RANCANG BANGUN ROBOT *THREE-OMNI-WHEEL DIRECTIONAL* DENGAN SENSOR KOMPAS UNTUK MENJAGA HEADING ROBOT

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI ELEKTRONIKA

Diajukan Untuk Memenuhi Persyaratan
Memperoleh Gelar Sarjana Teknik

Disusun Oleh :

ARFAI FAHRUL KHIZAM

NIM. 135060300111060

KEMENTERIAN RISET TEKNOLOGI DAN PENDIDIKAN TINGGI

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2017

LEMBAR PENGESAHAN

RANCANG BANGUN ROBOT *THREE-OMNI-WHEEL DIRECTIONAL* DENGAN SENSOR KOMPAS UNTUK MENJAGA HEADING ROBOT

SKRIPSI

TEKNIK ELEKTRO KONSENTRASI TEKNIK ELEKTRONIKA

Ditujukan untuk memenuhi persyaratan

memperoleh gelar Sarjana Teknik

ARFAI FAHRUL KHIZAM
NIM. 135060300111060

Skripsi ini telah direvisi dan disetujui oleh dosen pembimbing

Pada tanggal 16 Agustus 2017

Dosen Pembimbing I	Dosen Pembimbing II
Adharul Muttaqin, S.T., M.T.	**Ir. Nurussa'adah, M.T.**
NIP. 19760121 200501 1001	**NIP. 19680706 199203 2 001**

Mengetahui,

Ketua Jurusan Teknik Elektro

M. Aziz Muslim, S.T., M.T., Ph.D
NIP. 19741203 200012 1 001

JUDUL SKRIPSI :

RANCANG BANGUN ROBOT *THREE-OMNI-WHEEL DIRECTIONAL* DENGAN SENSOR KOMPAS UNTUK MENJAGA HEADING ROBOT.

Nama Mahasiswa : Arfa'I Fahrul Khizam

NIM : 135060300111060

Program Studi : Teknik Elektro

Konsentrasi : Teknik Elektronika

KOMISI PEMBIMBING :

Ketua : Adharul Muttaqin, S.T, M.T. .....

Anggota : Ir. Nurussa'dah, M.T. .....

TIM DOSEN PENGUJI :

Dosen Penguji 1: Dr. Ir. M. Aswin, M.T. .....

Dosen Penguji 2: Eka Maulana, S.T., M.T., M. Eng. .....

Dosen Penguji 3: Dr. Ir. Ponco Siwindarto, M.Eng.Sc .....

Tanggal Ujian : 14 Agustus 2017

SK Penguji : No. 1092/UN10.6/SK/2017

PERNYATAAN ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya dan berdasarkan hasil penelusuran berbagai karya ilmiah, gagasan dan masalah ilmiah yang diteliti dan diulas dalam Naskah Skripsi ini adalah asli dari pemikiran saya. Tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu Perguruan Tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur jiplakan, saya bersedia skripsi dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, pasal 25 ayat 2 dan pasal 70).

Malang, 16 Agustus 2017

ARFA'I FAHRUL KHIZAM

NIM. 135060300111060

Ucapan Terimakasih Kepada :
Bapak dan Ibuku yang tercinta
Adikku Iftah yang kusayangi

Replace Fear Of The Unknown With
Curiousity - Penelope Ward

Everytime You Sigh, A Little Bit Of
Happines Escapes - Nisio Isin

RINGKASAN

Arfa'I Fahrul Khizam, Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya, 22 Juni 2017, *Rancang Bangun Robot Three-Omni-Wheel Directional Dengan Sensor Kompas Untuk Menjaga Heading Robot*. Dosen Pembimbing: Adharul Muttaqin, Nurussa'adah

Penggerak diferensial dua roda biasanya digunakan pada robot memiliki keterbatasan arah gerak. Untuk menyelesaikan masalah tersebut dapat digunakan sistem penggerak menggunakan roda Omni. Roda omni merupakan roda yang memiliki roll atau roda kecil lagi di sisi roda, karena itu roda omni dapat bergerak ke segala arah. Sistem penggerak yang memanfaatkan roda omni salah satunya adalah *Three Wheel Omni Directional*. Akan tetapi sistem penggerak ini masih memiliki kelemahan, yaitu kecepatan putar setiap roda tidak sama yang dapat disebabkan oleh algoritma pemrograman, kondisi motor yang sudah tua, ataupun *driver* motor yang digunakan untuk menggerakan motor, hal ini dapat menyebabkan arah hadap robot bergeser dan membuat robot bergerak ke arah yang tidak diinginkan. Untuk menyelesaikan masalah ini, penulis menggunakan sensor kompas CMPS-11. Saat robot mengalami slip, arah hadap robot akan berubah, dengan menggunakan sensor kompas, perubahan arah hadap robot dapat terditeksi dan dapat dibuat program untuk memperbaiki arah hadapnya. Berdasarkan penilitian, keluaran tegangan pada *driver* motor berbeda-beda meskipun input tegangan dan *duty cycle* sinyal PWMnya sama. Lalu pada sensor kompas CMPS-11, nilai yang diberikan juga tidak sesuai dengan arah hadap sebenarnya, nilainya pun tidak linier, maka dari itu penulis membuat persamaan perbandingan sederhana agar nilainya sesuai dengan arah hadap sesungguhnya. Setelah nilai CMPS-11 sudah sesuai dengan arah hadapnya, dibuat algoritma untuk memperbaiki arah hadap robot saat terjadi slip. Dan berdasarkan percobaan robot sudah dapat memperbaiki arah hadapnya meskipun masih terdapat sedikit kesalahan.

Kata kunci – *heading* robot, sensor kompas, *three-omni-wheel drive*

SUMMARY

Arfa'I Fahrul Khizam, *Department of Electrical Engineering, Faculty of Engineering, University of Brawijaya, 22 June 2017, Design and Build Three-Omni-Wheel Directional Robot With Compass Sensor To Keep Heading Robot. Academic Supervisor:* Adharul Muttaqin, Nurussa'adah.

The two-wheel differential driver usually used on the robot has a limited movement of direction. To solve the problem can be used drive system using Omni wheel. Omni wheels are wheels that have a small roll or wheel again on the sides of the wheel, therefore omni wheels can move in all directions. Drive system that utilizes the omni wheel one of them is the Three Wheel Omni Directional. However, this drive system still has a weakness, is the rotation speed of each wheel is not the same which can be caused by programming algorithm, old motor condition, or motor driver used to move the motor, this can cause the direction of the robot facing shift and make the robot move In an unwanted direction. To solve this problem, the author uses CMPS-11 compass sensor. When the robot is slipped, the direction of the robot will change, using the compass sensor, changes in the direction of the robot can be detected and can be made program to correct its direction. Based on the research, the output voltage in the motor driver is different even if the input voltage and duty cycle of PWM signal is same. Then on the CMPS-11 compass sensor, the value given is also not in accordance with the direction of the actual face, the value is not linear, therefore the author makes a simple comparison equation for its value in accordance with the direction of the real face. After the CMPS-11 values are in accordance with its direction, an algorithm is developed to improve the robot's facing direction when slip occurs. And based on robot experiments, the robots have been able to correct the direction of the face even though there are still a few errors.

