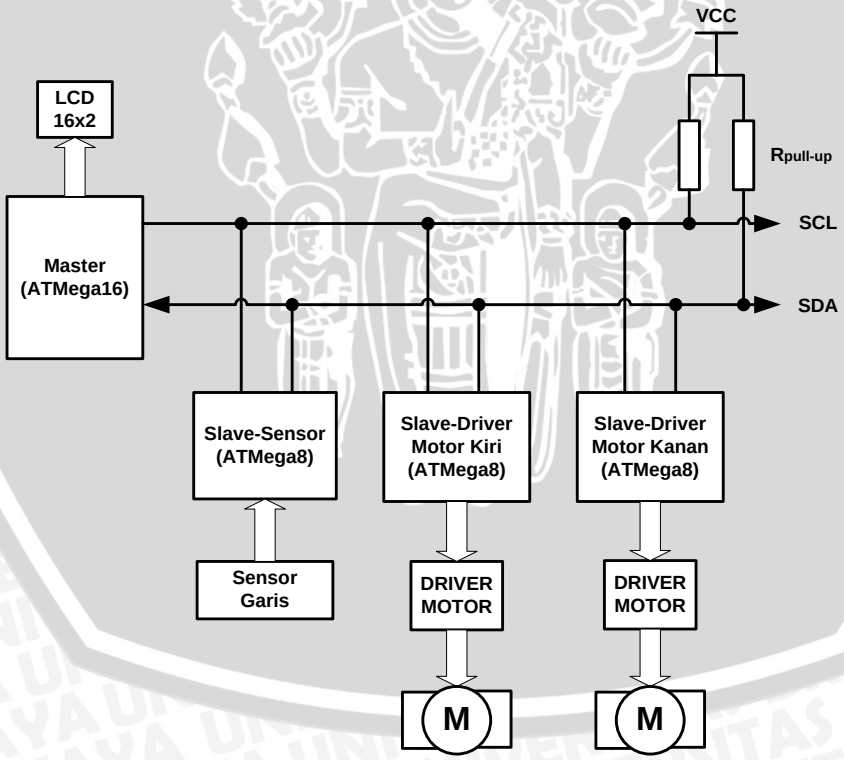
Gambar 4.3 menunjukkan tentang robot *management* sampah. Dari gambar tersebut terlihat adanya kerangka box yang berfungsi sebagai tempat box sampah, rangkaian elektrik diletakkan dibagian kepala robot yang nanti akan diberi penutup agar rangkaian tersebut tetap aman, sensor photodioda diletakkan dibagian depan bawah robot.

4.2.Perancangan Perangkat Keras (*Hardware*)

Dalam perancangan perangkat keras terdapat lima bagian perancangan. Yaitu diagram blok secara keseluruhan. Perancangan sistem catu daya sebagai penyuplai tegangan. Perancangan minimum sistem mikrokontroller ATmega16 sebagai *master*. Perancangan minimum sistem mikrokontroller ATmega8 sebagai *slave*. Perancangan antarmuka serial *multipoint* menggunakan I2C dan SPI.

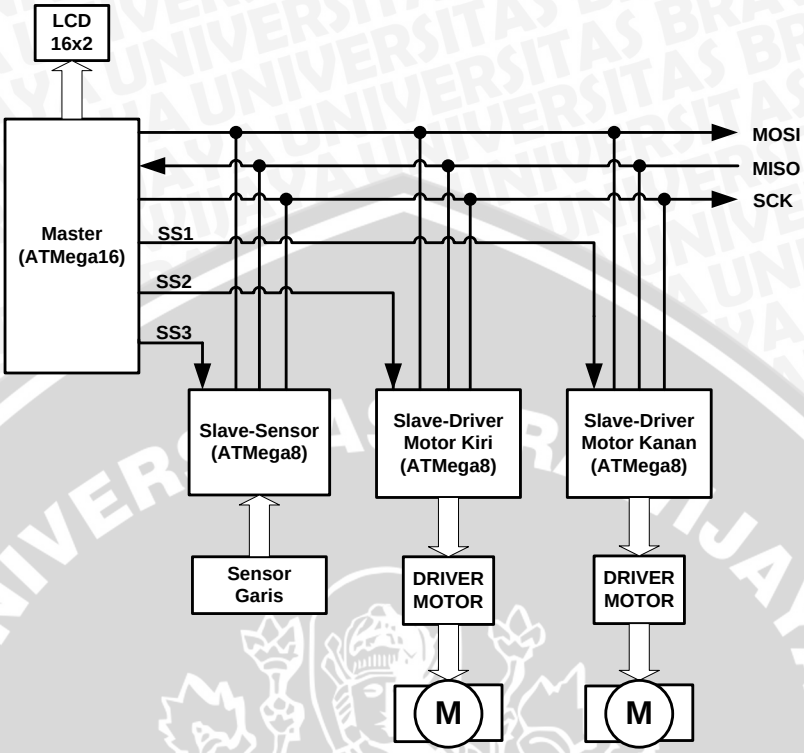
4.2.1. Diagram Blok

Secara garis besar, diagram blok perancangan *hardware* sistem dibagi menjadi dua yaitu diagram blok sistem menggunakan I2C dan diagram blok sistem menggunakan SPI. Diagram blok sistem menggunakan I2C ditunjukkan dalam Gambar 4.4.



Gambar 4.4. Diagram Blok Sistem menggunakan I2C

Sedangkan untuk diagram blok sistem menggunakan SPI ditunjukkan dalam Gambar 4.5.



Gambar 4.5. Diagram Blok Sistem menggunakan SPI

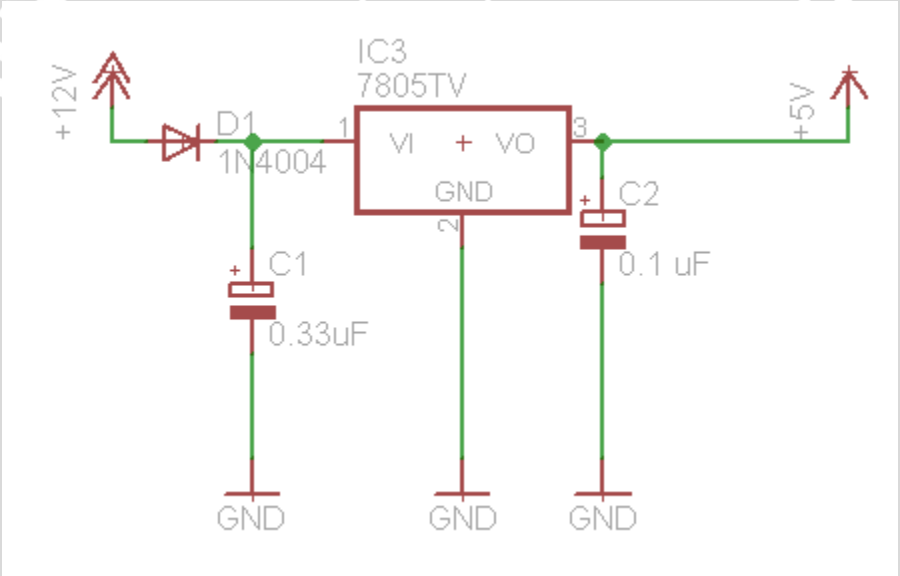
Dalam penelitian ini terdapat bagian perancangan dan pembuatan perangkat keras, yaitu:

1) Sensor garis digunakan untuk membaca garis dengan menggunakan sensor photodioda.
2) Mikrokontroler Slave-Sensor AVR ATMega8 sebagai pengolah pembacaan sensor.
3) Mikrokontroler Master AVR ATMega16 sebagai pengolah data dan pusat kendali sistem dari robot management sampah.
4) LCD digunakan untuk menampilkan pembacaan sensor atau data.
5) 2 buah Mikrokontroler ATMega8 Slave-Driver Motor Kiri dan Kanan sebagai pengendali Driver Motor Kiri dan Kanan.
6) 2 buah *Driver motor* digunakan untuk mengendalikan sinyal dari mikrokontroler untuk menggerakkan motor.
7) 2 buah Motor DC NISCA N4775 12V 1930rpm sebagai alat gerak atau aktuator robot.

4.2.2. Perancangan Catu Daya Sistem

Robot ini Menggunakan dua jenis catu daya. Catu daya 12V untuk motor yang bersumber dari AKI 12V. Serta catu daya 5V untuk rangkaian minimum sistem mikrokontroller ATmega16 dan ATmega8 yang bersumber dari baterai lipo (*lithium polimer*) 11,1V.

Pada perancangan digunakan catu daya sebesar 5V yang diperoleh dari rangkaian *Fixed Output Regulator* pada datasheet LM7805. IC 78xx mempunyai tiga kaki, satu V_{in}, satu untuk V_{out} dan satu untuk GND (Blocher, 2003). Skema rangkaian catu daya ditunjukkan dalam Gambar 4.6.

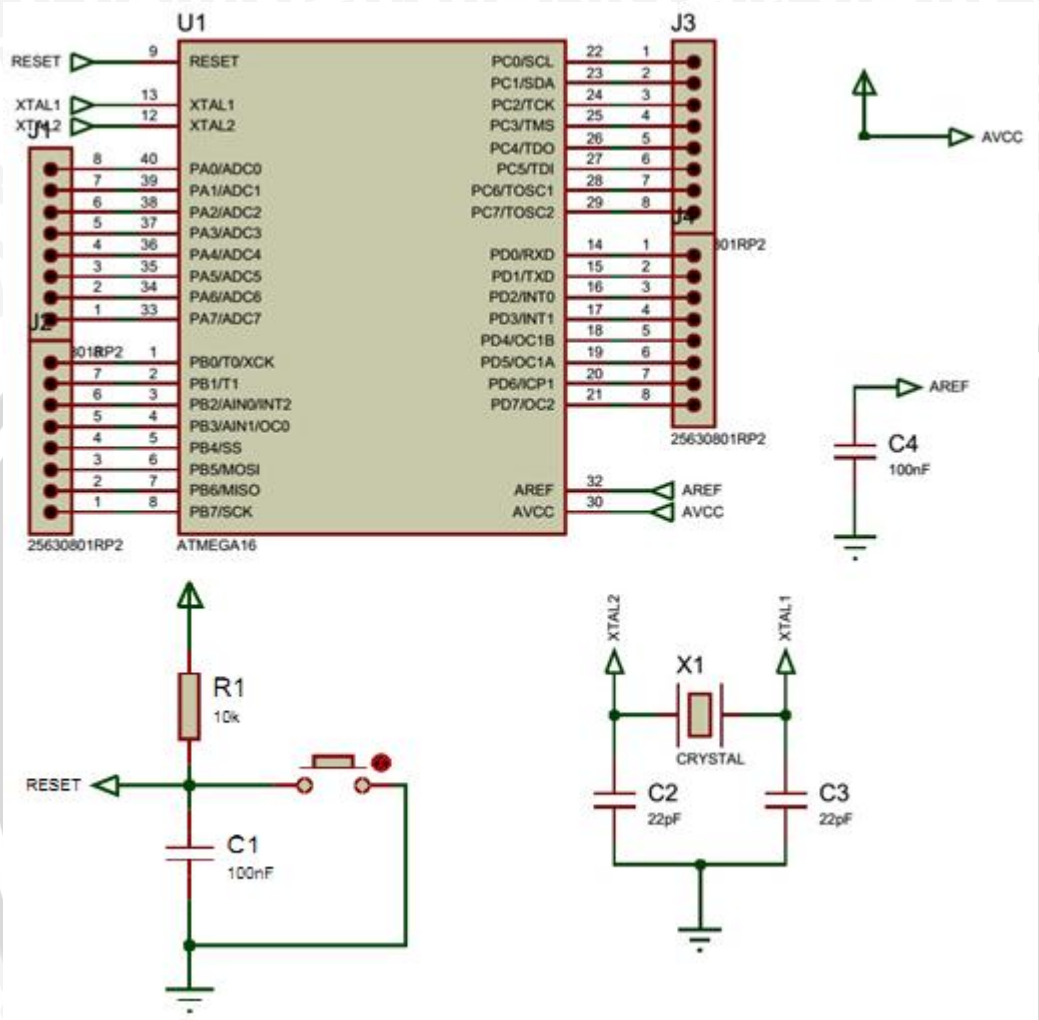


Gambar 4.6. Rangkaian Catu 5V

4.2.3. Perancangan Minimum Sistem Mikrokontroller ATmega16

Pada robot ini digunakan mikrokontroler ATmega16 sebagai *master*. Mikrokontroler *master* menerima data berupa posisi robot terhadap garis dari mikrokontroler *slave* sensor (ATmega8), data tersebut di proses, kemudian hasil dari pemrosesan data berupa PWM motor yang akan dikirimkan ke mikrokontroler *slave* (ATmega8) motor kanan dan Mikrokontroler *slave*

(ATmega8) motor kiri. Berikut konfigurasi dari pin I/O dari mikrokontroler ATmega16 (mikrokontroler *master)* ditunjukkan dalam Gambar 4.7.



Gambar 4.7. Rangkaian Minimum Sistem Mikrokontrole*r* ATmega16

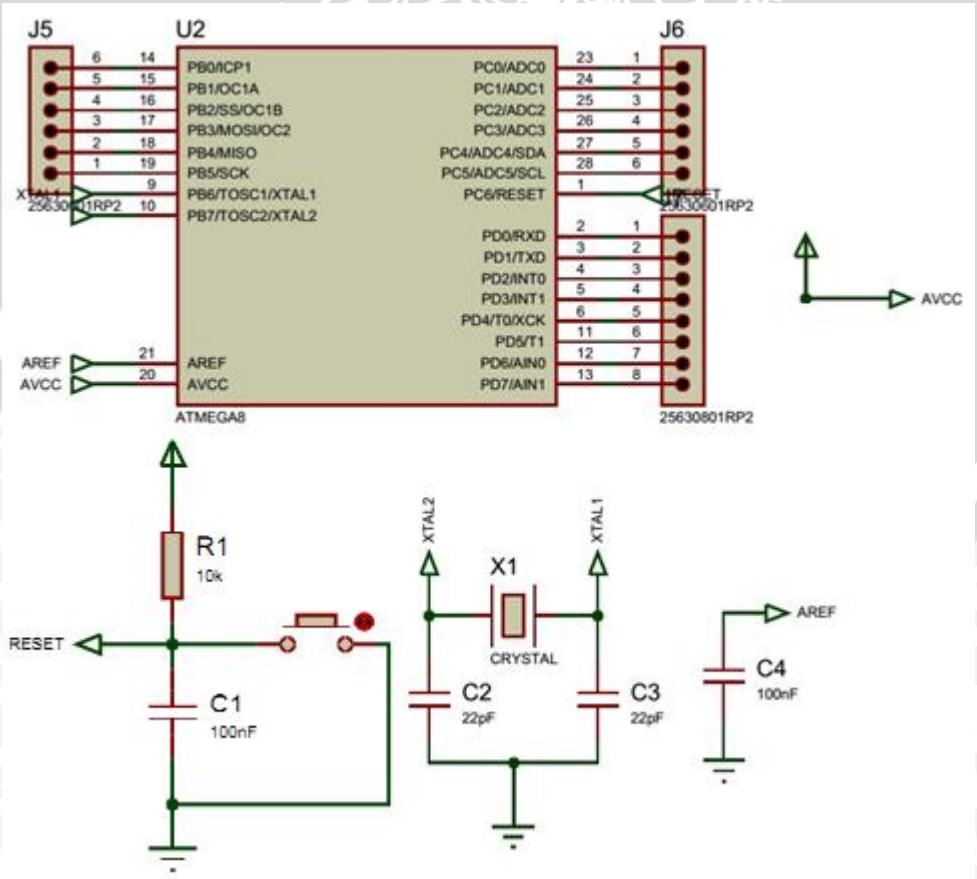
Rangkaian minimum sistem mikrokontoller master yang digunakan mempunyai 4 port, dengan 32 jalur I/O yang dapat di program menjadi masukan atau keluaran. Pin dan kaki IC yang digunakan antara lain:

- Pin B0 : Sebagai pin keluaran buzzer.
- Pin B1 : Sebagai pin masukan pulsa sensor PING.
- Pin B5(MOSI) : Sebagai pin keluaran data SPI.
- Pin B6(MISO) : Sebagai pin masukan data SPI.
- Pin B7(SCK) : Sebagai pin keluaran clock SPI.

- Pin C0(SCL) : Sebagai pin keluaran clock I2C.
- Pin C1(SDA) : Sebagai pin masukan maupun keluaran data I2C.
- Pin Reset : Dihubungkan dengan rangkaian switch Reset.
- Pin XTAL1 : Dihubungkan dengan kaki osilator 11059200 Hz.
- Pin XTAL2 : Dihubungkan dengan kaki osilator 11059200 Hz.
- Pin AREF : Dihubungkan dengan rangkaian AREF.
- Pin AVCC : Dihubungkan dengan rangkaian AVCC.