Keywords - heading robot, compass sensor, three-omni-wheel drive

PENGANTAR

Puji syukur Penulis panjatkan kepada Tuhan Yang Maha Esa karena berkat limpahan rahmat dan hidayat-Nya Penulis dapat menyelesaikan Laporan Skripsi yang berjudul " Rancang Bangun Robot *Three-Omni-Wheel Directional* Dengan Sensor Kompas Untuk Menjaga Heading Robot". Laporan ini dibuat dengan tujuan untuk untuk memenuhi persyaratan memperoleh gelar Sarjana Teknik pada Jurusan Teknik Elektro Konsentrasi Teknik Elektronika Fakultas Teknik Universitas Brawijaya Malang.

Dalam penyusunan Laporan ini tidak sedikit hambatan yang penulis hadapi, namun penulis menyadari bahwa kelancaran dalam penyusunan Laporan ini berkat bantuan, dorongan, dan bimbingan dari berbagai pihak baik secara langsung maupun tidak langsung, untuk itu penulis menyampaikan terima kasih yang sebesar-besarnya kepada:

1. Kedua orang tua tercinta, M. Subakir dan Masrini yang selalu memberi kasih sayang dan memberi doa serta pengorbanan yang tiada henti. Adik yang baik, Iftah. Keluarga yang selalu menjadi tujuan pulang.
2. Yang terhormat Bapak M. Aziz Muslim, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya
3. Yang terhormat Ibu Nurussa'adah, S.T., M.T. selaku Ketua Kelompok Dosen Konsentrasi Elektronika Jurusan Teknik Elektro Universitas Brawijaya.
4. Yang terhormat Bapak Adharul Muttaqin, S.T., M.T. selaku dosen pembimbing 1 yang selalu memberikan bimbingan, arahan, dan motivasi dalam penyusunan Skripsi ini.
5. Yang terhormat Ibu Nurussa'adah, S.T., M.T. selaku dosen pembimbing 2 yang selalu memberikan dukungan, semangat, dan solusi dalam penyusunan Skripsi ini.
6. Teman-teman Tim Robot angkatan 2013 Hasdi, Oky, Chandra, Surya, Alec, Ekki, Achnafyan, Hemi, Andi, Dicka, Doni, Itsna, Hasyim, Yuda, dan Hanif atas dukungan dan bantuan yang telah diberikan.
7. Teman-teman TEUB tercinta terutama teman-teman Paket B Konsentrasi Teknik Elektronika yang selalu memberikan semangat, dorongan dan bantuan pikiran.
8. Semua pihak yang berperan langsung maupun tidak langsung dalam penyusunan skripsi ini.

Penulis berharap semoga laporan ini dapat memberikan manfaat dan dapat dijadikan referensi di masa yang akan datang. Penulis sadar bahwa laporan ini masih

banyak kekurangan dan jauh dari sempurna, oleh karena itu kritik dan saran yang membangun penulis harapkan demi kesempurnaan laporan ini.

Malang, 16 Agustus 2017

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1. Latar Belakang

Sistem penggerak tradisional dua roda yang biasanya digunakan pada robot mempunyai keterbatasan dalam kemampuan bergerak dan memakan waktu lebih untuk bergerak ke posisi yang diinginkan (F. Ribeiro, 2002). Hal ini terjadi karena robot yang menggunakan dua roda hanya bisa bergerak maju, mundur, dan rotasi. Agar robot mampu bergerak translasi ke segala arah, robot minimal harus memiliki tiga buah roda, dan roda yang digunakan adalah roda *omni* atau *mechanum*.

Setiap sistem pasti ada kekurangannya masing masing, kekurangan sistem *3-omni wheel directional* adalah kurangnya daya cengkram pada lantai, sehingga sewaktu-waktu dapat terjadi slip (F. Ribeiro, 2002). Selain kesalahan yang disebabkan oleh penggunaan jumlah roda, ada juga kesalahan yang ditimbulkan karena algoritma pengontrolan motor. Kesalahan yang disebabkan oleh algoritma pengontrolan biasanya adalah tidak setaranya kecepatan yang dihasilkan oleh setiap motor dan pada akhirnya menyebabkan slip dan arah robot akan bergeser dari arah yang diinginkan.

Berdasarkan penelitian sebelumnya, untuk mengatur arah gerak robot digunakan fungsi pergerakan sudut, sehingga pengguna hanya memeberi *set point* yang berupa kecepatan dalam satuan RPM dan arah gerak robot yang berupa arah sudut, lalu dari sudut yang diberikan mikrokontroler akan menghasilkan kecepatan yang diperlukan setiap motor agar robot dapat bergerak ke arah sudut yang sudah ditentukan sebelumnya.

Dengan sistem yang seperti ini, sebenarnya robot sudah dapat bergerak sesuai dengan nilai arah sudut yang diberikan, akan tetapi masih memiliki kesalahan. Hal ini dikarenakan tidak ada *feedback* yang memberikan nilai arah hadap robot sebenarnya.

Untuk menyelesaikan masalah di atas, salah satu cara adalah menggunakan sensor kompas yang dapat mendeteksi kesalahan arah hadap robot. Pada penulisan kali ini digunakan CMPS-11. Kesalahan hadap robot dapat dicari dengan cara mengambil nilai hadap robot sebelum robot bergerak sebagai *set point* dan nilai arah hadap robot yang sebenarnya adalah nilai hadap robot saat robot bergerak. Saat terjadi kesalahan(terdapat selisih antara nilai arah hadap robot saat bergerak dengan nilai arah hadap robot sebelum bergerak) maka robot akan menyesuaikan arah hadapanya sampai kesalahannya sekecil mungkin. Dengan menggunakan sistem seperti ini, diharapkan nantinya robot dapat bergerak ke arah yang diinginkan dengan kesalahan sekecil mungkin.

1.2. Rumusan Masalah

Berdasarkan uraian latar belakang di atas, dapat disusun rumusan masalah sebagai berikut :

- Bagaimana merancang robot *omni* beroda tiga agar dapat bergerak sesuai dengan arah yang diinginkan?
- Bagaiman respon kecepatan motor dengan *set point* yang diberikan?

- Bagaimana cara menditeksi kesalahan arah hadap robot?

1.3. Batasan Masalah

Karena banyaknya kemungkinan yang akan terjadi pada perancangan ini, maka penulis membatasi masalah yang akan dibahas dalam perancangan ini meliputi :

- robot yang akan dibuat menggunakan roda *omni,*
- robot hanya menggunakan tiga motor dan tiga roda,
- Data yang diolah adalah data langsung dari CMPS-11.

1.4. Tujuan

Berdasarkan rumusan masalah pada subbab sebelumnya, tujuan dari perancangan ini adalah mengimplementasikan sensor CMPS-11 sebagai sistem navigasi pada robot sehingga robot dapat bergerak sesuai dengan arah sudut yang diberikan dengan kesalahan sekecil mungkin.

1.5. Manfaat Penilitian

Manfaat dari penilitian ini adalah sebagai refrensi untuk Robot *Three-Wheel-Omni-Directional*, atau jika memungkinkan untuk seluruh robot yang membutuhkan sistem navigasi untuk pergerakannya, sehingga arah gerak yang dituju tidak jauh berbeda dengan arah gerak yang diinginkan.

BAB II
TINJAUAN PUSTAKA

2.1. *Omnidirectional Wheels*

Semua roda *Omnidirectional* memiliki prinsip dasar yang sama, yaitu roda utama menghasilkan traksi normal sesuai arah putaran motor, selain itu, roda omni juga dapat bergerak tanpa gesekan kearah motor. Agar dapat bergerak seperti itu, roda di buat menggunakan roda yang berukuran lebih kecil yang di letakan sepanjang pinggiran atau sisi luar roda utama (Rojas, 2005).

Gambar 2.1 *Omni-Directional Wheel*
sumber : Garcia-Sillas (2015)

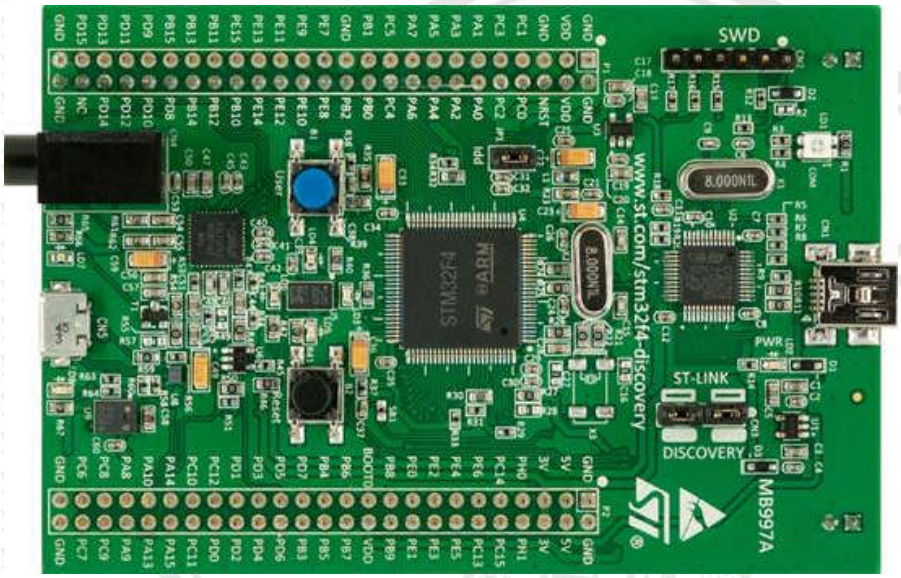
2.2. Mikrokontroller STM32F407VG

Pada perancangan ini, STM32F4 digunakan sebagai pusat control dan digunakan juga untuk mendapatkan *feedback* dari *planetary* dan data dari IMU-GY86. STM32F4 sendiri merupakan mikrokontroler dengan arsitektur ARM, yang memiliki performa lebih baik dari AVR, karena AVR memiliki set intruksi 16-bit RSIC, sedangkan ARM memiliki set intruksi 32-bit RSIC.

Fungsi-fungsi yang dimiliki STM32F4 antara lain :

- STM32F407VGT6 *microcontroller featuring* 32-bit ARM Cortex® -M4 dengan FPU core, 1-Mbyte *Flash memory*, 192-Kbyte RAM pada LQFP100 *package*.
- *On-board* ST-LINK/V2 pada STM32F4DISCOVERY atau ST-LINK/V2-A di STM32F407G-DISC1
- USB ST-LINK
- *Board power supply*:
  - *Through USB bus*,
  - *External power sources: 3 V and 5 V*.
- LIS302DL atau LIS3DSH ST MEMS *3-axis accelerome*
- *Two push buttons (user and reset)*
- *Clock, reset and supply management*
  - 1.8 V *to* 3.6 V *application supply and* I/Os
  - POR, PDR, PVD and BOR
  - 4-to-26 MHz *crystal oscillator*
- *Up to* 17 *timers*: *up to twelve* 16-bit *and two* 32- bit *timers up to* 168 MHz, *each with up to* 4 IC/OC/PWM *or pulse counter and quadrature (incremental).*

- *Up to* 15 *communication interfaces*
  - *Up to* 3 × I2C *interfaces* (SMBus/PMBus)
  - *Up to* 4 USARTs/2 UARTs (10.5 Mbit/s),
  - ISO 7816 *interface*, LIN, IrDA, *modem control*)
  - *Up to* 3 SPIs (37.5 Mbits/s), 2 *with muxed full-duplex* I2S *to achieve audio class accuracy via internal audio* PLL *or external clock,*
  - 2 × CAN *interfaces* (2.0B *Active*)
  - SDIO *interface*
- *Up to* 140 I/O *ports with interrupt capability*
  - *Up to* 136 *fast* I/Os *up to* 84 MHz
  - *Up to* 138 5V-*tolerant* I/Os.



Gambar 2.2 STM32F407VG Discovery
Sumber : STM32F4 Discovery Datasheet (2016:1)

2.3. Sistem Komunikasi I2C

Inter-integreted Circuit, yang merupakan akronim dari I2C merupaka sebuah protocol yang digunakan untuk memungkinkan lebih dari dua *slave* untuk berkomunikasi dengan satu atau lebih *master*, seperti SPI yang hanya digunakan untuk komunikasi jarak pendek, dan seperti UARTs yang menggunakan 2 kabel untuk bertukar informasi (Sparkfun, 2017).

Komunikasi menggunaka I2C lebih rumit jika dibandingkan dengan UART dan SPI. Karena pemberian sinyal harus mengikuti pada protocol tertentu untuk perangkat yang terhubung pada bus(kabel) untuk mengenali apakah perangkat tersebut valid atau tidak.

Berikut beberapa tahapan dalam komunikasi I2C :

- *Start Condition*

Untuk menginisialisasi alamat perangkat yang digunakan, SCL *master* diberi logika high dan SDA dalam kondisi low. Hal ini akan membuat perangkat *slave* mengetahui bahwa transmisi data akan segera dimulai (Sparkfun, 2017).

- *Address Frame*
  Alamat yang pertama dikirim adalah MSB yang diikuti oleh R/W bit yang mengindikasi apakah operasi *Read*(1) atau *Write*(0). *Bit* ke-9 pada *frame* merupakan bit NACK/ACK. Setelah 8 bit frame terkirim, perangkat *slave* akan memegang *control* pin SDA (Sparkfun, 2017).
- *Data Frames*
  Setelah alamat terkirim, pengiriman data dapat dimulai. Perangkat *master* akan tetap membangkitkan sinyal *clock* pada *interval regular*, dan data akan diletakan pada pin SDA (Sparkfun, 2017).
- *Stop Condition*
  Setelah semua data terkirim, *master* akan membangkitkan kondisi *Stop*. Kondisi *stop* ditandai dengan transisi SDA dari *low* ke *high*(0->1) setelah SCL transisi dari *low* ke *high*(0->1) di mana logika SCL akan tetap dalam kondisi *high* (Sparkfun, 2017).

2.4. *Rotary Encoder*

Rotary Encoder adalah suatu komponen elektro mekanis yang memiliki fungsi untuk memonitoring posisi sudut pada suatu poros yang berputar. Dari perputaran benda tersebut data yang termonitoring akan diubah ke dalam bentuk data digital oleh *rotary encoder* berupa lebar pulsa kemudian akan dihubungkan ke kontroler (Mikrokontroler/PLC). Berdasarkan data yang di dapat berupa posisi *anguler* (sudut) kemudian dapat diolah oleh kontroler sehingga mendapatkan data berupa kecepatan, arah, dan posisi dari perputaran porosnya. *Rotary Encoder* melacak gerakan poros motor untuk berbagai peralatan industri dan perangkat komersial.