4.2.4. Perancangan Minimum Sistem Mikrokontroller ATmega8

Pada robot ini juga digunakan 3 mikrokontroler ATmega8 sebagai *slave*. Fungsi dari 3 mikrokontroller tersebut antara lain sebagai pengolah data sensor photodiode, pengatur arah dan kecepatan putaran motor kanan, dan pengatur arah dan kecepatan putaran motor kiri. Konfigurasi kaki I/O dari mikrokontroler *slave* (ATmega8) ditunjukkan dalam Gambar 4.8.



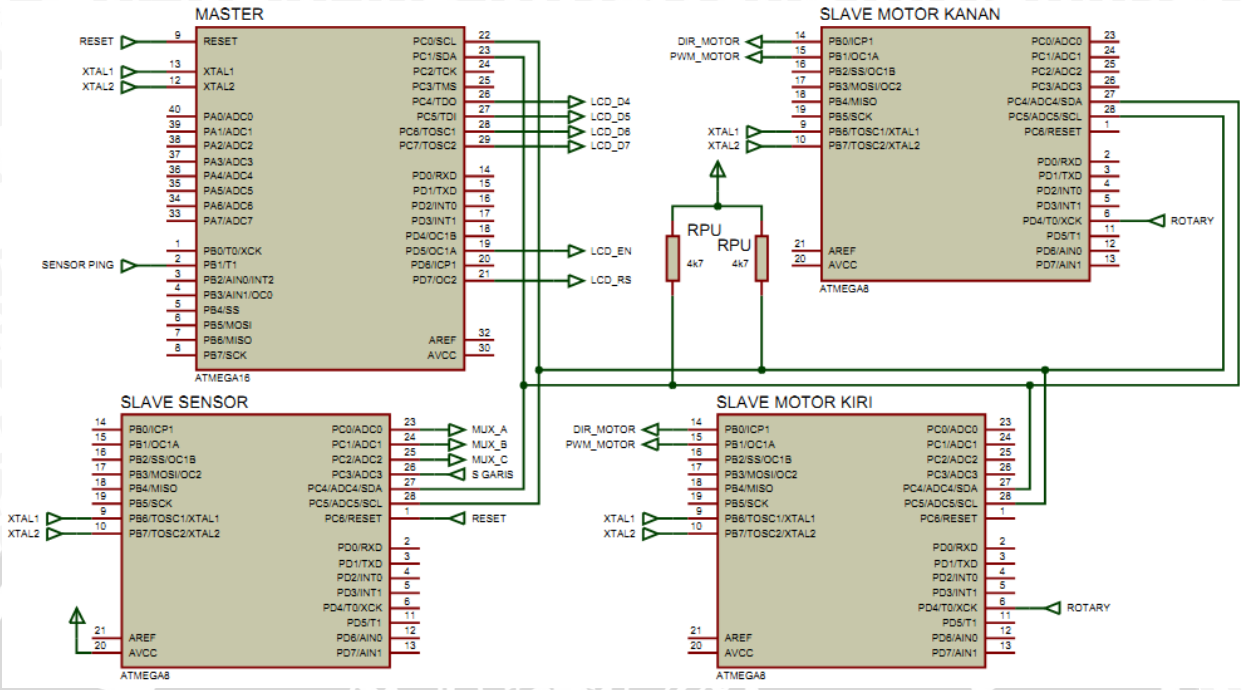
Gambar 4.8. Rangkaian Minimum Sistem Mikrokontroler ATmega8

Mikrokontroler *slave* (ATMega8) mempunyai 3 port, dengan 21 jalur I/O. Pin yang digunakan antara lain:

- Pin B0 : Sebagai pin keluaran ke driver motor.
- Pin B1 : Sebagai pin keluaran ke driver motor.
- Pin B3(MOSI) : Sebagai pin masukan data SPI.
- Pin B4(MISO) : Sebagai pin keluaran data SPI.
- Pin B5(SCK) : Sebagai pin masukan clock SPI.
- Pin C5(SCL) : Sebagai pin keluaran clock I2C.
- Pin C4(SDA) : Sebagai pin masukan maupun keluaran data I2C.
- Pin D4 : Sebagai pin masukan rotary encoder.
- Pin Reset : Dihubungkan dengan rangkaian switch Reset.
- Pin XTAL1 : Dihubungkan dengan kaki osilator 11059200 Hz.
- Pin XTAL2 : Dihubungkan dengan kaki osilator 11059200 Hz.
- Pin AREF : Dihubungkan dengan rangkaian AREF.
- Pin AVCC : Dihubungkan dengan rangkaian AVCC.

4.2.5. Perancangan Rangkaian Antarmuka Serial *Multipoint* menggunakan I2C

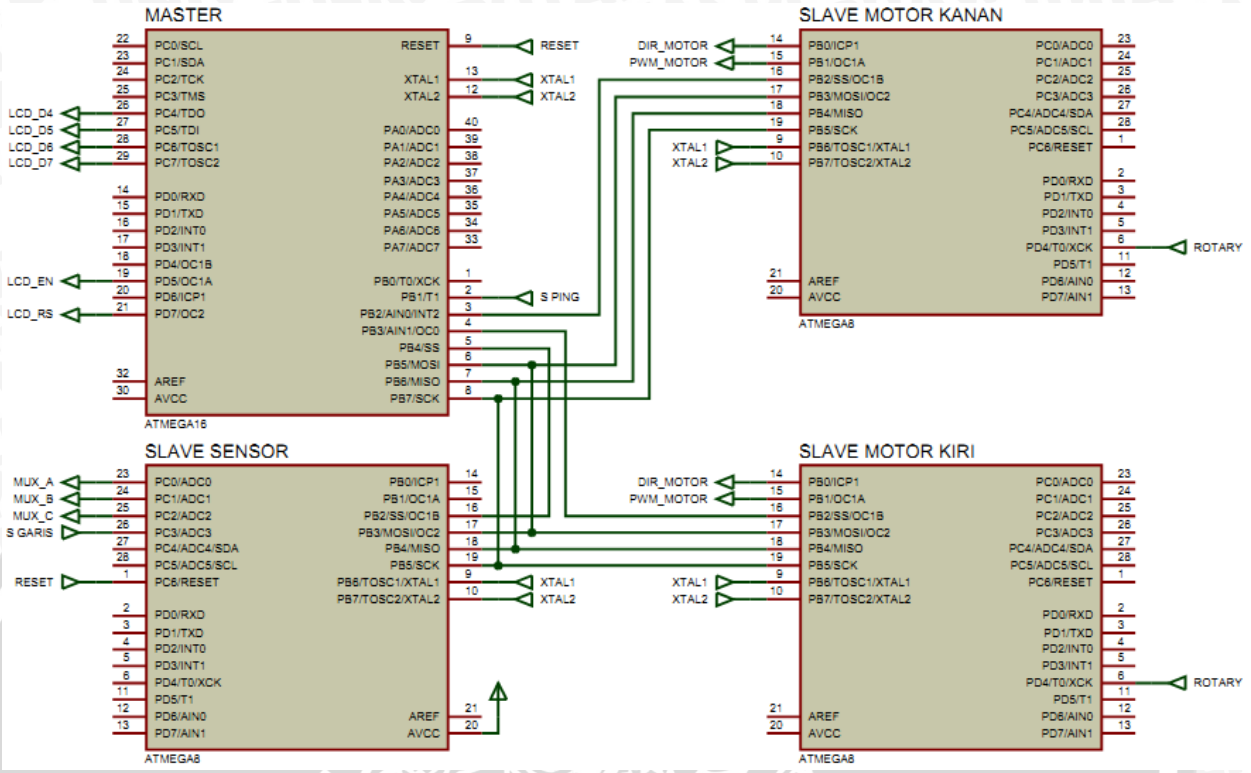
Pada perancangan rangkaian antarmuka serial *multipoint* menggunakan I2C terdapat 2 jalur bus yaitu SDA (*Serial Data*) dan SCL (*Serial Clock*). Masing-masing dari jalur tersebut di beri Resistor *Pull-Up* yang nilainya ada pada datasheet. Rangkaian antarmuka serial *multipoint* menggunakan I2C ditunjukkan dalam Gambar 4.9.



Gambar 4.9. Rangkaian Antarmuka Serial *Multipoint* menggunakan I2C

4.2.6. Perancangan Rangkaian Antarmuka Serial *Multipoint* menggunakan SPI

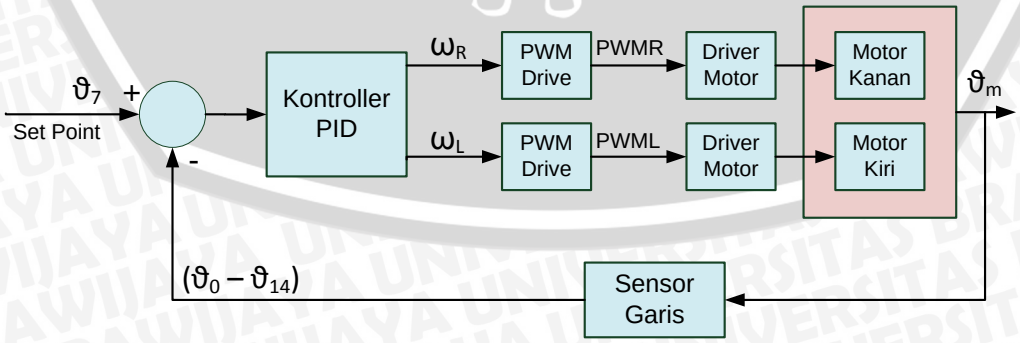
Pada perancangan rangkaian antarmuka serial *multipoint* menggunakan SPI terdapat 3 jalur bus yaitu MOSI (*Master Output Slave Input*), MISO (*Master Input Slave Output*), dan SCK (*Serial Clock*). Selain itu ada jalur SS (*Slave Selec*t). Pin SS ada pada setiap *slave*. Fungsi dari pin ini adalah untuk memilih *device slave* mana yang akan diajak berkomunikasi dengan *master*. Sehingga *master* membutuhkan 3 pin I/O untuk 3 *slave*. Rangkaian antarmuka serial *multipoint* menggunakan SPI ditunjukkan dalam Gambar 4.10.



Gambar 4.10. Rangkaian Antarmuka Serial *Multipoint* menggunakan SPI

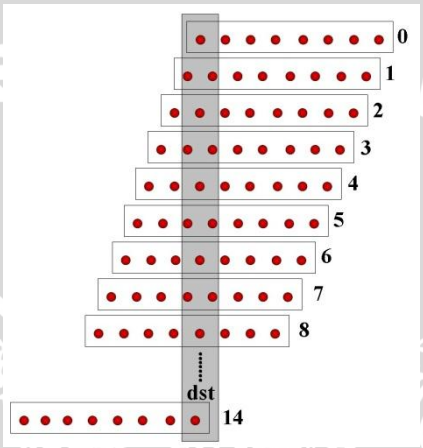
4.3.Perancangan Sistem Tuning Kontroler PID Menggunakan Metode Kedua Ziegler-Nichols pada Robot *Management* Sampah

Pada sistem kontroler PID ini dibutuhkan masukan sebagai umpan balik yang diproses oleh rumus PID. Umpan balik diperoleh dari sensor garis yang diletakkan di depan untuk mengukur posisi robot. Gambar 4.11 menunjukkan diagram kontrol PID dari Robot *Management* Sampah.



Gambar 4.11. Diagram Blok Kontrol PID Robot *Management* Sampah

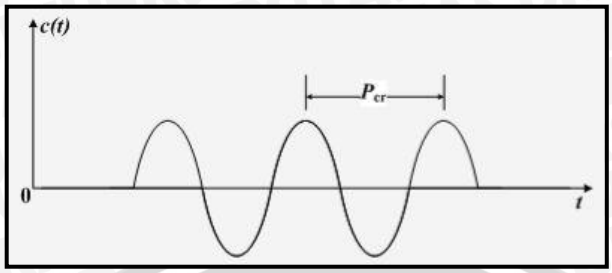
Untuk melakukan proses pengontrolan dengan PID diperlukan suatu nilai error yaitu selisih antara *set point* dengan posisi robot yang terukur sensor garis. Nilai *set point* (posisi sensor robot yang diinginkan) yang akan dipakai harus ditentukan terlebih dahulu sebelum mencari grafik osilasi berkesinambungan. *Set poin*t pada sistem ini adalah posisi robot yang berada pada tengah garis yang bernilai 7. Nilai posisi sensor robot *management* sampah ditunjukkan dalam Gambar 4.12.



Gambar 4.12 Nilai Posisi Sensor Robot *Management* Sampah

Kontroler PID memiliki tiga parameter yang sangat berpengaruh pada kerja kontroler ini yaitu konstanta proporsional (Kp), konstanta integral (Ki), dan konstanta diferensial (Kd). Oleh karena itu dilakukan *tuning* eksperimen untuk mendapatkan nilai Kp, Ki, dan Kd yang tepat sehingga kontroler dapat bekerja secara optimal. Pada perancangan kontroler PID robot *management* sampah ini, menggunakan *tuning* parameter Ziegler-Nichols metode kedua.

Pada proses *tuning* kontrol PID dengan menggunakan metode osilasi Ziegler Nichols dimulai dengan memberikan nilai 0 pada parameter Ki dan Kd. Kemudian nilai Kp dinaikkan sedikit demi sedikit hingga didapatkan grafik pergerakan posisi sensor yang berkesinambungan. Kesinambungan yang dimaksud adalah saat grafik memiliki amplitudo yang sama pada setiap periodenya seperti yang ditunjukkan dalam Gambar 4.13.



Gambar 4.13 Osilasi Berkesinambungan dengan Periode Pcr

Setelah didapatkan grafik yang berkesinambungan langkah selanjutnya adalah menghitung nilai Kcr dan Pcr. Kcr adalah nilai Kp saat terjadi osilasi berkesinambungan sedangkan Pcr adalah periode kesinambungan dari grafik. Setelah didapatkan Kcr dan Pcr langkah selanjutnya adalah menghitung nilai Kp, Ti, dan Td sesuai dengan aturan Ziegler-Nichols.

$$\text{Kp} = 0.6\ x\ \text{Kcr}$$

$$\text{Ti} = 0.5\ \text{x}\ \text{Pcr}$$

$$\text{Td} = 0.125\ \text{x}\ \text{Pcr}$$

Nilai Ki dan Kd didapatkan dengan menggunakan perhitungan sebagai berikut.

$$\text{Ki} = \frac{\text{Kp}}{\text{Ti}}$$

$$\text{Kd} = \text{Kp}\ \text{x}\ \text{Td}$$

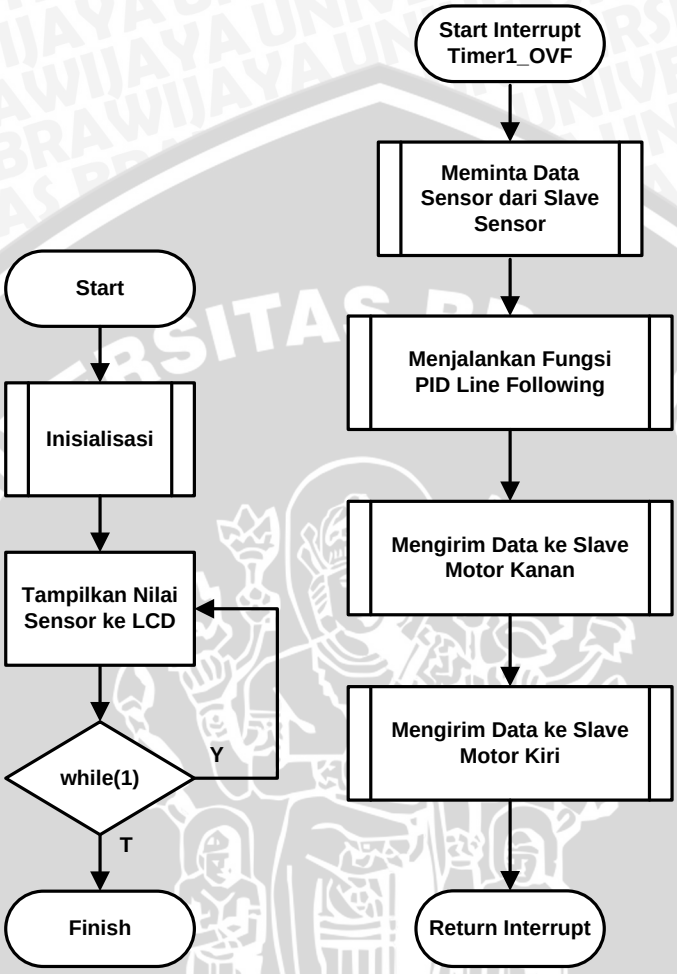
4.4.Perancangan Perangkat Lunak

Dalam skripsi ini perancangan perangkat lunak terdiri atas tiga bagian, yaitu perancangan perangkat lunak untuk mikrokontroler *master*, mikrokontroler *slave sensor*, dan mikrokontroler *slave motor*. Perancangan perangkat lunak *master* untuk mengolah masukan dari *slave sensor* dengan menggunakan sistem kontrol PID kemudian mengirim hasil keluaran ke *slave motor*. perancangan perangkat lunak *slave sensor* untuk membaca dan mengolah data sensor garis *photodioda*. Sedangkan perancangan perangkat lunak *slave motor* mengolah data dari *master* agar menjadi sinyal keluaran ke driver motor.