Untuk aplikasi industri, *encoders* tambahan (digunakan ketika hanya posisi relatif yang diperlukan) biasanya digunakan dengan motor ac induksi. Sebaliknya, *encoders* mutlak (yang memberikan output biner yang berbeda di setiap posisi, sehingga posisi poros benar-benar tetap) sering dipasangkan dengan magnet permanen pada motor *brushless* di aplikasi servo. Seringkali, umpan balik *encoder* digunakan untuk memastikan sinkronisasi posisi stator motor dan rotor untuk pergerakan diterapkan ketika gulungan magnet rotor berada dalam kisaran posisi yang tepat (untuk memaksimalkan torsi).

2.5. Metode Kontroler *Proportional Integral Diferensial* (PID)

Metode pengontrolan yang mudah tidak selalu menghasilkan *output* yang baik. Pengontrolan yang lebih maju dan standar pada industry adalah *PID Controller*. Kontrol ini terdiri dari kontrol *proportional* , *integral*, dan kontrol *derivative* (Braunl, 2008).

2.5.1 Kontrol Proporsional

Untuk banyak aplikasi kontrol, perubahan mendadak antara nilai kontrol motor tetap dan nol tidak menghasilkan perilaku kontrol yang mulus. Kita dapat perbaiki ini dengan menggunakan istilah linier atau proporsional sebagai gantinya. Rumus untuk pengontrol proporsional (P kontroler) adalah:

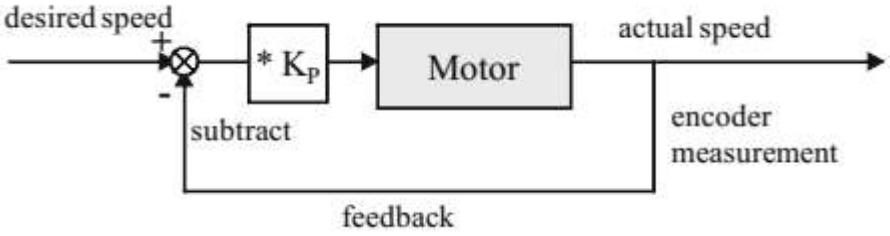
$$R(t) = K_P \cdot (v_{des}(t) - v_{act}(t)) \quad (1)$$

Keterangan:

V_{des} : Kecepatan yang diinginkan

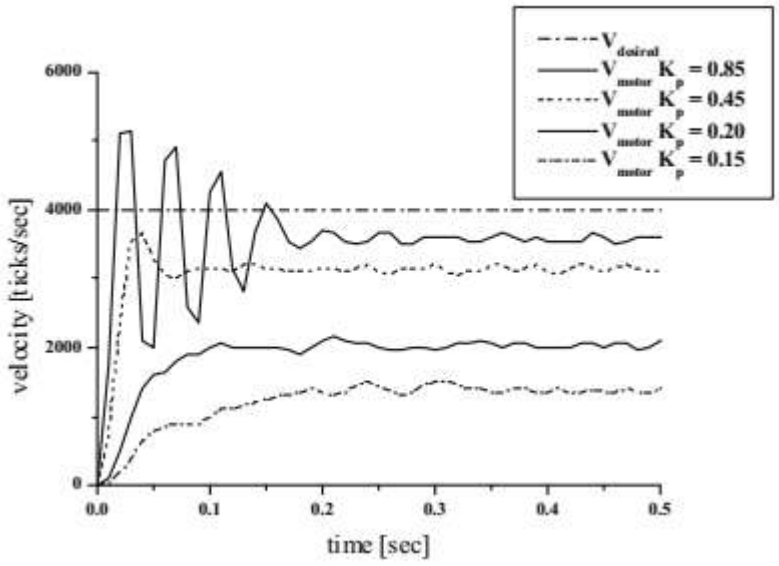
V_{act} : Kecepatan actual

Perbedaan antara nilai yang diinginkan dengan nilai aktualnya disebut "*error function*". Gambar 2.3 menunjukan skematik dari kontroler proporsional. Lalu pada gambar 2.4 ditunjukan bagaimana pengaruh nilai *gain proportional* (Kp) terhadap motor yang dikontrol. Semakin tinggi nilai Kp, respon semakin cepat, akan tetapi nilai yang semakin besar tersebut akan menghasilkan osilasi yang tidak diinginkan pada sistem (Braunl, 2008).



Gambar 2.3 *Proportional Controller*

Sumber : (Braunl, 2008)



Gambar 2.4 respon step untuk *proportional controller*

Sumber : (Braunl, 2008)

dan gambar 2.5 menunjukan *proportional controller* pada pemrograman Bahasa C/C++

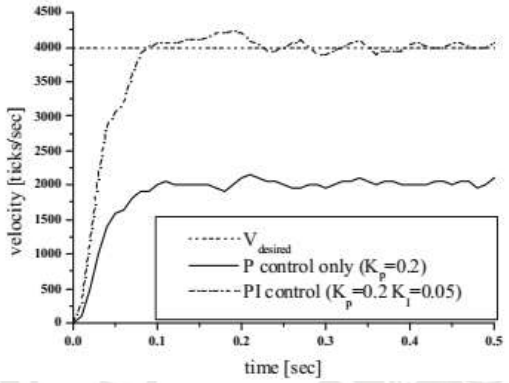
```
1   e_func = v_des - v_act;   /* error function */
2   r_mot  = Kp*e_func;       /* motor output */
```

Gambar 2.5 kode *P controller*

Sumber : (Braunl, 2008)

2.5.2 Kontrol Integral

Tidak seperti kontroler P, kontroler I (*Integral Control)* sangat jarang digunakan sendiri, biasanya kontroler ini dipadukan dengan kontroler P atau kontroler PD. Penggunakan kontroler I ini bertujuan untuk menghilangkan atau mengurangi *error steady-state* pada kontroler P. kontroller I yang baik dapat mengurangi *error steady-state* sampai dengan 0 seperti yang ditunjukan pada gambar 2.6



Gambar 2.6 Respon Step untuk Kontroler I

Sumber : (Braunl, 2008)

Lalu untuk pemrograman Bahasa C dapat dilihat pada gambar 2.7 di bawah ini

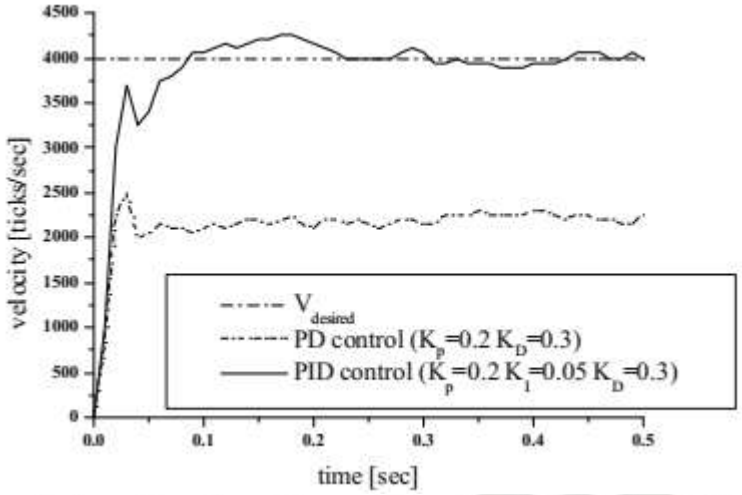
```
1   static int r_old=0, e_old=0;
2   ...
3   e_func = v_des - v_act;
4   r_mot  = r_old + Kp*(e_func-e_old) + Ki*(e_func+e_old)/2;
5   r_mot = min(r_mot, +100);  /* limit output */
6   r_mot = max(r_mot, -100);  /* limit output */
7   r_old  = r_mot;
8   e_old  = e_func;
```