4.4.1. Perancangan Program Mikrokontroler *Master*

Program utama mikrokontroler *master* dirancang untuk melakukan proses antarmuka dengan *slave sensor*, mengolah data sensor dengan sistem kontrol PID,

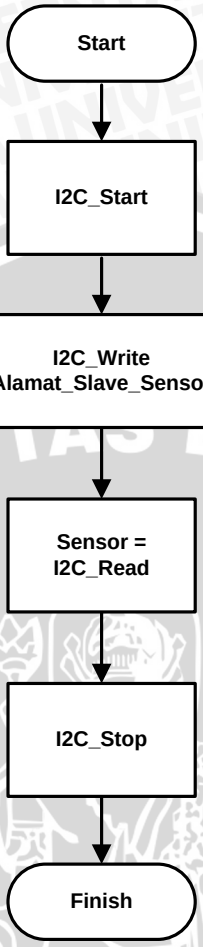
dan mengirim hasil keluaran berupa kecepatan dan direction ke mikrokontroler *slave motor*. Diagram alir program utama *master* ditunjukkan dalam Gambar 4.14



Gambar 4.14 Diagram Alir Program Utama *Master*

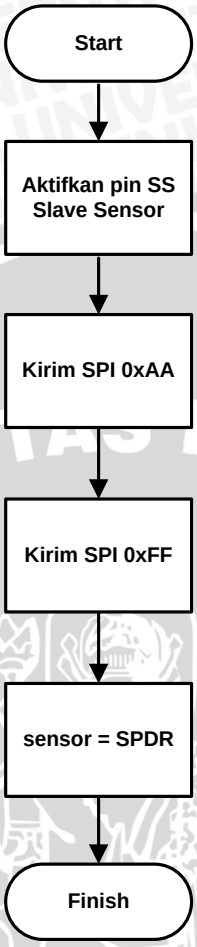
4.4.2. Diagram Alir Proses Meminta Data Sensor dari *Slave Sensor*

Diagram alir ini berfungsi untuk proses mikrokontroller *master* meminta data ke *slave sensor*. Diagram alir proses meminta data sensor dari *slave sensor* dengan protokol I2C ditunjukkan dalam Gambar 4.15.



Gambar 4.15 Diagram Alir Proses Meminta Data Sensor dari *Slave Sensor* dengan Protokol I2C

Sedangkan untuk diagram alir proses meminta data sensor dari *slave sensor* dengan protokol SPI ditunjukkan dalam Gambar 4.16.



Gambar 4.16 Diagram Alir Proses Meminta Data Sensor dari *Slave Sensor* dengan Protokol SPI

4.4.3. Diagram Alir Fungsi PID Line Following

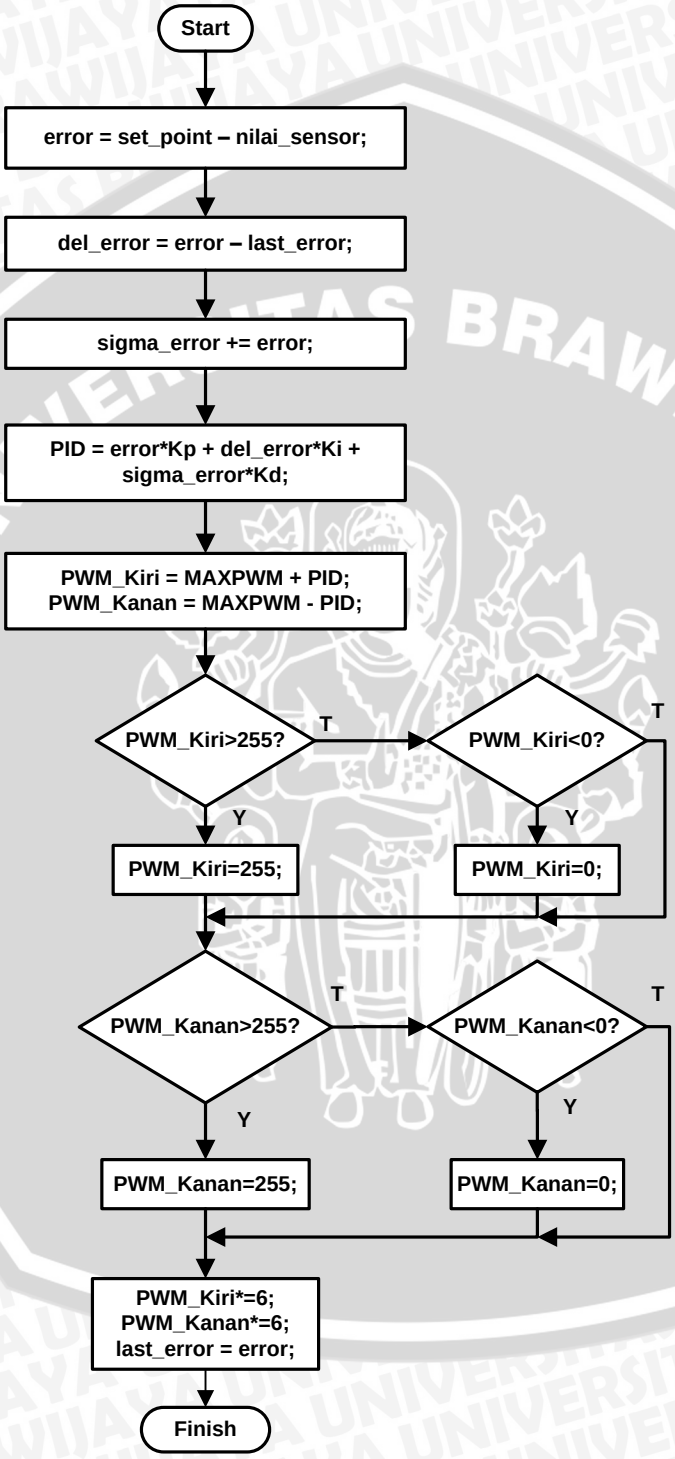
Diagram Alir perhitungan PID berisi program untuk proses perhitungan kontroler PID. PID adalah suatu perhitungan matematis yang terdiri dari proporsional, integral, dan derivative yang ditambahkan secara bersama-sama dan menghasilkan suatu nilai yang digunakan untuk mengendalikan robot. Rumusan PID secara umum adalah sebagai berikut

$$\text{Output(t)} = \text{Kp.error(t)} + \int Ki.error(t)dt + \text{Kd.}\frac{d\,error(t)}{dt}$$

Dalam aplikasi PID sebagai kontrol posisi sensor garis, rumusan tersebut didekati dengan:

$$\text{Output(t)=Kp.error(t)+(Ki.Ts).}\sum error(t)\text{+(Kd/Ts).(error(t)-error(t-1))}$$

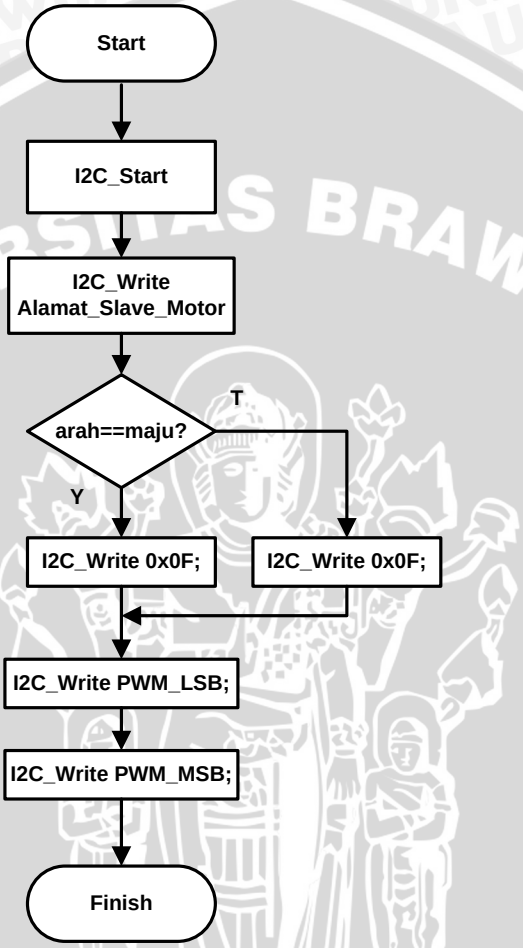
Nilai s*et point* adalah 7 yaitu nilai sensor pada saat robot berada ditengah-tengah garis. Diagram alir perhitungan PID ditunjukkan dalam Gambar 4.17.



Gambar 4.17 Diagram Alir Perhitungan PID

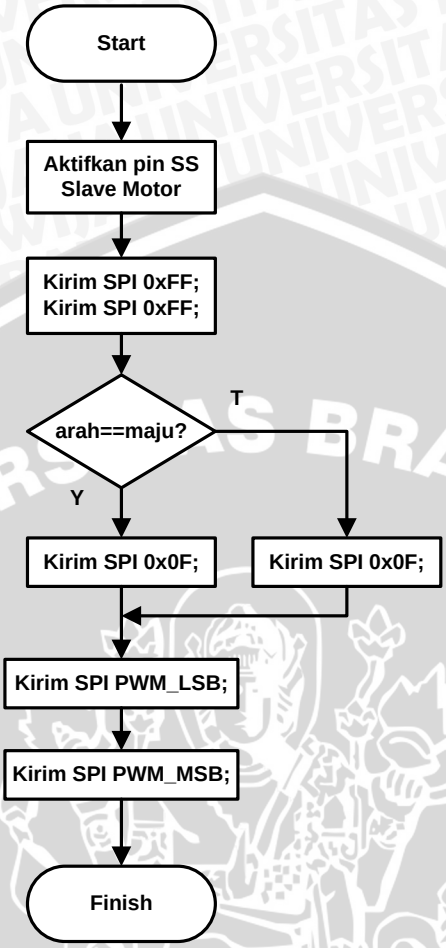
4.4.4. Diagram Alir Proses Mengirim Data ke *Slave Motor*

Diagram alir ini berfungsi untuk proses mikrokontroller master mengirim data ke slave motor. Diagram alir proses mengirim data ke slave motor dengan protokol I2C ditunjukkan dalam Gambar 4.18



Gambar 4.18 Diagram Alir Proses Mengirim Data ke *Slave Motor* dengan Protokol I2C

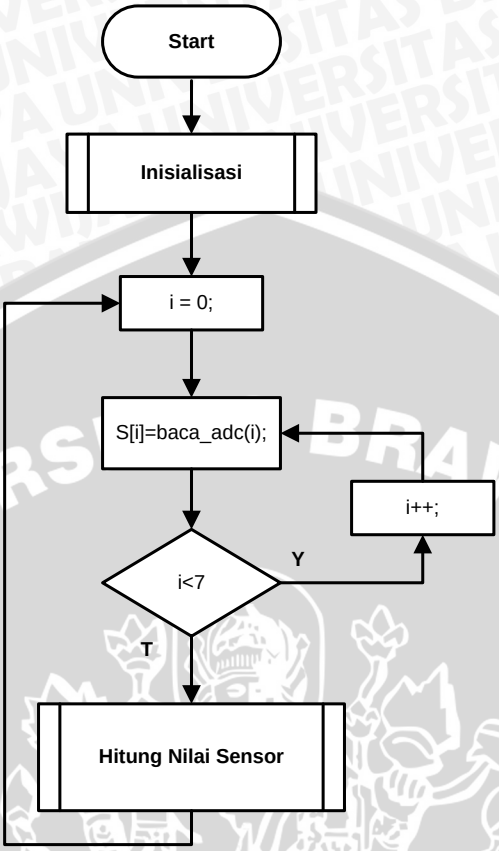
Sedangkan untuk diagram alir proses mengirim data ke *slave motor* dengan protokol SPI ditunjukkan dalam Gambar 4.19



Gambar 4.19 Diagram Alir Proses Mengirim Data ke *Slave Motor* dengan Protokol SPI

4.4.5. Perancangan Program Mikrokontroler *Slave Sensor*

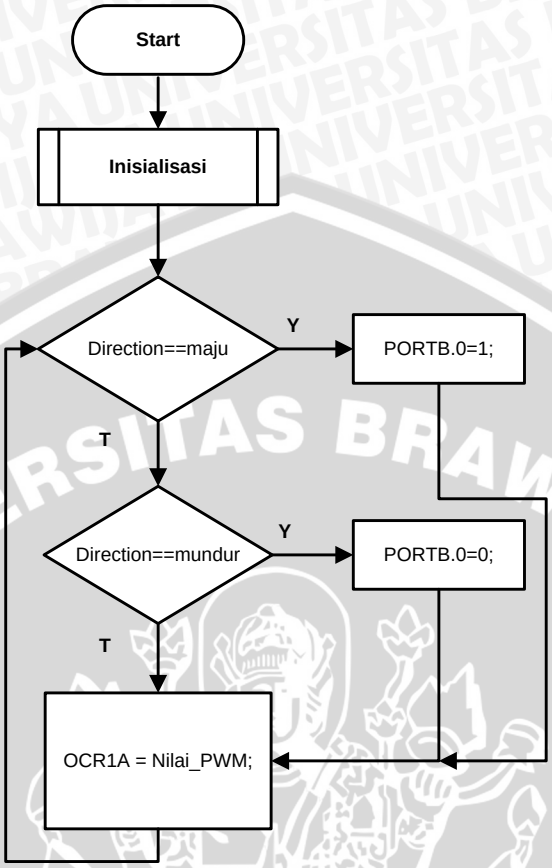
Perangkat lunak pada mikrokontroller *Slave Sensor* difungsikan sebagai pengolah data dari sensor garis melalui ADC 8-bit dan dengan multiplekser analog. Sensor garis yang digunakan adalah photodioda dan LED. Jumlah sensor photodioda yang akan digunakan sebanyak 8. *Flowchart* program mikrokontroler *slave sensor* ditunjukkan dalam Gambar 4.20.



Gambar 4.20 *Flowchart* Program Utama Mikrokontroler *Slave Sensor*

4.4.6. Perancangan Program Mikrokontroler *Slave Motor*

Perangkat lunak pada mikrokontroller *Slave Motor* kanan maupun kiri diprogram menerjemahkan data kecepatan dan *direction* motor dari *Master* untuk diolah menjadi sinyal keluaran PWM dan logika *direction* ke driver motor. *Flowchart* program mikrokontroler *slave motor* ditunjukkan dalam Gambar 4.21.



Gambar 4.21 *Flowchart* Program Utama Mikrokontroler *Slave Motor*

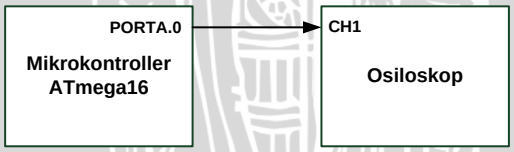
BAB V
PENGUJIAN DAN ANALISIS

Pengujian dan analisis dilakukan untuk menganalisis apakah sistem telah bekerja sesuai perancangan. Pengujian dilakukan per blok kemudian secara keseluruhan. Pengujian yang perlu dilakukan adalah sebagai berikut:

1) Pengujian *Timer* Mikrokontroler
2) Pengujian *Tuning* PID.
3) Pengujian Waktu Eksekusi pada Protokol I2C dan SPI.
4) Pengujian Respons Robot pada Protokol I2C dan SPI dengan Waktu Sampling yang Bervariasi.
5) Pengujian Respons Robot pada Protokol I2C dan SPI dengan Beban Robot yang Bervariasi.

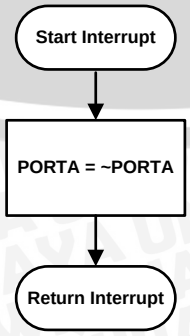
5.1. Pengujian *Timer* Mikrokontroler

Pengujian ini dilakukan dengan tujuan untuk mengetahui apakah *timer* mikrokontroler tersebut akurat. Prosedur pengujian dilakukan dengan mikrokontroler diprogram agar setiap terjadi interupsi *timer overflow* maka PORTA akan di togle dan menampilkan sinyal keluaran pada osiloskop. Diagram blok pengujian *timer* mikrokontroler ditunjukkan dalam Gambar 5.1.