Gambar 2.7 Kode pemrograman untuk Kontroler PI

Sumber : (Braunl, 2008)

2.5.3 Kontrol *Derivative*

Sama seperti kontroler I, kontroler D (*derivative controller*) juga jarang digunakan sendiri, kontroler ini biasanya digabungkan dengan kontroler P ataupun kontroler PI. Maksud dari penggunaan kontroler D adalah untuk mempercepat respon kontrol P berdasarkan berubahan masukannya. Gambar 2.8 menunjukan perbedaan respon step antara kontroler P, kontroler PI, dan kontroler PID, dan gambar 2.9 menunjukan pemrograman dalam Bahasa C untuk kontrole PID (Braunl, 2008)



Gambar 2.8 respon step kontroler PID

Sumber : (Braunl, 2008)

```
1   static int r_old=0, e_old=0, e_old2=0;
2   ...
3   e_func = v_des - v_act;
4   r_mot  = r_old + Kp*(e_func-e_old) + Ki*(e_func+e_old)/2
5                  + Kd*(e_func - 2* e_old + e_old2);
6   r_mot = min(r_mot, +100);  /* limit output */
7   r_mot = max(r_mot, -100);  /* limit output */
8   r_old  = r_mot;
9   e_old2 = e_old;
10  e_old  = e_func;
```

Gambar 2.9 Pemrograman kontroler PID

Sumber : (Braunl, 2008)

2.5.4 PID *Parameter Tuning*

Tuning ketiga parameter PID Kp, Ki, dan Kd merupakan hal yang penting. Petunjuk berikut dapat digunakan untuk mendapatkan nilai yang baik :

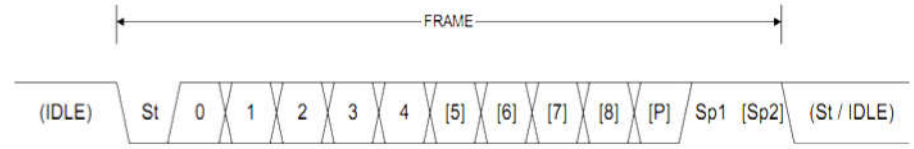
1. Tentukan berapa nilai *set point*, lalu matikan fungsi kontroler I dan D. setelah itu naikan nilai Kp sampai terjadi osilasi.
2. Jika sistem osilasi, bagi Kp dengan 2,
3. Naikan Kd dan amati saat menaikan atau menurunkan nilai *set point* sekitar 5%, pilih nilai Kd yang menghasilkan respon teredam.
4. Naikan nilai Ki sampai terjadi osilasi, lalu bagi Ki dengan 2 atau 3 (Braunl, 2008).

2.6. Komunikasi Serial

Pada prinsipnya, komunikasi serial ialah komunikasi di mana pengiriman data dilakukan per bit, sehingga lebih lambat dibandingkan komunikasi paralel seperti pada port printer yang mampu mengirim 8 bit sekaligus dalam sekali waktu. Devais pada komunikasi serial port dibagi menjadi 2 (dua) kelompok yaitu *Data Communication Equipment* (DCE) dan *Data Terminal Equipment* (DTE). Pengiriman data secara serial

dapat berupa sinkron, yaitu pengiriman *clock* dilakukan bersamaan dengan data, atau berupa asinkron, yaitu pengiriman *clock* tidak bersamaan dengan data namun secara dua tahap, saat data dikirim dan data diterima. Untuk istilah yang sering digunakan untuk mengirim dan menerima data secara asinkron biasa disebut *Universal Asynchronous Receiver Transmitter* (UART). Komunikasi data serial menggunakan UART sangat umum dan mudah penggunaannya, misalnya pada port serial PC. Pada UART jalur pengiriman dan penerimaan data serial dipisahkan.

Setiap pengiriman data pada UART menggunakan bit tanda *start* bit dan *stop* bit. Jalur data yang digunakan hanya satu untuk setiap pengiriman data. Data-data serial dikirim melewati jalur data satu persatu setiap satuan waktu. Format pengiriman data serial secara asinkron ditunjukkan dengan Gambar 2.16.

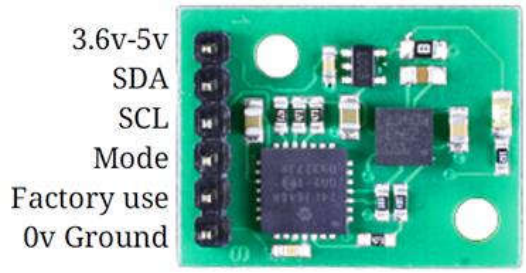


Gambar 2.10 Format Frame Data Serial USART
Sumber: Atmel, 2007:137

2.7. Sensor Kompas

Kompas merupakana sensor yang sangat berguna pada robot, terutama untuk *self-localization.* Robot otomatis membutuhkan sensor yang terpasang pada badan robot untuk memantau keadaan posisi dan orientasinya. Metode standar yang dapat digunakan adalah dengan menggunakan *rotary encoder* yang terpasang di setiap roda. Akan tetapi karena adanya slip pada roda, maka metode ini kurang baik untuk digunakan. Maka dari itu, menggunakan sensor kompas untuk memantau posisi dan orientasi robot adalah hal yang baik untuk dilakukan (Braunl, 2008).

Pada perancangan ini, digunakan modul sensor IMU CMPS11. CMPS-11 adalah modul sensor IMU yang terdiri dari 3-sumbu *magnetometer,* 3-sumbu *gyrometer,* dan 3-sumbu *accelerometer.* Modul ini dapat diakses oleh mikrokontroller menggunakan dua jenis antar-muka, yaitu I2C dan UART.



Gambar 2.11 Modul Sensor Kompas CMPS-11
Sumber : Dokumentasi Penulis

Modul IMU ini menggunakan *chip* sensor LSM9DS0 yang memiliki fitur sebagai berikut

- 3 kanal akselerasi, 3 kanal tingkatan sudut, dan 3 kanal medan magnet,
- ±2/±4/±6/±8/±16 *g linear acceleration full scale*,
- ±2/±4/±8/±12 *gauss magnetic full scale*,
- ±245/±500/±2000 dps *angular rate full scale*,
- 16-bit data *output,*
- Antar-muka I2C atau SPI,
- Tegangan input antara 2.4V sampai 3.6V,
- Mode *low-power*,
- *Programmable Interupt Generators,*
- *Embedded Self-Test,*
- Sensor Suhu,
- *Embedded* FIFO (STMicroelectronics, 2013).