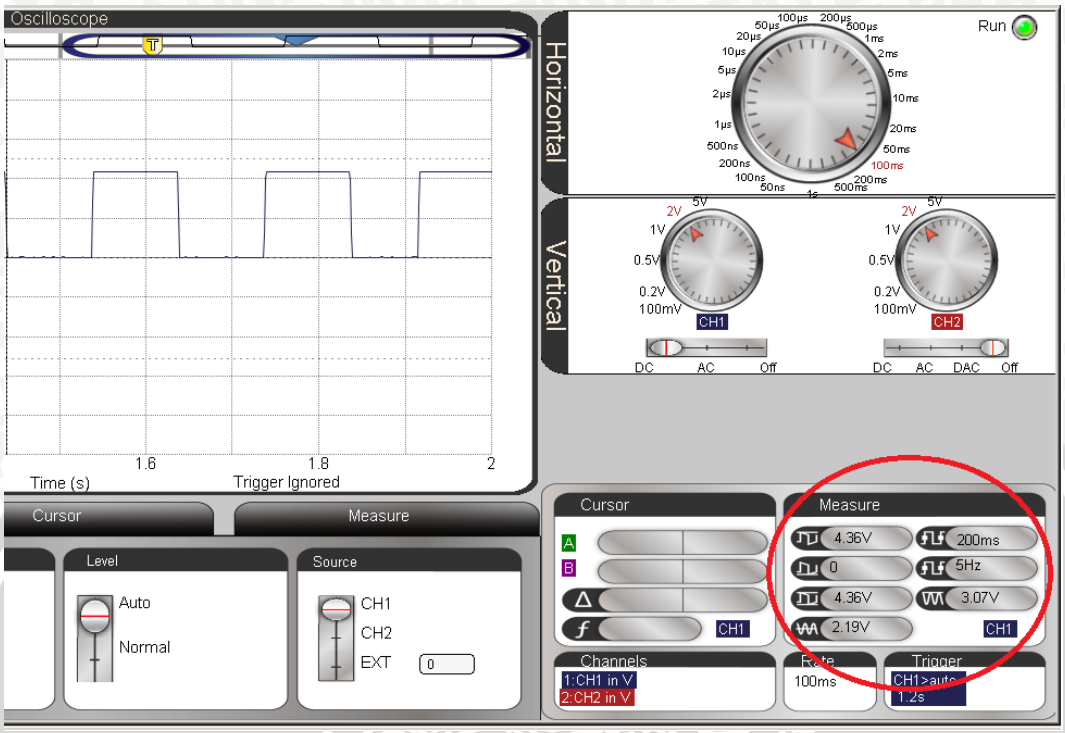
Gambar 5.1. Diagram Blok Pengujian *Timer* Mikrokontroler

Diagram alir dari program pengujian *timer* mikrokontroler ditunjukkan dalam Gambar 5.2.



Gambar 5.2. Diagram Alir Program Pengujian *Timer* Mikrokontroler

Tampilan osiloskop ketika *timer* mikrokontroler di *setting overflow* 100ms ditunjukkan dalam Gambar 5.3.



Gambar 5.3. Tampilan Osiloskop ketika *Timer* Mikrokontroler di *Setting Overflow* 100ms.

Dari Gambar 5.3 pada tab *Measure* terlihat bahwa hasil pengukuran periode bernilai 200ms. Sehingga separuh dari periode tersebut adalah 100ms. Pengujian ini dilakukan berulang-ulang dengan *setting timer* yang bervariasi. Data hasil pengujian *timer* mikrokontroler ditunjukkan dalam Tabel 5.1.

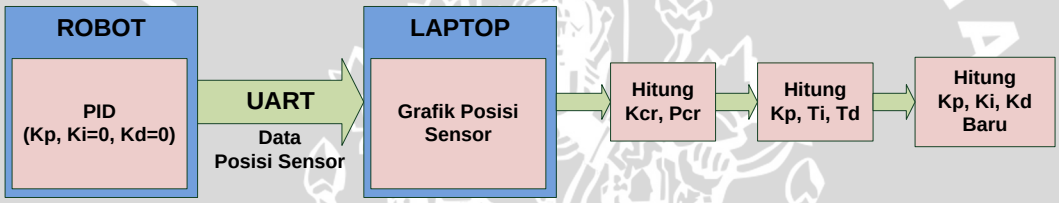
Tabel 5.1. Data Hasil Pengujian *Timer* Mikrokontroler

No	**Setting *Timer* Mikrokontroler(ms)**	**Pengukuran *Timer* Osiloskop(ms)**	**|*Error*(%)|**
1	100	100	0
2	200	200	0
3	300	299,5	0,167
4	400	500	0
5	500	500	0
	Error rata-rata(%)		0,033

Berdasarkan hasil pengujian pada Tabel 5.1 dapat disimpulkan bahwa *timer* mikrokontroler cukup akurat dengan *error* rata-rata 0,033%.

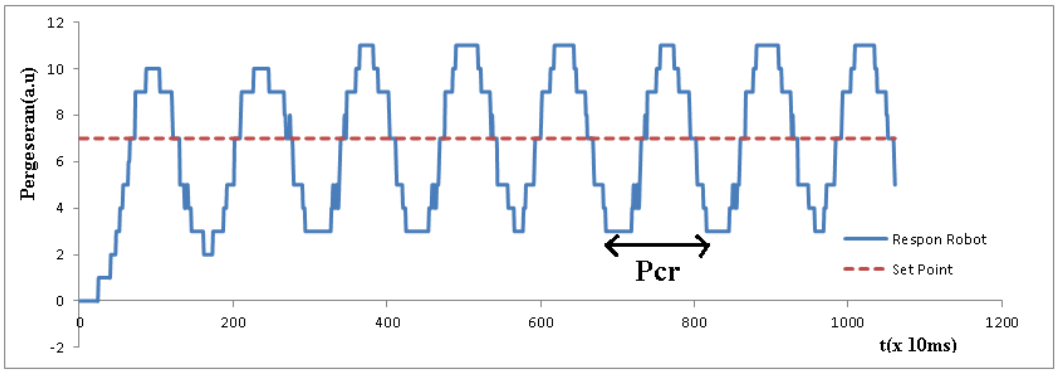
5.2. Pengujian *Tuning* Kontroler PID Menggunakan Metode Kedua Ziegler-Nichols

Pengujian ini bertujuan untuk menentukan parameter Kp, Ki, dan Kd. Proses *Tunning* PID pada perancangan ini menggunakan metode kedua Ziegler Nichols sehingga harus dicari terlebih dahulu nilai Kp saat terjadi osilasi berkesinambungan pada robot dengan nilai Ki dan Kd adalah 0. Dari nilai Kp saat terjadi osilasi berkesinambungan dapat dicari Kcr dan Pcr. Dengan nilai Kcr dan Pcr, dapat dihitung nilai Ki dan Kd. Diagram blok pengujian *tuning* PID pada robot *management* sampah dengan metode kedua Ziegler-Nichols ditunjukkan dalam Gambar 5.4.



Gambar 5.4 Diagram Blok Pengujian *Tuning* PID dengan Metode kedua Ziegler-Nichols

Proses *Tuning* diawali dengan merubah nilai Kp dari 0, 3, 5, 8, 9, dan 10. Grafik respons pergerakan robot yang berkesinambungan didapat saat nilai kontroler proporsional 10 (Kp=10). Hasil pengujian respons pergerakan robot dengan menggunakan kontroler proporsional dengan nilai 10 (Kp=10) dapat dilihat dalam Gambar 5.5.



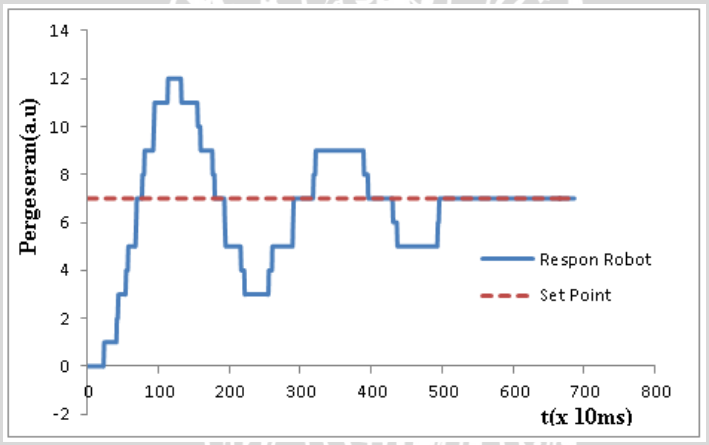
Gambar 5.5 Grafik respons pergerakan robot dengan Kp=10, Ki=0, Kd=0

Gambar 5.5 menunjukkan bahwa pada saat kontroler proporsional bernilai 10 pergerakan robot dapat membentuk osilasi berkesinambungan.

Respons sistem menampilkan data setiap 10 ms sehingga nilai Kcr dan Pcr dapat dihitung. Nilai Ki dan Kd dapat diperoleh dengan perhitungan sebagai berikut

- Kcr = 10
- Pcr = (618 - 490) x Waktu Sampling = 128 x 10 x 10^{-3} = 1,280s
- Kp = 0,6 x Kcr = 0,6 x 10 = 6
- Ti = 0,5 x Pcr = 0,5 x 1,28 = 0,64
- Td = 0,125 x Pcr = 0,125 x 1,28 = 0,16
- $Ki = \frac{Kp}{Ti} = \frac{6}{0,64} = 9,375$
- $Kd = Kp \times Td = 6 \times 0,16 = 0,96$

Hasil *tuning* parameter PID dengan menggunakan metode kedua Ziegler-Nichols pada robot *management* sampah diperoleh nilai Kp = 6, Ki = 9,375, dan Kd = 0,96. Hasil pengujian respons pergerakan robot dengan menggunakan nilai parameter Kp, Ki, Kd tersebut ditunjukkan dalam Gambar 5.6.



Gambar 5.6 Grafik respons pergerakan robot dengan Kp=6, Ki=9,375, Kd=0,96.

Berdasarkan hasil pengujian pada Gambar 5.6 dapat disimpulkan bahwa robot dapat bergerak mengikuti garis dengan baik dengan parameter respons sistem tr = 0,7s; td = 0,545s; tp = 1,14s; ts = 4,96s, dan MP = 71,42%.

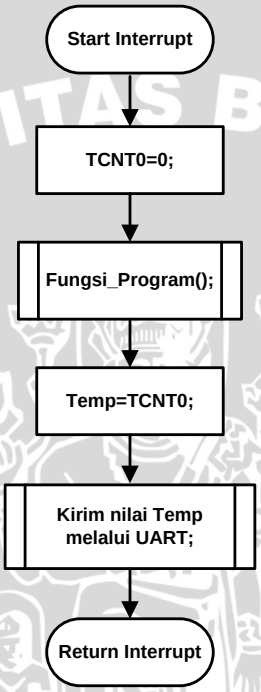
5.3. Pengujian Waktu Eksekusi pada Protokol I2C dan SPI

Pengujian ini bertujuan untuk mengetahui waktu yang dibutuhkan masing-masing protokol I2C dan SPI ketika mengeksekusi program. Pengujian dilakukan dengan cara menjalankan *timer* mikrokontroler dengan memberi nilai 0 pada register TCNT0. Kemudian setelah mengeksekusi program, nilai TCNT0 kita baca

melalui komunikasi UART. Diagram blok pengujian waktu eksekusi ditunjukkan dalam Gambar 5.7.

Gambar 5.7 Diagram Blok Pengujian Waktu Eksekusi.

Untuk diagram alir program pengujian waktu eksekusi ditunjukkan dalam Gambar 5.8.



Gambar 5.8 Diagram Alir Program Pengujian Waktu Eksekusi

Data hasil waktu eksekusi I2C dan SPI ditunjukkan dalam Tabel 5.2.

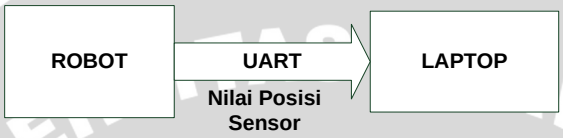
Tabel 5.2. Data Hasil Waktu Eksekusi I2C dan SPI

No	Jenis Perintah (Data)	Waktu (µs)	
		I2C	SPI
1	Meminta Posisi dari Sensor	202,55	104,17
2	Mengirim & Meminta Data Motor	729,17	300,93
3	Eksekusi Fungsi LineFollowing	399,14	399,14
Total		1330,86	804,24

Berdasarkan hasil pengujian dalam Tabel 5.2 dapat disimpulkan bahwa waktu eksekusi pada protokol SPI lebih cepat dibandingkan I2C dengan selisih waktu total 526,62µs.

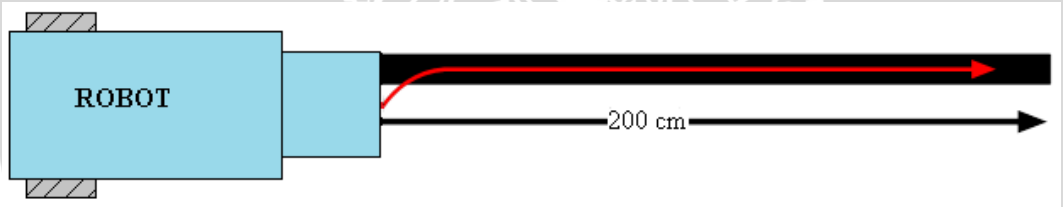
5.4. Pengujian Respons Robot pada Protokol I2C dan SPI dengan Waktu Sampling yang bervariasi

Pengujian ini bertujuan untuk membandingkan respons sistem ketika menggunakan protokol I2C dan SPI dengan waktu sampling yang bervariasi. Variasi waktu sampling yang digunakan yaitu 10ms, 50ms, dan 100ms. Pengujian ini dilakukan dengan beban konstan. Beban rangka robot tanpa bak sampah kurang lebih seberat 5kg. Diagram blok pengujian ini ditunjukkan dalam Gambar 5.9.



Gambar 5.9 Diagram Blok Pengujian Respons Robot

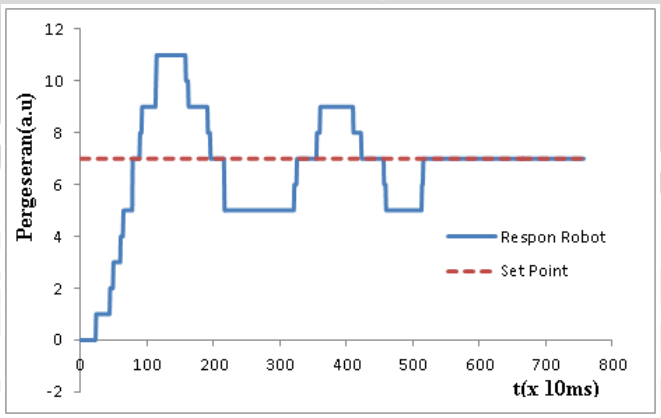
Prosedur pengujian memonitoring pergerakan robot pada lintasan lurus dengan posisi awal sensor robot berada di sebelah kanan lintasan seperti ditunjukkan dalam Gambar 5.10.



Gambar 5.10 Pengujian Respons Robot

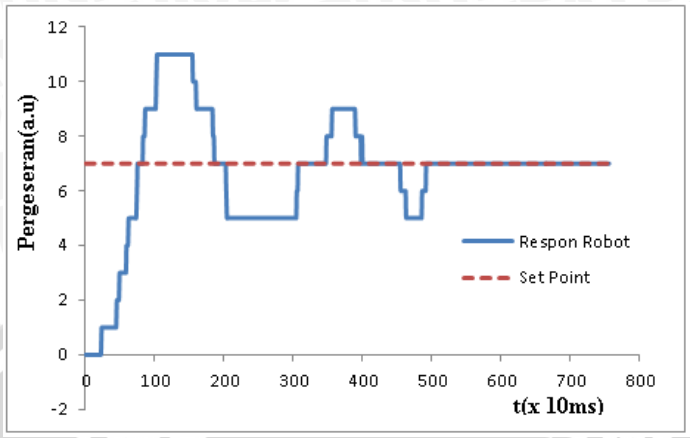
5.4.1. Hasil Pengujian dengan Waktu Sampling 10ms

Hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan I2C ditunjukkan dalam Gambar 5.11.



Gambar 5.11 Grafik Respons Robot menggunakan I2C dengan Waktu Sampling 10ms

Sedangkan hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan SPI ditunjukkan dalam Gambar 5.12.



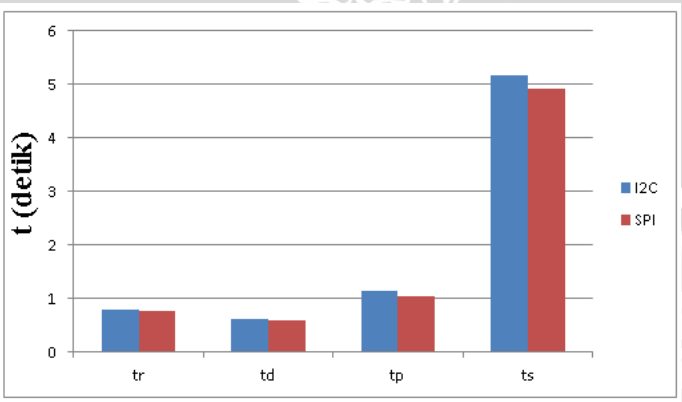
Gambar 5.12 Grafik Respons Robot menggunakan SPI dengan Waktu Sampling 10ms

Dari hasil kedua pengujian tersebut bisa didapatkan nilai masing-masing parameter respons sistem. Data hasil parameter respons sistem ditunjukkan dalam Tabel 5.3.

Tabel 5.3. Data Hasil Parameter Respons I2C dan SPI dengan Waktu Sampling 10ms

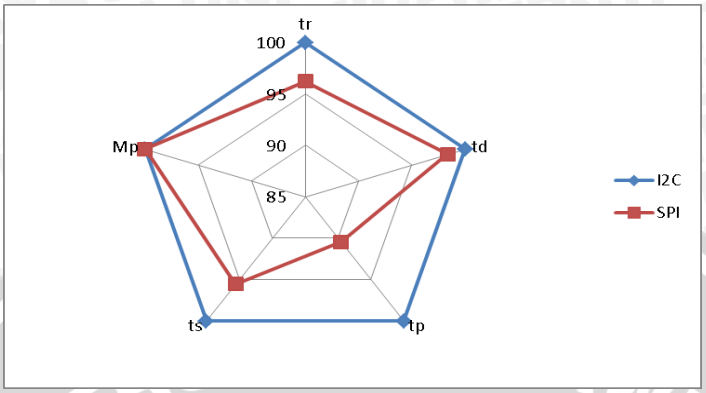
Parameter	I2C	SPI	Satuan
tr	0,79	0,76	sekon
td	0,605	0,595	sekon
tp	1,15	1,04	sekon
ts	5,16	4,93	sekon
Mp	57,14	57,14	%

Dari Tabel 5.3 dapat kita bandingkan parameter respons sistem saat menggunakan I2C dan SPI. Grafik hasil perbandingan kedua sistem ditunjukkan dalam Gambar 5.13.



Gambar 5.13 Grafik Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Waktu Sampling 10ms

Normalisasi grafik radar hasil perbandingan respons sistem menggunakan protokol I2C dan SPI dengan waktu sampling 10ms ditunjukkan dalam Gambar 5.14.

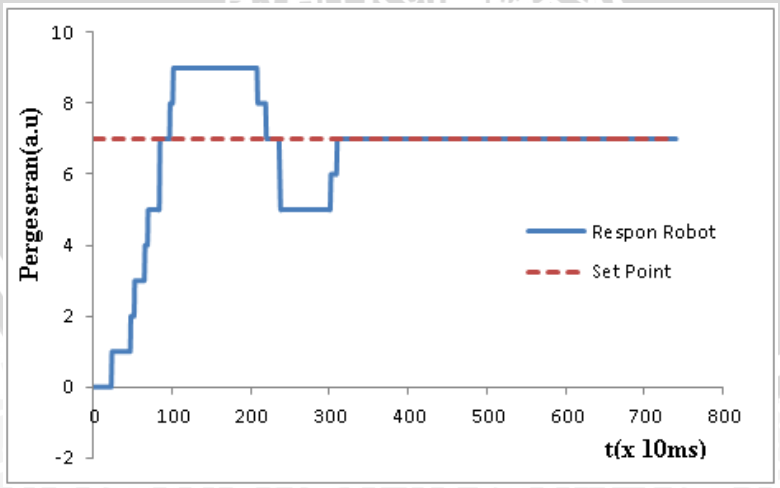


Gambar 5.14. Normalisasi Grafik Radar Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Waktu Sampling 10ms

Berdasarkan hasil pengujian dengan waktu sampling 10ms dapat disimpulkan bahwa robot dapat bergerak mengikuti garis dengan baik. Respons robot saat menggunakan protokol SPI lebih cepat dibandingkan saat menggunakan protokol I2C dengan selisih waktu tr = 0,03s; td = 0,01s; tp = 0,11s; ts = 0,23s.

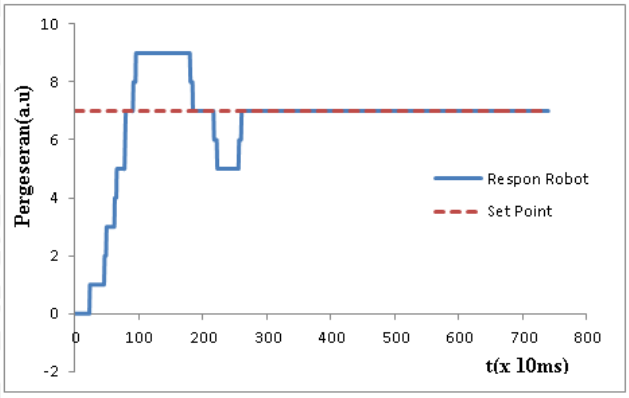
5.4.2. Hasil Pengujian dengan Waktu Sampling 50ms

Hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan I2C ditunjukkan dalam Gambar 5.15.



Gambar 5.15 Grafik Respons Robot menggunakan I2C dengan Waktu Sampling 50ms

Sedangkan hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan SPI ditunjukkan dalam Gambar 5.16 .



Gambar 5.16 Grafik Respons Robot menggunakan SPI dengan Waktu Sampling 50ms

Dari hasil kedua pengujian tersebut bisa didapatkan nilai masing-masing parameter respons sistem. Data hasil parameter respons sistem ditunjukkan dalam Tabel 5.4.

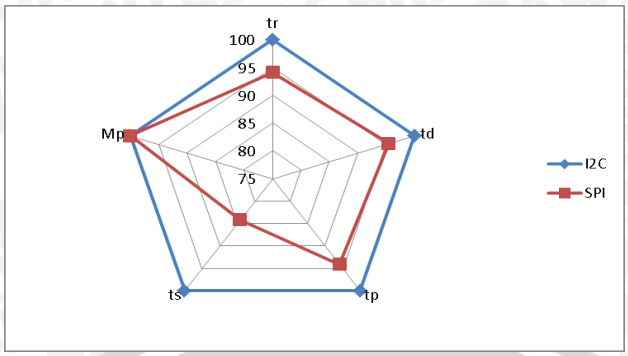
Tabel 5.4. Data Hasil Parameter Respons I2C dan SPI dengan Waktu Sampling 50ms

Parameter	I2C	SPI	Satuan
tr	0,85	0,8	sekon
td	0,655	0,625	sekon
tp	1,02	0,96	sekon
ts	3,1	2,61	sekon
Mp	28,57	28,57	%

Dari Tabel 5.4 dapat kita bandingkan parameter respons sistem saat menggunakan I2C dan SPI. Grafik hasil perbandingan kedua sistem ditunjukkan dalam Gambar 5.17.

Gambar 5.17 Grafik Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Waktu Sampling 50ms

Normalisasi grafik radar hasil perbandingan respons sistem menggunakan protokol I2C dan SPI dengan waktu sampling 50ms ditunjukkan dalam Gambar 5.18.

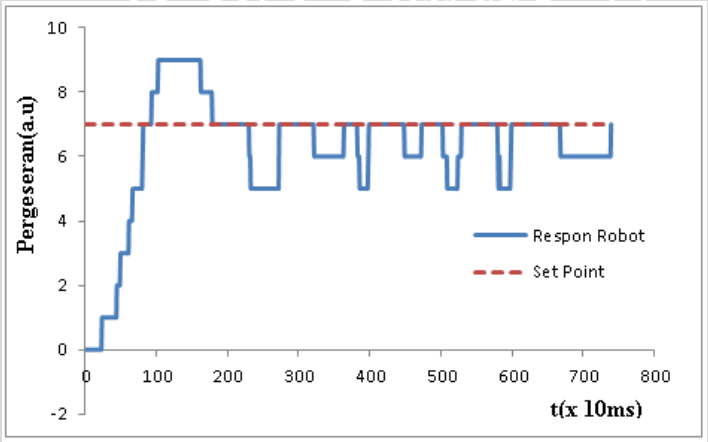


Gambar 5.18. Normalisasi Grafik Radar Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Waktu Sampling 50ms

Berdasarkan hasil pengujian dengan waktu sampling 50ms dapat disimpulkan bahwa robot dapat bergerak mengikuti garis dengan baik. Respons robot saat menggunakan protokol SPI lebih cepat dibandingkan saat menggunakan protokol I2C dengan selisih waktu tr = 0,05s; td = 0,03s; tp = 0,06s; ts = 0,49s.

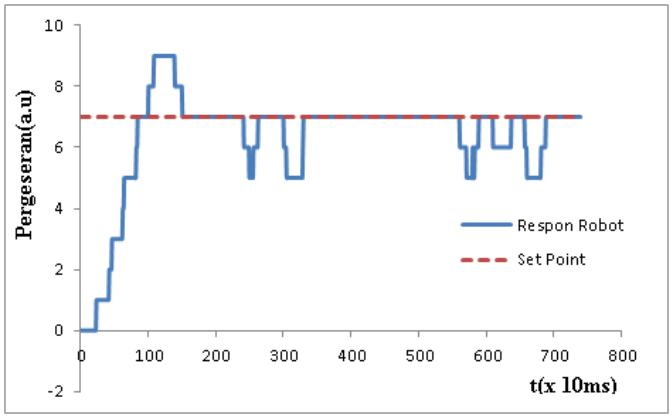
5.4.3. Hasil Pengujian dengan Waktu Sampling 100ms

Hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan I2C ditunjukkan dalam Gambar 5.19.



Gambar 5.19 Grafik Respons Robot menggunakan I2C dengan Waktu Sampling 100ms

Sedangkan hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan SPI ditunjukkan dalam Gambar 5.20.



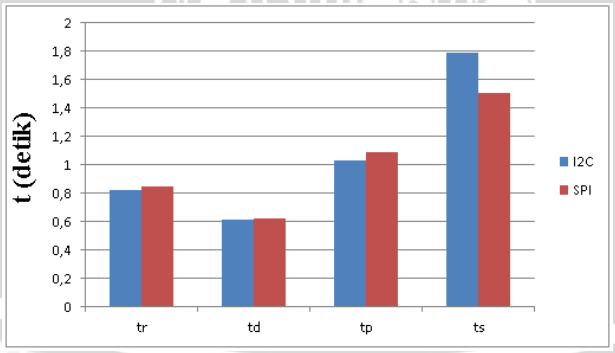
Gambar 5.20 Grafik Respons Robot menggunakan SPI dengan Waktu Sampling 100ms

Dari hasil kedua pengujian tersebut bisa didapatkan nilai masing-masing parameter respons sistem. Data hasil parameter respons sistem ditunjukkan dalam Tabel 5.5.

Tabel 5.5. Data Hasil Parameter Respons I2C dan SPI dengan Waktu Sampling 100ms

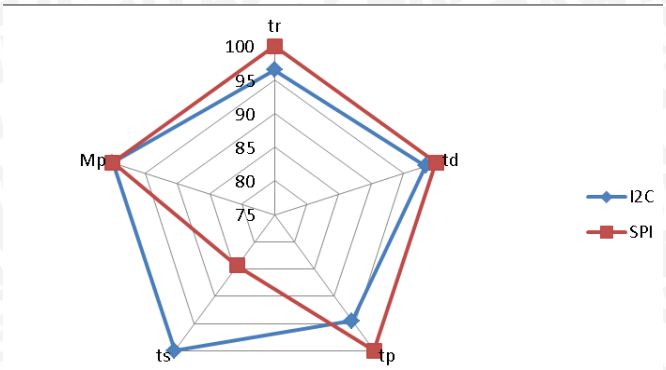
Parameter	I2C	SPI	Satuan
tr	0,82	0,85	sekon
td	0,615	0,625	sekon
tp	1,03	1,09	sekon
ts	1,79	1,51	sekon
Mp	28,57	28,57	%

Dari Tabel 5.5 dapat kita bandingkan parameter respons sistem saat menggunakan I2C dan SPI. Grafik hasil perbandingan kedua sistem ditunjukkan dalam Gambar 5.21.



Gambar 5.21 Grafik Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Waktu Sampling 100ms

Normalisasi grafik radar hasil perbandingan respons sistem menggunakan protokol I2C dan SPI dengan waktu sampling 100ms ditunjukkan dalam Gambar 5.22.



Gambar 5.22. Normalisasi Grafik Radar Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Waktu Sampling 100ms

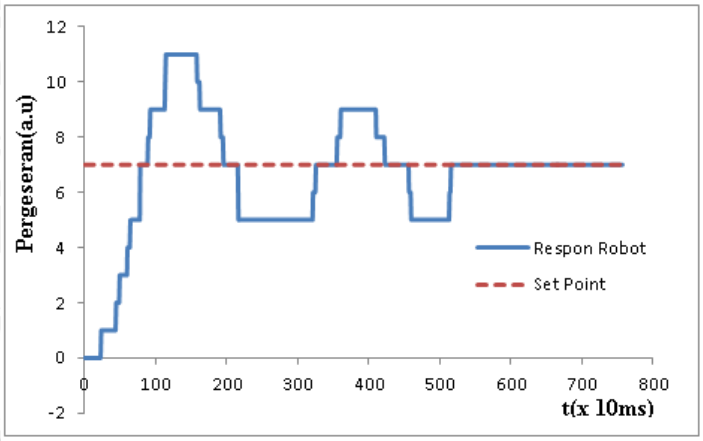
Berdasarkan hasil pengujian dengan waktu sampling 100ms dapat disimpulkan bahwa robot dapat bergerak mengikuti garis dengan baik. Untuk parameter respons robot tr, td, dan tp saat menggunakan protokol I2C lebih cepat dibandingkan saat menggunakan protokol SPI dengan selisih waktu tr = 0,03s; td = 0,01s; tp = 0,06s. Sedangkan untuk parameter ts saat menggunakan protokol SPI lebih cepat dibandingkan saat menggunakan protokol I2C dengan selisih waktu ts = 0,28s.

5.5. Pengujian Respons Robot pada Protokol I2C dan SPI dengan Beban Robot yang Bervariasi

Pengujian ini bertujuan untuk membandingkan respons sistem ketika menggunakan protokol I2C dan SPI dengan beban robot yang bervariasi. Variasi beban robot yang digunakan yaitu 5kg, 10kg, dan 15kg . Pengujian ini dilakukan dengan waktu sampling konstan yaitu 10ms.

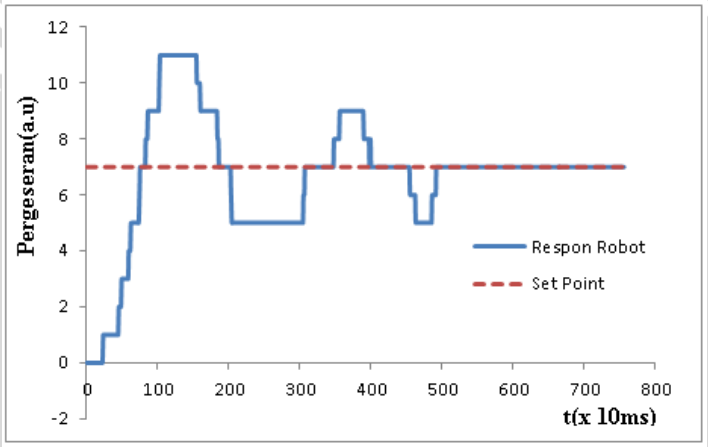
5.5.1. Hasil Pengujian dengan Beban Robot 5kg

Hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan I2C ditunjukkan dalam Gambar 5.23.



Gambar 5.23 Grafik Respons Robot menggunakan I2C dengan Beban Robot 5kg

Sedangkan hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan SPI ditunjukkan dalam Gambar 5.24.



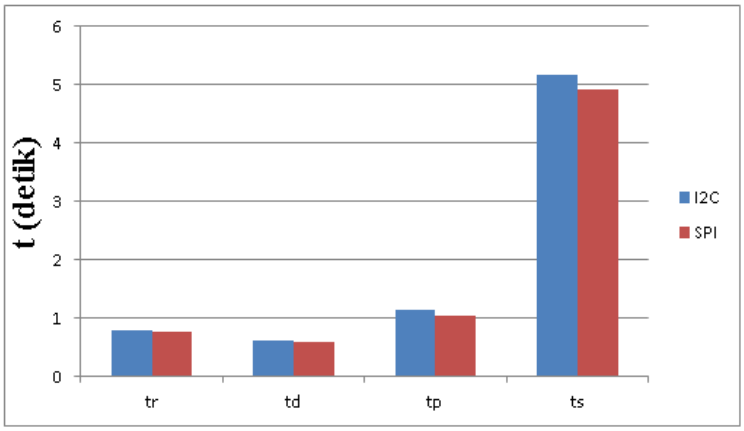
Gambar 5.24 Grafik Respons Robot menggunakan SPI dengan Beban Robot 5kg

Dari hasil kedua pengujian tersebut bisa didapatkan nilai masing-masing parameter respons sistem. Data hasil parameter respons sistem ditunjukkan dalam Tabel 5.6.