Kelebihan yang dimiliki oleh modul IMU ini dibanding modul IMU yang lain adalah adanya mode kalibrasi untuk menghilangkan penguatan sensor dan *offset* dari *magnetometer* dan *accelerometer*.

Selain adanya mode kalibrasi, modul ini juga memiliki keluaran *pitch* dan *roll* yang sudah ter-kalman-filter yang mnggunakan sensor *gyrometer* untuk mengurangi kesalahan yang disebabkan oleh efek akselerasi yang tidak diinginkan, seperti goncangan dari modul PCB. Selain itu, sensor ini juga dapat memberikan nilai *yaw* dengan resolusi 0.1.

2.8. Motor DC *Planetary*

Motor DC adalah adalah motor listrik yang memerlukan suplai tegangan arus searah pada kumparan medan untuk diubah menjadi energi gerak mekanik. Kumparan medan pada motor dc disebut *stator* (bagian yang tidak berputar) dan kumparan jangkar disebut *rotor* (bagian yang berputar). Motor arus searah, sebagaimana namanya, menggunakan arus langsung yang tidak langsung/*direct-unidirectional* (Purnama, 2012). Motor DC memiliki 3 bagian atau komponen utama untuk dapat berputar sebagai berikut.

Bagian atau komponen utama Motor DC yaitu:

- Kutub medan

  Motor DC sederhana memiliki dua kutub medan: kutub utara dan kutub selatan. Garis magnetik energi membesar melintasi ruang terbuka diantara kutub-kutub dari utara ke selatan. Untuk motor yang lebih besar atau lebih komplek terdapat satu atau lebih elektromagnet (Purnama, 2012).

- *Current Elektromagnet* atau Dinamo

Dinamo yang berbentuk silinder, dihubungkan ke as penggerak untuk menggerakan beban. Untuk kasus motor DC yang kecil, dinamo berputar dalam medan magnet yang dibentuk oleh kutub-kutub, sampai kutub utara dan selatan magnet berganti lokasi (Purnama, 2012).

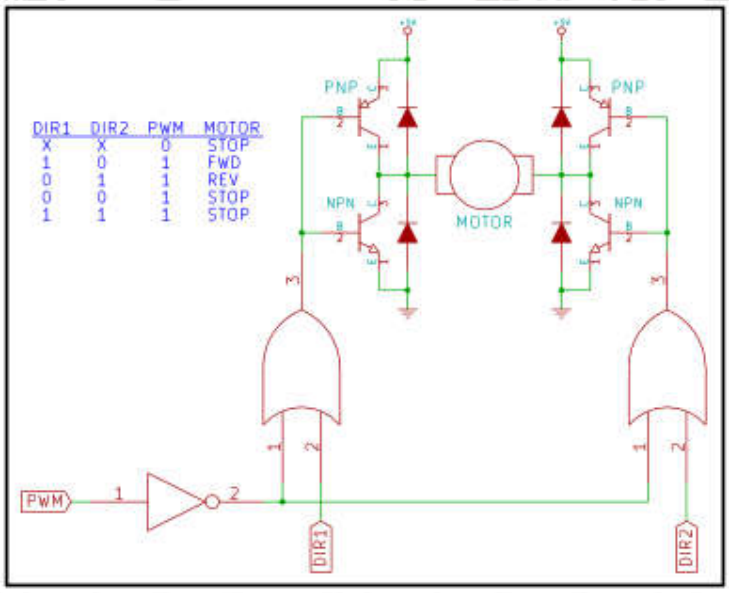
- *Commutator*

Komponen ini terutama ditemukan dalam motor DC. Kegunaannya adalah untuk transmisi arus antara dinamo dan sumber daya (Purnama, 2012).

2.9. *Driver* Motor

Pada dasarnya, keluaran dari mikrokontroller tidak dapat mengendalikan motor secara langsung, makadari itu, penggunaan *driver* motor dibutuhkan dalam perancangan ini.

Driver umum yang biasanya digunakan untuk mengendalikan motor DC adalah *driver H-bridge*. *H-bridge* adalah sebuah perangkat keras berupa rangkaian yang berfungsi untuk menggerakkan motor. Rangkaian ini diberi nama *H-bridge* karena bentuk rangkaiannya yang menyerupai huruf H seperti pada gambar 9 di bawah ini



Gambar 2.11 *Driver H-Bridge*

Sumber : http://nathandumont.com/blog/h-bridge-tutorial

Pada perancangan kali ini, penulis menggunakan modul *driver* L298N. L298 dapat digunakan pada tegangan tinggi dan memiliki keluaran arus sampai dengan 2 A setiap *channel*-nya dengan ketahanan terhadap suhu yg cukup tinggi. Tabel 2.1 menunjukkan *absolute maximum rating* dari L298N. Gambar 2.9 menunjukkan *pin* konfigurasi dari L298.

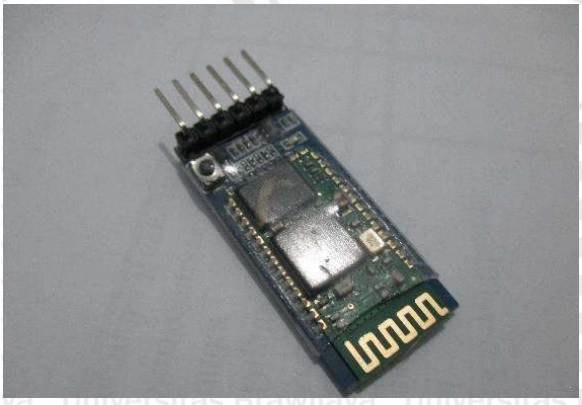
Tabel 2.1. *Absolute Maximum Rating Pin* IC L298N

Symbol	Parameter	Value	Unit
V_S	*Power supply*	50	V
V_{SS}	*Logic Supply Voltage*	7	V
V_I, V_{en}	*Input and Enable Voltage*	-0.3 to 7	V
I_O	*Peak Output Current (Each Channel)*		
	-Non Repetitive	3	A
	-Repetitive (80% on -20% off; t_{on} = 10ms)	2.5	A
	-DC Operation	2	A
V_{sens}	*Sensing Voltage*	-1 to 2.3	V
P_{tot}	*Total Power Dissipation* (T_{case} = 75˚C)	25	W
T_{op}	*Junction Operation Temperature*	-25 to 130	C
T_{stg}, T_j	*Storage and Junction Temperature*	-40 to 150	C

2.10 Modul *Bluetooth* HC-05

Bluetooth Module HC-05 merupakan module komunikasi nirkabel pada frekuensi 2.4GHz dengan pilihan koneksi bisa sebagai slave, ataupun sebagai master. Dapat digunakan dengan mikrokontroler untuk membuat aplikasi wireless bentuk fisik modul bluetooth HC-05 ditunjukkan dalam Gambar 2.8. Adapun spesifikasinya adalah sebagai berikut:

- Frekuensi : 2.4GHz,
- Catu Daya : 5 V atau 3,3 V DC,
- Dimensi : 3.57cm x 1.52cm.



Gambar 2.12 Modul Bluetooth HC-05
sumber : dokumentasi penulis

BAB III
METODE PENILITIAN

Untuk menyelesaikan rumusan masalah dan merealisasikan tujuan penelitian maka diperlukan langkah-langkah metode untuk menyelesaikan masalah tersebut. Adapun langkah-langkah yang perlu dilakukan untuk merealisasikan sistem yang dirancang adalah penentuan studi literatur, spesifikasi alat, perancangan dan pembuatan alat, pengujian alat, dan pengambilan kesimpulan.