Tabel 5.6. Data Hasil Parameter Respons I2C dan SPI dengan Beban Robot 5kg

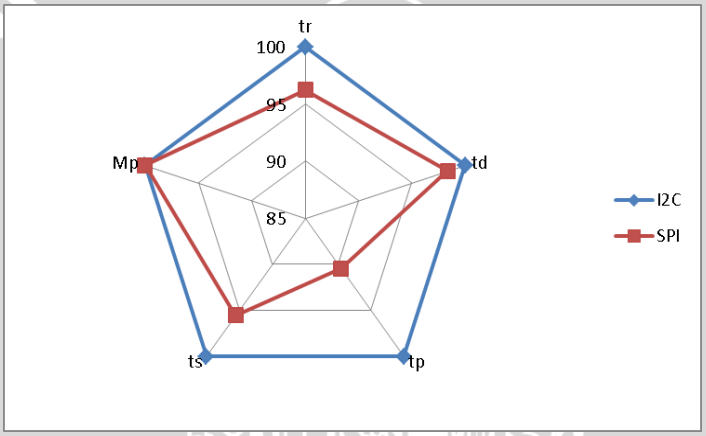
Parameter	I2C	SPI	Satuan
tr	0,79	0,76	sekon
td	0,605	0,595	sekon
tp	1,15	1,04	sekon
ts	5,16	4,93	sekon
Mp	57,14	57,14	%

Dari Tabel 5.6 dapat kita bandingkan parameter respons sistem saat menggunakan I2C dan SPI. Grafik hasil perbandingan kedua sistem ditunjukkan dalam Gambar 5.25.



Gambar 5.25 Grafik Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Beban Robot 5kg

Normalisasi grafik radar hasil perbandingan respons sistem menggunakan protokol I2C dan SPI dengan beban robot 5kg ditunjukkan dalam Gambar 5.26.

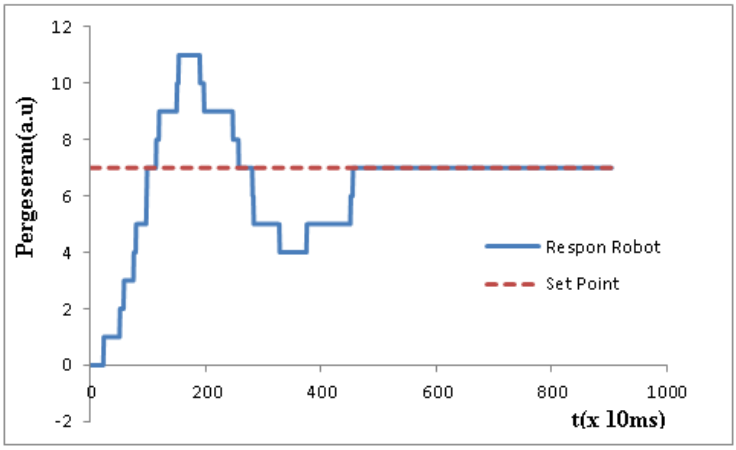


Gambar 5.26. Normalisasi Grafik Radar Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Beban Robot 5kg

Berdasarkan hasil pengujian dengan beban robot 5kg dapat disimpulkan bahwa robot dapat bergerak mengikuti garis dengan baik. Respons robot saat menggunakan protokol SPI lebih cepat dibandingkan saat menggunakan protokol I2C dengan selisih waktu tr = 0,03s; td = 0,01s; tp = 0,11s; ts = 0,23s.

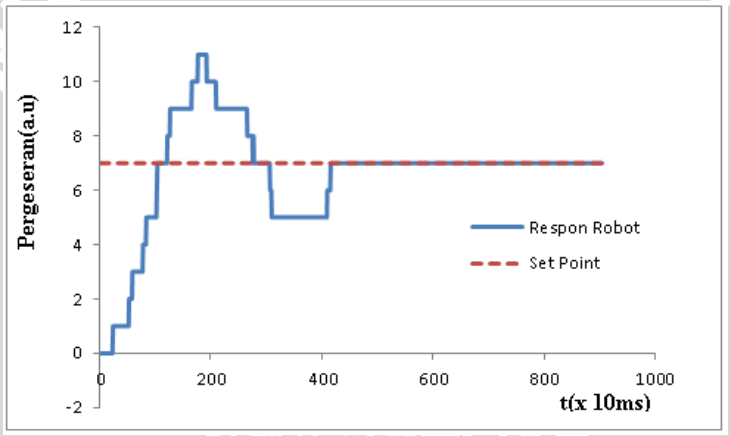
5.5.2. Hasil Pengujian dengan Beban Robot 10kg

Hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan I2C ditunjukkan dalam Gambar 5.27.



Gambar 5.27 Grafik Respons Robot menggunakan I2C dengan Beban Robot 10kg

Sedangkan hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan SPI ditunjukkan dalam Gambar 5.28.



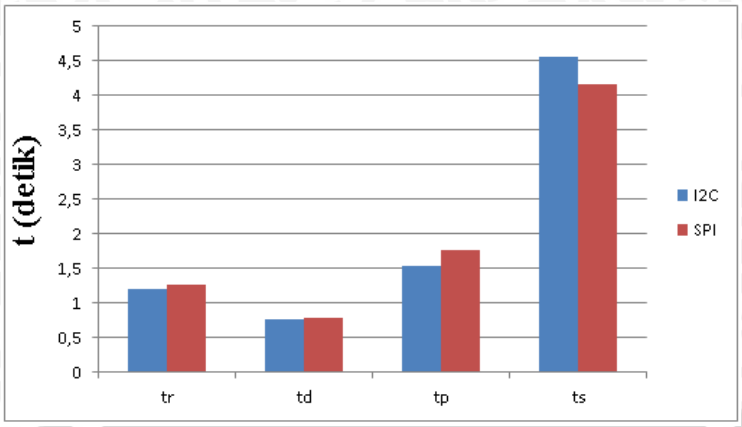
Gambar 5.28 Grafik Respons Robot menggunakan SPI dengan Beban Robot 10kg

Dari hasil kedua pengujian tersebut bisa didapatkan nilai masing-masing parameter respons sistem. Data hasil parameter respons sistem ditunjukkan dalam Tabel 5.7.

Tabel 5.7. Data Hasil Parameter Respons I2C dan SPI dengan Beban Robot 10kg

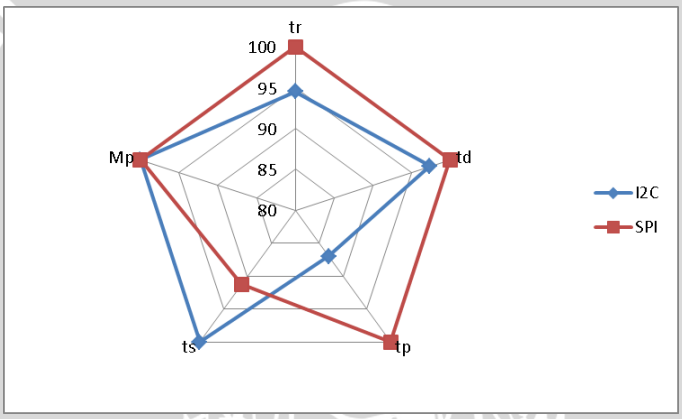
Parameter	I2C	SPI	Satuan
tr	1,2	1,27	sekon
td	0,755	0,775	sekon
tp	1,54	1,77	sekon
ts	4,56	4,16	sekon
Mp	57,14	57,14	%

Dari Tabel 5.7 dapat kita bandingkan parameter respons sistem saat menggunakan I2C dan SPI. Grafik hasil perbandingan kedua sistem ditunjukkan dalam Gambar 5.29.



Gambar 5.29 Grafik Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Beban Robot 10kg

Normalisasi grafik radar hasil perbandingan respons sistem menggunakan protokol I2C dan SPI dengan beban robot 10kg ditunjukkan dalam Gambar 5.30.

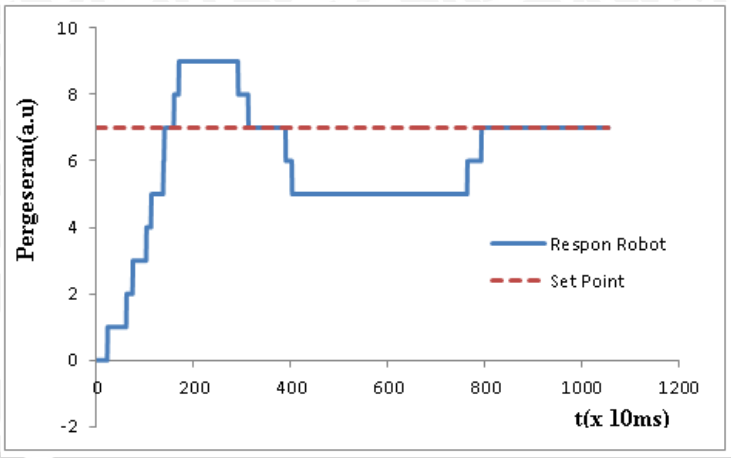


Gambar 5.30. Normalisasi Grafik Radar Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Beban Robot 10kg

Berdasarkan hasil pengujian dengan beban robot 10kg dapat disimpulkan bahwa robot dapat bergerak mengikuti garis dengan baik. Untuk parameter respons robot tr, td, dan tp saat menggunakan protokol I2C lebih cepat dibandingkan saat menggunakan protokol SPI dengan selisih waktu tr = 0,07s; td = 0,02s; tp = 0,23s. Sedangkan untuk parameter ts saat menggunakan protokol SPI lebih cepat dibandingkan saat menggunakan protokol I2C dengan selisih waktu ts = 0,4s.

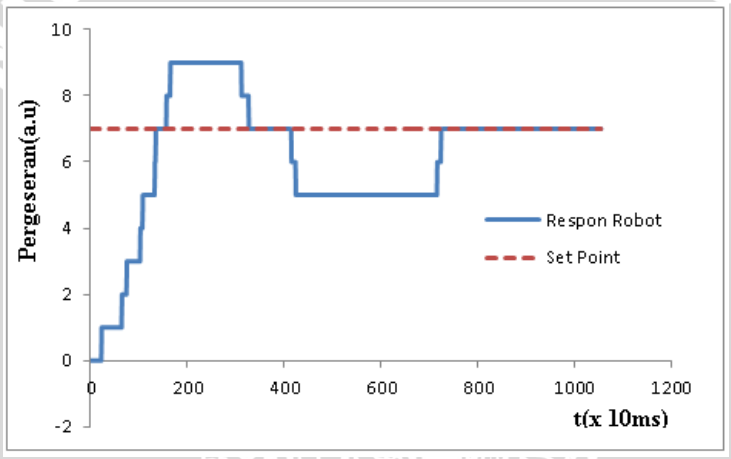
5.5.3. Hasil Pengujian dengan Beban Robot 15kg

Hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan I2C ditunjukkan dalam Gambar 5.31.



Gambar 5.31 Grafik Respons Robot menggunakan I2C dengan Beban Robot 15kg

Sedangkan hasil pengujian untuk respons posisi sensor robot saat sistem menggunakan SPI ditunjukkan dalam Gambar 5.32.



Gambar 5.32 Grafik Respons Robot menggunakan SPI dengan Beban Robot 15kg

Dari hasil kedua pengujian tersebut bisa didapatkan nilai masing-masing parameter respons sistem. Data hasil parameter respons sistem ditunjukkan dalam Tabel 5.8.

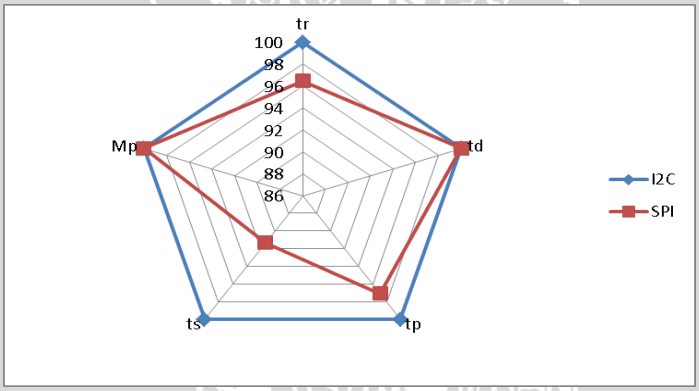
Tabel 5.8. Data Hasil Parameter Respons I2C dan SPI dengan Beban Robot 15kg

Parameter	I2C	SPI	Satuan
tr	1,41	1,36	sekon
td	1,03	1,03	sekon
tp	1,71	1,66	sekon
ts	7,94	7,25	sekon
Mp	28,57	28,57	%

Dari Tabel 5.8 dapat kita bandingkan parameter respons sistem saat menggunakan I2C dan SPI. Grafik hasil perbandingan kedua sistem ditunjukkan dalam Gambar 5.33.

Gambar 5.33 Grafik Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Beban Robot 15kg

Normalisasi grafik radar hasil perbandingan respons sistem menggunakan protokol I2C dan SPI dengan beban robot 15kg ditunjukkan dalam Gambar 5.34.



Gambar 5.34. Normalisasi Grafik Radar Hasil Perbandingan Respons Sistem menggunakan I2C dan SPI dengan Beban Robot 15kg

Berdasarkan hasil pengujian dengan beban robot 15kg dapat disimpulkan bahwa respons robot tr, tp, dan ts saat menggunakan protokol SPI lebih cepat dibandingkan saat menggunakan protokol I2C dengan selisih waktu tr = 0,05s; tp = 0,05s; ts = 0,69s. Sedangkan untuk parameter ts saat menggunakan protokol SPI nilainya sama saat menggunakan protokol I2C yaitu td = 1,03s.

Perbedaan respons robot *management* sampah ketika menggunakan protokol I2C dan SPI tidak terlalu signifikan dikarenakan selisih waktu eksekusi program total hanya 526,62µs dan robot bergerak dengan kecepatan kurang lebih 10cm/s.

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan hasil perancangan dan pengujian yang telah dilakukan, dapat diambil kesimpulan sebagai berikut :

1) Rangkaian antarmuka protokol I2C pada sistem ini menggunakan dua jalur koneksi utama SCL (*Serial Clock Line)* sebagai sinyal clock dan SDA (*Serial Data)* sebagai transmisi data. Masing-masing jalur dengan resistor *pull-up* 4,7kΩ. Sedangkan rangkaian antarmuka protokol SPI menggunakan empat jalur koneksi utama MOSI (*Master Out – Slave In)*, MISO (*Master In – Slave Out)*, SCK (*Serial Clock)*, dan SS (*Slave Select)* ke masing-masing *slave*.
2) Perangkat lunak pada mikrokontroller *Slave Sensor* difungsikan sebagai pengolah data dari sensor garis melalui ADC 8-*bit* dan dengan multiplekser analog. Perangkat lunak pada mikrokontroller *Master* difungsikan sebagai pengolah data sensor 8-*bit* dari *Slave Sensor* menjadi data kecepatan dan *Direction* motor kanan dan kiri untuk dikirim ke mikrokontroller *Slave Motor* kanan dan kiri. Perangkat lunak pada mikrokontroller *Slave Motor* kanan maupun kiri menerjemahkan data kecepatan dan *direction* motor dari *Master* untuk diolah menjadi sinyal keluaran PWM dan logika *direction* ke *driver motor*.
3) Eksekusi program pada protokol SPI lebih cepat dibandingkan dengan protokol I2C dengan selisih waktu 526,62µs. Respon sistem pada protokol SPI juga cenderung lebih cepat dari pada protokol I2C pada waktu sampling 10ms dan beban robot 5kg dengan selisih waktu tr = 0,03s; td = 0,01s; tp = 0,11s; ts = 0,23s.

6.2 Saran

Beberapa hal yang direkomendasikan untuk pengembangan lebih lanjut adalah:

1) Penulis menyarankan untuk menggunakan protokol I2C pada robot ini karena pengkabelannya lebih sederhana dari pada SPI dan respons robot untuk keperluan pembuangan sampah sudah cukup.
2) Untuk memperoleh grafik respons pergerakan robot yang lebih halus disarankan menggunakan jumlah sensor yang lebih banyak.
3) Perlu dilakukan penelitian atau pengembangan selanjutnya untuk menerapkan sistem kontrol kaskade pada robot, yaitu penggabungan antara kontrol robot *line follower* dan kontrol kecepatan motor pada robot.

DAFTAR PUSTAKA

Alexandrescu, N., Apostolescu, T. C., Udrea, C., Duminica, D., Cartal, L. A. 2010. *Autonomous Mobile Robot with Displacements in a Vertical Plane and Applications in Cleaning Services*. Bucharest: University of Bucharest.

Atmel. 2010. *ATmega16/ATmega16L Datasheet*. San Jose: Atmel Corporation. http://www.atmel.com/images/doc2466.pdf. (diakses 19 Juni 2014).

Atmel. 2011. *ATmega8/ATmega8L Datasheet*. San Jose: Atmel Corporation. http://www.atmel.com/images/doc2553.pdf. (diakses 19 Juni 2014).

Blocher, R. 2003. *Dasar Elektronika*. Yogyakarta: Andi.

Corelis. 2013. *SPI Tutorial*. Cerritos: Corelis Corporation. http://www.corelis.com/contact/index.htm. (diakses 19 Juni 2014).

Nalwan, P. A. 2004. *Penggunaan dan Antarmuka Modul LCD M1632*. Jakarta: PT Elek Media Komputindo Kelompok Gramedia.

Nuraini, D. & Pangestu, S. 2011. *Tugas Akhir Robot Pengangkut Sampah Otomatis untuk Distribusi Sampah di dalam Gedung (Studi Kasus di Gedung Robotika ITS).* Skripsi tidak dipublikasikan Surabaya: ITS.

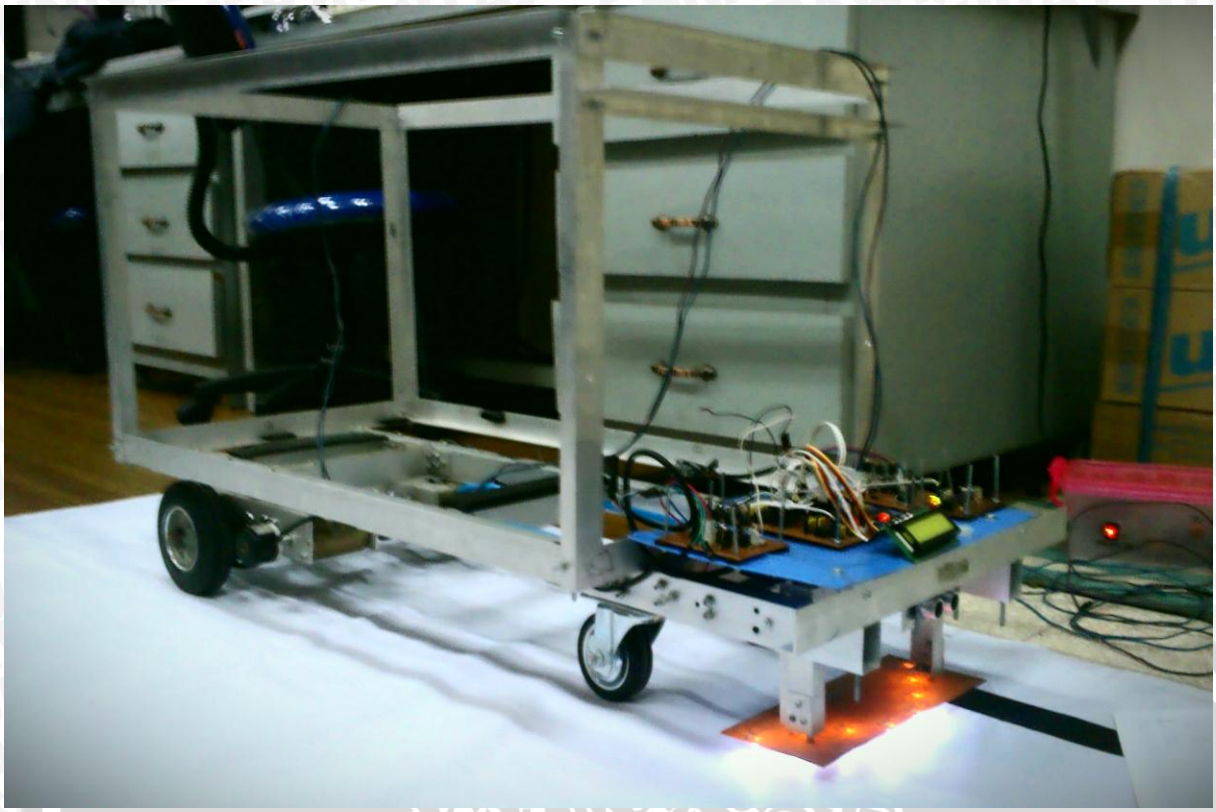
NXP. 2014. *I2C Bus Specification and User Manual*. Eindhoven: NXP. http://www.nxp.com/documents/user_manual/UM10204.pdf. (diakses 19 Juni 2014).

Ogata, K. 1997. *Teknik Kontrol Automatik Jilid 1*. Jakarta: Penerbit Erlangga.

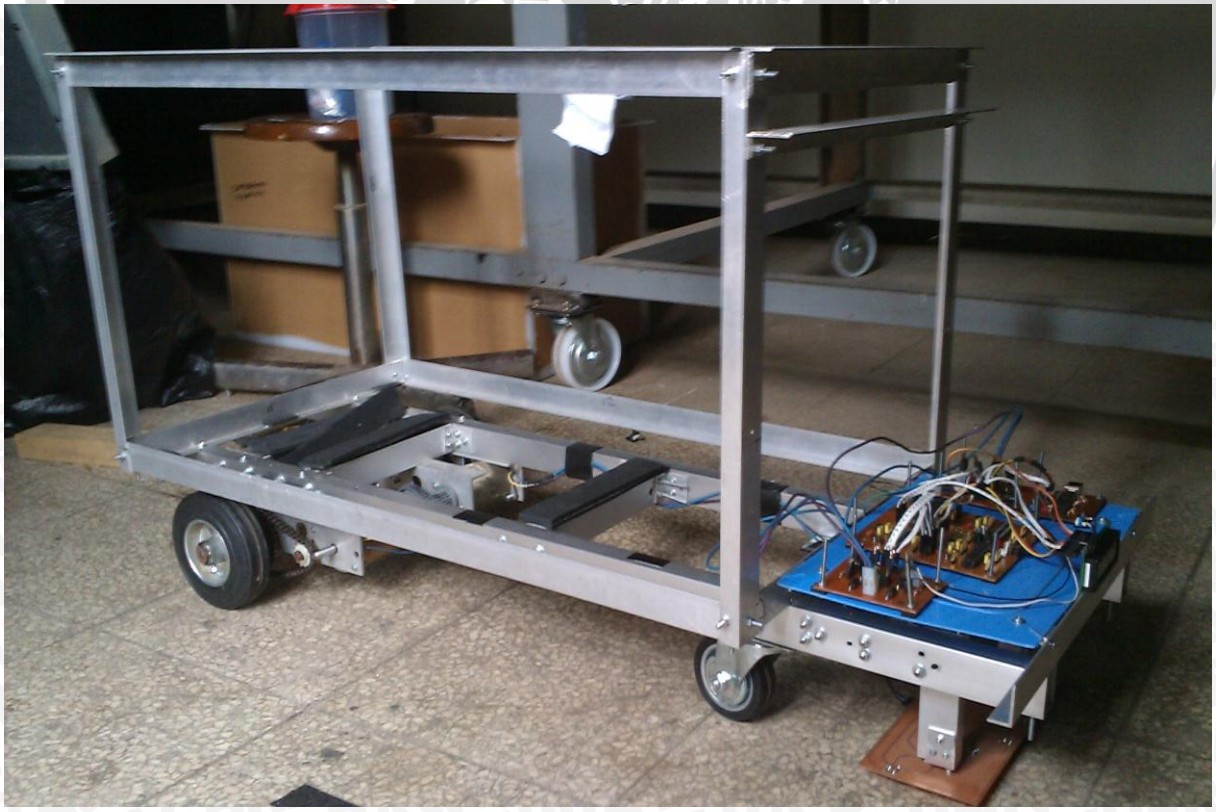
Winoto, A. 2008. *Mikrokontroller AVR Atmega8/32/16/8535 dan Pemrogramannya dengan Bahasa C pada WinAVR*. Bandung: Informatika.

LAMPIRAN I

Dokumentasi Alat

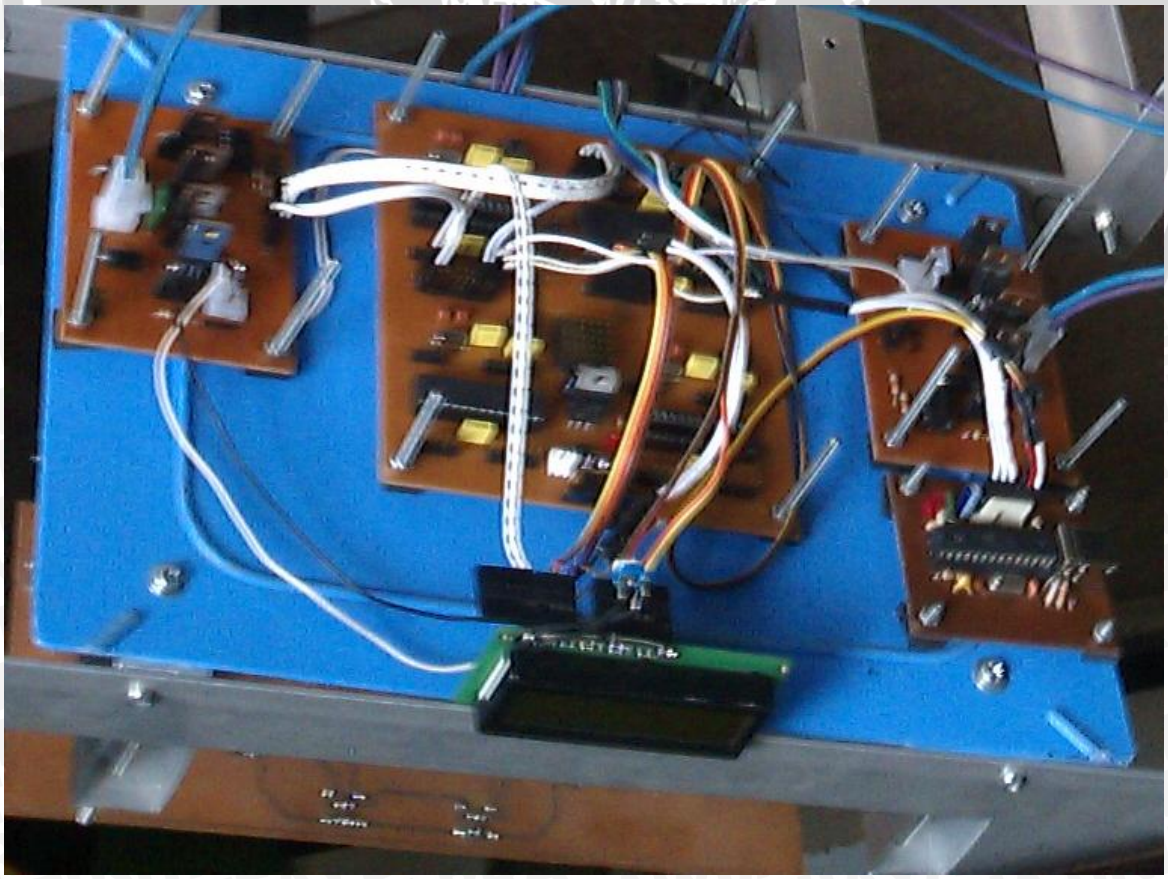


Gambar 1. Robot Saat Pengujian



Gambar 2. Robot Setelah Selesai Pembuatan

Gambar 3. Robot Tampak Perspektif



Gambar 4. Bagian Elektrik Robot

LAMPIRAN II

Listing Program

LISTING PROGRAM MIKROKONTROLER MASTER

```
#define waktu_sampling          10      //satuan ms
#define time_sampling   (65535-43200*waktu_sampling/1000)

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <compat/twi.h>

#include "SPI.h"
#include "lcd.h"

volatile float Kp=1, Ki=1, Kd=1, PID;
volatile float lpwm, rpwm, MAXPWM=100;
volatile float error, set_point=7, del_error, last_error, sigma_error;
volatile int sensor;
volatile int dirka,dirki,pwmka,pwmki,j=0,sampling_cnt=1;
volatile int puluhribu,ratusan,puluhan,satuan,temp;

void PID_Line_Following();
void InitUART(void);
void TransmitByte( unsigned char data );
void tampil(int dat);

int main (void)
{
        _delay_ms(2500);
        i2c_init();
        //SPI_MasterInit_int();
        DDRC = (1<<4)|(1<<5)|(1<<6)|(1<<7);
        DDRD = (1<<4)|(1<<5)|(1<<6)|(1<<7);
        lcd_init();
        lcd_clrscr();
        DDRA = 255;
        _delay_ms(7500);
        InitUART();
        TCCR0 = ( (1<<CS01) | (1<<CS00) );
        TIMSK = 0x04;
        TCCR1B = ( (1<<CS12) | (0<<CS10) );
        TCNT1 = time_sampling;
        sei();
        while (1)
        {
                lcd_gotoxy(0,1);
                tampil(sensor);
        }
        return 0;
}

ISR(TIMER1_OVF_vect){
        TCNT1 = time_sampling;
```

```
/*
    count=0;
    DDRB  |= _BV(1);//ARAH=OUT;
    PORTB |= _BV(1);//pulsa=1;
    _delay_us(5);
    PORTB &= ~_BV(1);//pulsa=0;
    // port as input
    DDRB  &= ~_BV(1);//ARAH=INP;
    // with pull-up
    PORTB |= _BV(1);//pulsa=1;
    while (bit_is_clear(PINB,1)) {};
    while (bit_is_set(PINB,1))
    {
    count++;
    }
    waktu=(count*0.09012906482); // KRISTAL = 11,059200Mhz
    jarak=((waktu*0.3495)/2);
    jarak=jarak*1.3;       //Kalibrasi
          if(jarak<=150)
                PORTB |= _BV(0);//bel=1;
          else
                PORTB &= ~_BV(0);//bel=0;
          if(jarak<=100)
                MAXPWM = 0;
          else if(jarak<=200){
                MAXPWM = (jarak-100)*10;
                if(MAXPWM>200)
                      MAXPWM=200;
          }
          else
                MAXPWM = 200;
*/
/*
          PV = jarak;
          error = (float)(PV-100);
    P = Kp * error;

    I = (Ki * (error + last_error)) * 0.2;

    D = (Kd * (error - last_error)) / 0.2;

    last_error = error;

    MV = P + I + D;

    speed_temp = MV;

    if(speed_temp<-255)  //speed_temp=speed sementara
          speed=-255;
          else if(speed_temp>255)
                speed=255;
    else
```

```
		speed=speed_temp;

		if(speed>=0){
			dirka=1;
			dirki=1;
		}
		else{
			dirka=0;
			dirki=0;
			speed=-speed;
		}
		pwmka=6*speed;
		pwmki=pwmka;
*/
/*
		if(jarak>=100)
			scanBlackLine2();
		else{

			dirka=0;
			dirki=0;
			speed=150-jarak;

		pwmka=6*speed;
		pwmki=pwmka;

		}
*/
		//SENSOR
		//**********************I2C**********
		i2c_start();
		check_status(TW_START);
		i2c_write(0xD0+TW_READ);
		check_status(TW_MR_SLA_ACK);
		sensor = i2c_readNak();
		check_status(TW_MR_DATA_NACK);
		i2c_stop();
		//**********************SPI*************
/*
		PORTA=0b10100000;
		SPI_MasterTransmit_int(0xAA);
		SPI_MasterTransmit_int(0xff);
		sensor   = SPDR;
*/
		j++;
		if(j>=sampling_cnt){
			PID_Line_Following();
			j=0;
		}

		//MOTOR KANAN
		//**********************I2C**********
```

```
		i2c_start();
		check_status(TW_START);
		i2c_write(0xD2+TW_WRITE);
		check_status(TW_MT_SLA_ACK);
		if(dirka)
			i2c_write(0x0f);//maju
		else
			i2c_write(0xf0);//mundur
		check_status(TW_MT_DATA_ACK);
		i2c_write(pwmka%255);//lsb
		check_status(TW_MT_DATA_ACK);
		i2c_write(pwmka/255);//msb
		check_status(TW_MT_DATA_ACK);
		i2c_stop();
		//*********************SPI*************
/*
		PORTA=0b01100000;
		SPI_MasterTransmit_int(255);
		SPI_MasterTransmit_int(255);
		if(dirka)
			SPI_MasterTransmit_int(0x0f);//maju
		else
			SPI_MasterTransmit_int(0xf0);//mundur
		SPI_MasterTransmit_int(pwmka%255);//lsb
		SPI_MasterTransmit_int(pwmka/255);//msb
*/
		//MOTOR KIRI
		//*********************I2C***********
		i2c_start();
		check_status(TW_START);
		i2c_write(0xD4+TW_WRITE);
		check_status(TW_MT_SLA_ACK);
		if(dirki)
			i2c_write(0x0f);//maju
		else
			i2c_write(0xf0);//mundur
		check_status(TW_MT_DATA_ACK);
		i2c_write(pwmki%255);//lsb
		check_status(TW_MT_DATA_ACK);
		i2c_write(pwmki/255);//msb
		check_status(TW_MT_DATA_ACK);
		i2c_stop();
		//*********************SPI*************
/*
		PORTA=0b11000000;
		SPI_MasterTransmit_int(255);
		SPI_MasterTransmit_int(255);
		if(dirki)
			SPI_MasterTransmit_int(0x0f);
		else
			SPI_MasterTransmit_int(0xf0);
		SPI_MasterTransmit_int(pwmki%255);//lsb
```

```
        SPI_MasterTransmit_int(pwmki/255);//msb
*/
        temp = sensor;
        puluhribu = temp/1000 +48;
        ratusan   = (temp%1000)/100 +48;
        puluhan   = (temp%100)/10 +48;
        satuan    = (temp%10) +48;
        TransmitByte(puluhribu);
        TransmitByte(ratusan);
        TransmitByte(puluhan);
        TransmitByte(satuan);
        TransmitByte(0x0D);
}

void PID_Line_Following(){
        error = set_point - sensor;
        del_error = error - last_error;
        sigma_error += error;

        PID = error*Kp + del_error*Ki + sigma_error*Kd;

        rpwm = MAXPWM + PID;
        lpwm = MAXPWM - PID;

        if (lpwm < 0) lpwm = 0;
        if (lpwm > 255) lpwm = 255;
        if (rpwm < 0) rpwm = 0;
        if (rpwm > 255) rpwm = 255;

        pwmki = 6*lpwm;
        pwmka = 6*rpwm;
        dirki=1;
        dirka=1;

        last_error = error;
}

void InitUART(void){
        UCSRA=0x00;
        UCSRB=0x98;
        UCSRC=0x86;
        UBRRH=0x00;
        UBRRL=11;
}

void TransmitByte( volatile unsigned char data ){
        while ( !(UCSRA & (1<<UDRE)) );       // Wait for empty transmit buffer
        UDR = data;// Start transmittion
}

void tampil(int dat){
   int data;
```

```
    data = dat / 10000;
    data+=0x30;
    lcd_putch(data);

          dat%=10000;
    data = dat / 1000;
    data+=0x30;
    lcd_putch(data);

    dat%=1000;
    data = dat / 100;
    data+=0x30;
    lcd_putch(data);

    dat%=100;
    data = dat / 10;
    data+=0x30;
    lcd_putch(data);

    dat%=10;
    data = dat + 0x30;
    lcd_putch(data);
}
```