Sebelum menyusun langkah-langkah pembuatan alat, ada baiknya terlebih dahulu menentukan target sistem yang akan dibuat, sehingga alat yang akan dibuat jelas sistem dan fungsinya, berikut merupakan target sistem yang akan dibuat :

1. Robot dapat bergerak ke segela arah dengan menggunakan sistem *Three-Omni Directional*.
2. Robot dapat mengetahui arah hadap robot yang sebenarnya menggunakan sensor *Accelerometer* dan *Magnetometer* yang terdapat pada modul CMPS-11.
3. Menggunakan aplikasi android untuk memberikan *Set point* dan memantau arah hadap robot.

3.1. Penentuan Spesifikasi Alat

Spesifikasi alat secara keseluruhan ditetapkan terlebih dahulu sebagai acuan dalam perancangan selanjutnya. Spesifikasi alat yang direncanakan adalah sebagai berikut:

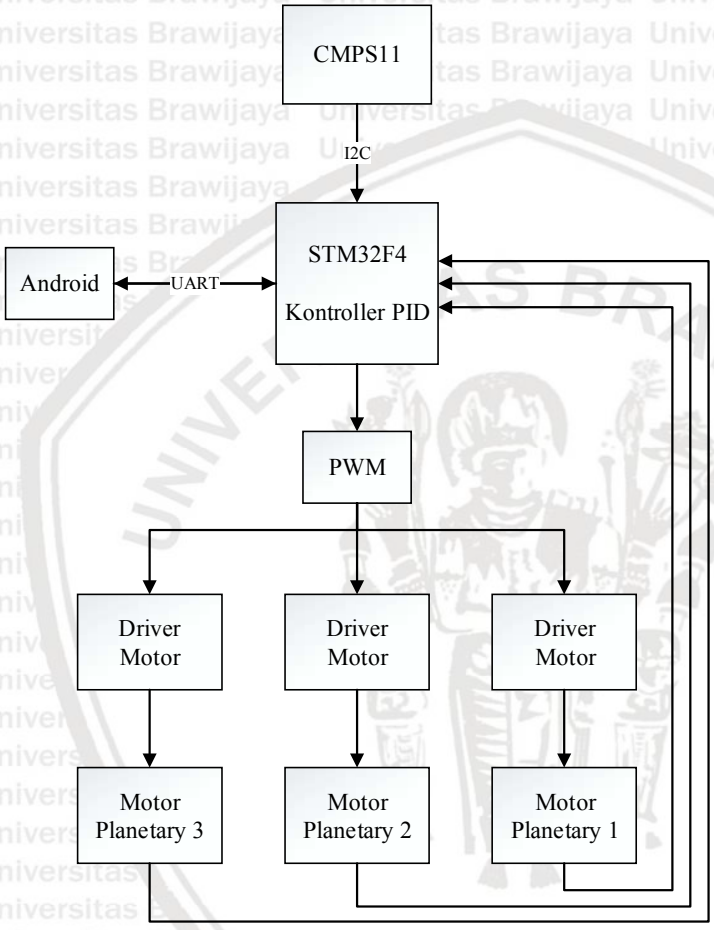
1. Mekanika robot berbahan dasar PCB dan mika *acrylic.*
2. Dimensi robot berbentuk lingkaran dengan diameter 25cm.
3. Menggunakan 3 buah motor DC sebagai actuator.
4. Menggunakan 3 buah roda *Omni* dengan diameter 5cm.
5. Menggunakan IMU CMPS-11 untuk mendapatkan nilai heading yang sebenarnya.
6. Menggunakan sistem *omni-kinematics* untuk menggerakan robot.
7. Driver motor yang digunakan adalah sebuah L298N H-bridge untuk menggerakkan dua buah motor DC.
8. Menggunakan mikrokontroller STM32F4.
9. Menggunakan modul *Bluetooth* HC-05 untuk komunikasi robot dengan android.
10. Menggunakan 1 sumber tegangan 12V dari baterai lippo.
11. Menggunakan modul regulator switching LM2576 *simple switcher* untuk memberikan tegangan 5V ke STM32F4, HC-05, dan CMPS-11.

3.2. Perancangan dan Pembuatan Alat

Dalam perancangan kali ini, yang pertama dilakukan penulis adalah membuat sebuah blok diagram sistem kesulurhan, setelah itu membuat spesifikasi mekanik robotnya. Dan yang terakhir adalah perancangan bagian *software.*

3.2.1. Perancangan blok diagram kesuluruhan

Pada perancangan alat diperlukan perencanaan blok diagram sistem yang dapat menjelaskan sistem secara garis besar dan diharapkan alat dapat bekerja sesuai dengan rencana



Gambar 3.1 Diagram Blok Sistem

Fungsi dari masing-masing bagian dalam blok ini adalah sebagai berikut :

1. CMPS-11 yang digunakan untuk memantau *heading* robot.
2. Pada perancangan ini, perancang menggunakan android sebagai kontrolernya. Yang di mana pengiriman data dari android ke mikrokontroler STM32F4 menggunakan koneksi *Bluetooth.* Data yang dikirim berupa arah gerak robot yang diingnkan. Dan mikrokontroler juga akan mengirim data ke android berupa arah gerak robot sebenarnya yang didapat dari CMPS-11.

3. STM32F4 sebagai mikrokontroler utama digunakan untuk mendapatkan data dari CMPS-11 dan data arah gerak dari *user* serta mengolahnya dengan fungsi kontroler PID untuk dijadikan PWM, yang di mana PWM akan digunakan untuk menggerakan motor.
4. Android digunakan sebagai media *user* untuk mengendalikan robot serta memonitoring arah gerak robot.
5. Motor digunakan sebagai actuator utama robot.
6. Dari motor *planetary* akan menghasilkan keluaran yang berupa data dari *rotary encoder* yang akan menjadi *feedback* untuk mikrokontroler.

Prinsip kerja sistem ini adalah awalnya robot akan menerima data dari user yang berupa arah gerak robot. Lalu pada mikrokontroler data tersebut akan diolah dan dijadikan PWM untuk masing-masing motor sehingga robot akan bergerak ke arah yang sudah ditentukan sebelumnya. Saat robot bergerak, mikrokontroler juga menerima data dari CMPS-11, pada mikrokontroler data dari IMU akan diolah dan dijadikan nilai *heading* aktual robot, lalu nilai *heading* akan dikirim ke *user* untuk *monitoring* serta diolah menggunakan sistem PID untuk menjaga arah gerakan robot sesuai dengan arah robot yang diinginkan *user*. Selain menerima data dari IMU, mikrokontroler juga menerima data dari *rotary encoder,* yang di mana data tersebut dapat diolah sehingga menghasilkan nilai kecepatan putaran roda dalam satuan *Radian/minute* (RPM). Lalu data RPM juga akan di kontrol dengan PID sehingga kecepatan putaran motor akan setabil.