LISTING PROGRAM MIKROKONTROLER SLAVE SENSOR

```
#define ADC_VREF_TYPE 0x60

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <compat/twi.h>

#include "twi_slave.h"
// #include "SPI.h"

volatile unsigned char TWI_buf[4];     // Transceiver buffer. Set the size in the header file
volatile unsigned char TWI_msgSize  = 0;           // Number of bytes to be transmitted.
volatile unsigned char TWI_state    = 0xF8;  // State byte. Default set to
TWI_NO_STATE.

uint8_t i=0;
volatile unsigned char
data_sensor[8],a[8],tengah[8]={94,76,101,152,143,95,100,153},sensor,PV;

unsigned char read_sensor(unsigned char num);
unsigned char read_adc(unsigned char adc_input);

int main (void)
{
        _delay_ms(500);
        //SPI_SlaveInit();
        TWAR = 0xD1;// 0x08;
        TWCR = 0x45;
        sei();
        ADMUX=ADC_VREF_TYPE & 0xff;
        ADCSRA=0x87;
        DDRC |= _BV(0); DDRC |= _BV(1); DDRC |= _BV(2);


        while (1)
        {

                for(i=0;i<8;i++){
                        data_sensor[i] = read_sensor(i);
                        if (data_sensor[i]>tengah[i]) a[i]=0;
                else a[i]=1;
                }

                sensor=(
(a[7]*128)+(a[6]*64)+(a[5]*32)+(a[4]*16)+(a[3]*8)+(a[2]*4)+(a[1]*2)+(a[0]*1) );
                switch(sensor){
              case 0b11111110:       // ujung kanan
                PV = 0;
                break;
              case 0b11111100:
```

```
                PV = 1;
                break;
            case 0b11111101:
                PV = 2;
                break;
            case 0b11111001:
                PV = 3;
                break;
                case 0b11111011:
                PV = 4;
                break;
            case 0b11110011:
                PV = 5;
                break;
                case 0b11110111:
                PV = 6;
                break;
            case 0b11100111:        // tengah
                PV = 7;
                break;
                case 0b11101111:
                PV = 8;
                break;
            case 0b11001111:
                PV = 9;
                break;
                case 0b11011111:
                PV = 10;
                break;
            case 0b10011111:
                PV = 11;
                break;
                case 0b10111111:
                PV = 12;
                break;
            case 0b00111111:
                PV = 13;
                break;
            case 0b01111111:        // ujung kiri
                PV = 14;
                break;
        }
            TWI_buf[0] = PV;


    }
}
/*
ISR(SPI_STC_vect){
    SPDR=PV;
}
*/
```

```
ISR(TWI_vect){
        static unsigned char TWI_bufPtr;

 switch (TWSR)
 {
   case TWI_STX_ADR_ACK:            // Own SLA+R has been received; ACK has been
returned
//    case TWI_STX_ADR_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as
Master; own SLA+R has been received; ACK has been returned
     TWI_bufPtr   = 0;                         // Set buffer pointer to first data location
   case TWI_STX_DATA_ACK:          // Data byte in TWDR has been transmitted;
ACK has been received
     TWDR = TWI_buf[TWI_bufPtr++];
     TWCR = (1<<TWEN)|                         // TWI Interface enabled
          (1<<TWIE)|(1<<TWINT)|                // Enable TWI Interupt and clear the flag
to send byte
          (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|          //
          (0<<TWWC);                           //
     break;
          case TWI_STX_DATA_NACK:        // Data byte in TWDR has been
transmitted; NACK has been received.
                              // I.e. this could be the end of the transmission.
     if (TWI_bufPtr == TWI_msgSize) // Have we transceived all expected data?
     {
//       TWI_statusReg.lastTransOK = TRUE;              // Set status bits to completed
successfully.
     }
     else                      // Master has sent a NACK before all data where sent.
     {
      TWI_state = TWSR;                        // Store TWI State as errormessage.
     }

     TWCR = (1<<TWEN)|                         // Enable TWI-interface and release TWI
pins
          (1<<TWIE)|(1<<TWINT)|                // Keep interrupt enabled and clear the
flag
          (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|          // Answer on next address
match
          (0<<TWWC);                           //

     //TWI_busy = 0;   // Transmit is finished, we are not busy anymore
     break;
   case TWI_SRX_GEN_ACK:            // General call address has been received; ACK
has been returned
//    case TWI_SRX_GEN_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as
Master; General call address has been received; ACK has been returned
//      TWI_statusReg.genAddressCall = TRUE;
   case TWI_SRX_ADR_ACK:            // Own SLA+W has been received ACK has been
returned
//    case TWI_SRX_ADR_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as
Master; own SLA+W has been received; ACK has been returned
```

```
                                    // Dont need to clear
TWI_S_statusRegister.generalAddressCall due to that it is the default state.
//      TWI_statusReg.RxDataInBuf = TRUE;
      TWI_bufPtr   = 0;                         // Set buffer pointer to first data location

                                    // Reset the TWI Interupt to wait for a new event.
      TWCR = (1<<TWEN)|                         // TWI Interface enabled
            (1<<TWIE)|(1<<TWINT)|               // Enable TWI Interupt and clear the flag
to send byte
            (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|    // Expect ACK on this
transmission
            (0<<TWWC);
      //TWI_busy = 1;

      break;
    case TWI_SRX_ADR_DATA_ACK:      // Previously addressed with own SLA+W;
data has been received; ACK has been returned
    case TWI_SRX_GEN_DATA_ACK:      // Previously addressed with general call;
data has been received; ACK has been returned
      TWI_buf[0]     = TWDR;//TWI_buf[TWI_bufPtr++]     = TWDR;
//      TWI_statusReg.lastTransOK = TRUE;         // Set flag transmission successfull.
                                    // Reset the TWI Interupt to wait for a new event.
      TWCR = (1<<TWEN)|                         // TWI Interface enabled
            (1<<TWIE)|(1<<TWINT)|               // Enable TWI Interupt and clear the flag
to send byte
            (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|    // Send ACK after next
reception
            (0<<TWWC);                  //
      //TWI_busy = 1;
      break;
    case TWI_SRX_STOP_RESTART:      // A STOP condition or repeated START
condition has been received while still addressed as Slave
                                    // Enter not addressed mode and listen to address
match
      TWCR = (1<<TWEN)|                         // Enable TWI-interface and release TWI
pins
            (1<<TWIE)|(1<<TWINT)|               // Enable interrupt and clear the flag
            (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|    // Wait for new address match
            (0<<TWWC);                  //

      //TWI_busy = 0;  // We are waiting for a new address match, so we are not busy

      break;
    case TWI_SRX_ADR_DATA_NACK:     // Previously addressed with own SLA+W;
data has been received; NOT ACK has been returned
    case TWI_SRX_GEN_DATA_NACK:     // Previously addressed with general call;
data has been received; NOT ACK has been returned
    case TWI_STX_DATA_ACK_LAST_BYTE: // Last data byte in TWDR has been
transmitted (TWEA = “0”); ACK has been received
//    case TWI_NO_STATE             // No relevant state information available; TWINT =
“0”
```

```
   case TWI_BUS_ERROR:        // Bus error due to an illegal START or STOP
condition
     TWI_state = TWSR;              //Store TWI State as errormessage, operation also
clears noErrors bit
     TWCR =   (1<<TWSTO)|(1<<TWINT);   //Recover from TWI_BUS_ERROR, this
will release the SDA and SCL pins thus enabling other devices to use the bus
     break;
   default:
     TWI_state = TWSR;                          // Store TWI State as errormessage,
operation also clears the Success bit.
     TWCR = (1<<TWEN)|                          // Enable TWI-interface and release TWI
pins
          (1<<TWIE)|(1<<TWINT)|                 // Keep interrupt enabled and clear the
flag
          (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|      // Acknowledge on any new
requests.
          (0<<TWWC);                            //

    //TWI_busy = 0; // Unknown status, so we wait for a new address match that might be
something we can handle
          }

}
unsigned char read_adc(unsigned char adc_input)
{
          ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
          // Delay needed for the stabilization of the ADC input voltage
          _delay_us(50);
          // Start the AD conversion
          ADCSRA|=0x40;
          // Wait for the AD conversion to complete
          while ((ADCSRA & 0x10)==0);
          ADCSRA|=0x10;
          return ADCH;
}

void selectora(int a,int b,int c)
{
          if(a)
                    PORTC |= _BV(0);
          else
                    PORTC &= ~_BV(0);
          if(b)
                    PORTC |= _BV(1);
          else
                    PORTC &= ~_BV(1);
          if(c)
                    PORTC |= _BV(2);
          else
                    PORTC &= ~_BV(2);
}
```

```
unsigned char read_sensor(unsigned char num){

        unsigned char temp=0;

        switch(num)
   {
        case 6: selectora(1,1,1);       break;
        case 5: selectora(1,1,0);       break;
      case 7: selectora(1,0,1);        break;
      case 4: selectora(1,0,0);        break;
      case 0: selectora(0,1,1);        break;
      case 3: selectora(0,1,0);        break;
      case 2: selectora(0,0,1);         break;
      case 1: selectora(0,0,0);        break;
        }
        temp = read_adc(3);
        return temp;
}
```

LISTING PROGRAM MIKROKONTROLER SLAVE MOTOR

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "SPI.h"
#include "twi_slave.h"

volatile int data_spi_rx;
volatile int data_spi_tx;
volatile uint8_t i, j=0,direction,k=0,terima=0,first=0;
volatile int rpm_actual[10],rpm_set_point[10],header[2];
volatile int PWM;

volatile unsigned char TWI_buf[4];     // Transceiver buffer. Set the size in the header file
volatile unsigned char TWI_msgSize  = 0;            // Number of bytes to be transmitted.
volatile unsigned char TWI_state    = 0xF8;  // State byte. Default set to
TWI_NO_STATE.

void setMotorSpeed(int speed);

int main (void)
{
        _delay_ms(5000);
        //SPI_SlaveInit();
        TWAR = 0xD3;// D3=kanan, D5=kiri
        TWCR = 0x45;
        sei();
        TCCR1A=0xA1;
        TCCR1B=0x0C;
        DDRB |= _BV(0);
        DDRB |= _BV(1);
        while (1)
        {
                PWM = (255*rpm_set_point[2]+rpm_set_point[1])/6;
                if(rpm_set_point[0]==0x0F)
                        setMotorSpeed(PWM );
                else if(rpm_set_point[0]==0xF0)
                        setMotorSpeed(-PWM);

        }
}
/*
ISR(SPI_STC_vect){
        data_spi_rx=SPDR;
        if(data_spi_rx==255)
                k++;
        else
                k=0;

        if(k>=2){
                j=0;
```

```
                terima=1;
        }
        else if(terima){
                rpm_set_point[j] = data_spi_rx;
                j++;
                if(j>=3)
                        terima=0;
        }

        SPDR=rpm_actual[j];//data_sensor[j];
}
*/
ISR(TWI_vect){
static unsigned char TWI_bufPtr;

switch (TWSR){
case TWI_STX_ADR_ACK:// Own SLA+R has been received; ACK has been returned
TWI_bufPtr   = 0;// Set buffer pointer to first data location
case TWI_STX_DATA_ACK:// Data byte in TWDR has been transmitted; ACK has
been received
TWDR = rpm_actual[TWI_bufPtr++];
TWCR = (1<<TWEN)|  // TWI Interface enabled
(1<<TWIE)|(1<<TWINT)|  // Enable TWI Interupt and clear the flag to send byte
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|
(0<<TWWC);
break;
case TWI_STX_DATA_NACK:  // Data byte in TWDR has been transmitted; NACK has
been received.
// I.e. this could be the end of the transmission.
if (TWI_bufPtr == TWI_msgSize) // Have we transceived all expected data?
{
//TWI_statusReg.lastTransOK = TRUE;// Set status bits to completed successfully.
}
else // Master has sent a NACK before all data where sent.
{
TWI_state = TWSR; // Store TWI State as errormessage.
}

TWCR = (1<<TWEN)|  // Enable TWI-interface and release TWI pins
(1<<TWIE)|(1<<TWINT)| // Keep interrupt enabled and clear the flag
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Answer on next address match
(0<<TWWC);
break;
case TWI_SRX_ADR_ACK: // Own SLA+W has been received ACK has been returned
TWI_bufPtr   = 0;  // Set buffer pointer to first data location

// Reset the TWI Interupt to wait for a new event.
TWCR = (1<<TWEN)|   // TWI Interface enabled
(1<<TWIE)|(1<<TWINT)|   // Enable TWI Interupt and clear the flag to send byte
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Expect ACK on this transmission
(0<<TWWC);
```

```
break;
case TWI_SRX_ADR_DATA_ACK: // Previously addressed with own SLA+W;
case TWI_SRX_GEN_DATA_ACK: // Previously addressed with general call;
rpm_set_point[TWI_bufPtr++]     = TWDR;
//      TWI_statusReg.lastTransOK = TRUE;     // Set flag transmission successfull.
// Reset the TWI Interupt to wait for a new event.
TWCR = (1<<TWEN)|   // TWI Interface enabled
(1<<TWIE)|(1<<TWINT)|    // Enable TWI Interupt and clear the flag to send byte
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Send ACK after next reception
(0<<TWWC);  //
break;
case TWI_SRX_STOP_RESTART:       // A STOP condition
// Enter not addressed mode and listen to address match
TWCR = (1<<TWEN)|          // Enable TWI-interface and release TWI pins
(1<<TWIE)|(1<<TWINT)|      // Enable interrupt and clear the flag
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|   // Wait for new address match
(0<<TWWC);

break;
case TWI_SRX_ADR_DATA_NACK: // Previously addressed with own SLA+W;
case TWI_SRX_GEN_DATA_NACK:  // Previously addressed with general call;
case TWI_STX_DATA_ACK_LAST_BYTE: // Last data byte in TWDR has been
transmitted
//    case TWI_NO_STATE   // No relevant state information available;
case TWI_BUS_ERROR:   // Bus error due to an illegal START or STOP condition
TWI_state = TWSR;   //Store TWI State as errormessage,
TWCR =   (1<<TWSTO)|(1<<TWINT);  //Recover from TWI_BUS_ERROR,
break;
default:
TWI_state = TWSR;    // Store TWI State as errormessage
TWCR = (1<<TWEN)|   // Enable TWI-interface and release TWI pins
(1<<TWIE)|(1<<TWINT)|   // Keep interrupt enabled and clear the flag
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|// Acknowledge on any new requests.
(0<<TWWC);
}
}

void setMotorSpeed(int speed){

   unsigned char reverse = 0;

   if (speed < 0){
      speed = -speed;    // make speed a positive quantity
      reverse = 1;    // preserve the direction
   }
   if(speed>10)
   speed-=10;
   else
   speed=0;

   if (speed > 0xFF)    // 0xFF = 255
```

```
        speed = 0xFF;

    if (reverse){
        PORTB &= ~_BV(0);
        OCR1A = speed;
    }
    else{    // forward
        PORTB |= _BV(0);
        OCR1A = 255 - speed;
    }

}
```