3.2.2. Perancangan Mekanika Robot

Sistem mekanika yang baik, akan mendukung pergerakan robot menjadi lebih baik pula, oleh sebab itu perancangan mekanik dalam hal ini bodi dan rangka robot haruslah proporsional dengan berat, panjang dan lebar serta tinggi dari robot. Agar robot memiliki beban yang tidak terlalu berat maka untuk perancangan dan pembuatan perangkat keras mekanika dan elektrik di desain menjadi satu kesatuan.

Untuk rincian dimensi robot adalah sebagai berikut :

- Diameter robot = 25cm
- Tinggi robot maksimum = 15cm.

3.2.3. Perancangan Perangkat Lunak

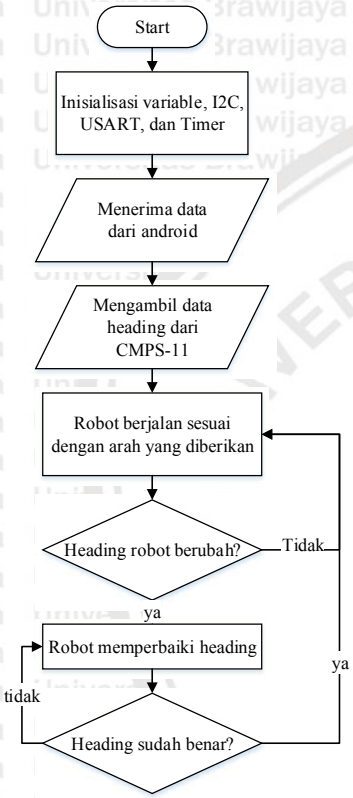
Perancangan *Software* dilakukan dengan merancang diagram alir (*flowchart*) terlebih dahulu. Diagram alir ini berfungsi sebagai alur kerja untuk setiap perangkat keras yang dikendalikan oleh mikrokontroler STM32 Nucleo serta proses-proses perhitungan yang dikerjakan oleh mikrokontroler. Bahasa pemrograman yang digunakan dalam program utama adalah bahasa C dengan menggunakan *compiler CooCox*.

3.2.3.1. *Flowchart* sistem kesuluruhan

Secara umum, saat robot dinyalakan, hal yang pertama dilakukan adalah mengkonfigurasi fungsi I2C, PWM, serta pin-pin yang digunakan. Setelah konfigurasi

selesai, Selanjutnya adalah menerima data dari android/*user* yang berupa arah gerak robot yang diinginkan, lalu data yang diterima tersebut diolah di mikrokontroler dan dijadikan PWM untuk mengendalikan kecepatan motor serta mengontrol dua pin keluaran untuk mengendalikan arah putaran motornya.

Selagi robot berjalan, mikrokontroler akan menerima data dari modul CMPS-11 dan data yang dihasilkan oleh *rotary encoder*. Data dari CMPS-11 akan diolah untuk menjaga *heading* robot, sedangkan data yang dihasilkan oleh *rotary encoder* akan diolah menjadi kecepatan putaran roda dan digunakan untuk menjaga kestabilan kecepatan putaran roda. Untuk lebih jenisnya dapat dilihat pada diagram alir di bawah ini.



Gambar 3.2 Diagram alir keseluruhan pada mikrokontroler

3.2.3.2. Perancangan Rangkaian Antarmuka Mikorkontroler Utama

Pada perancangan alat ini, digunakan STM32F4 sebagai mikrokontroler utama, berikut perancangan pin-pin yang digunakan

PIN A.0 = dihubungkan dengan pin ch.a *rotary encoder* motor1

PIN A.1 = dihubungkan dengan pin ch.a *rotary encoder* motor2

PIN A.2 = dihubungkan dengan pin ch.a *rotary encoder* motor3

PIN B.0 = dihubungkan dengan pin ch.b *rotary encoder* motor1

PIN B.1 = dihubungkan dengan pin ch.b *rotary encoder* motor2

PIN B.3 = dihubungkan dengan *input 4 driver* motor L298N pertama

PIN B.4 = dihubungkan dengan pin ch.b *rotary encoder* motor3

PIN B.6 = dihubungkan dengan pin SCL CMPS-11

PIN B.7 = dihubungkan dengan pin SDA CMPS-11

PIN B.10 = dihubungkan dengan pin TX HC-05

PIN B.11 = dihubungkan dengan pin RX HC-05

PIN B.15 = dihubungkan dengan *input 1 driver* motor L298N pertama

PIN D.1 = dihubungkan dengan *input 1 driver* motor L298N kedua

PIN D.3 = dihubungkan dengan *input 2 driver* motor L298N kedua

PIN D.7 = dihubungkan dengan *input 3 driver* motor L298N pertama

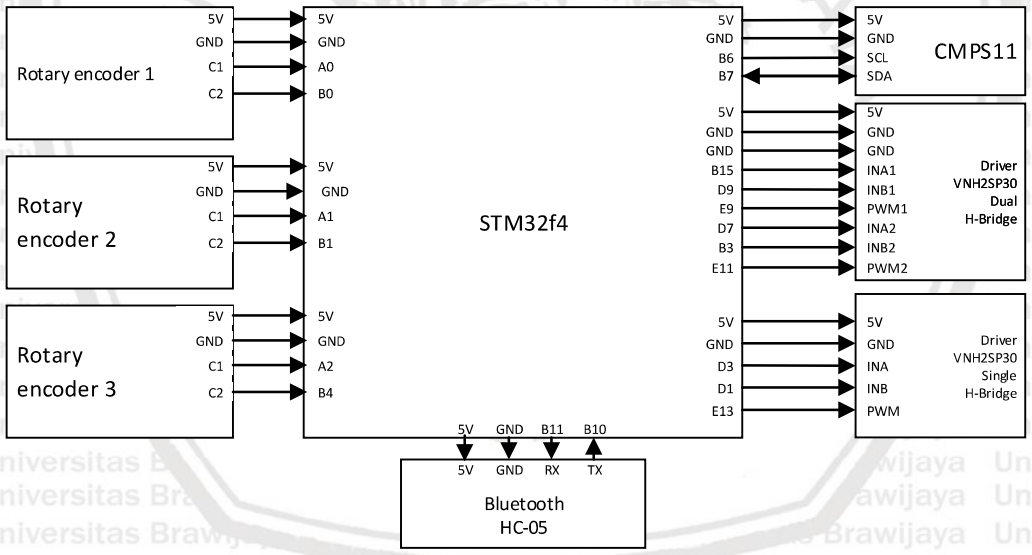
PIN D.9 = dihubungkan dengan *input 2 driver* motor L298N pertama

PIN E.9 = dihubungkan dengan *ENABLE* A *driver* L298N pertama

PIN E.11 = dihubungkan dengan *ENABLE* B *driver* L298N pertama

PIN E.13 = dihubungkan dengan *ENABLE* A *driver* L298N kedua

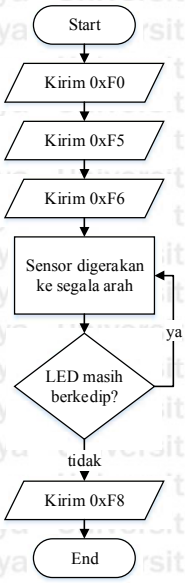
Adapun diagram antarmuka mikorkontroler utama secara kesuluruhan dalam gambar 13.



Gambar 3.3 Diagram Blok Antarmuka Mikrokontroler Utama Secara Keseluruhan

3.2.3.3. Kalibrasi Sensor Kompas CMPS-11

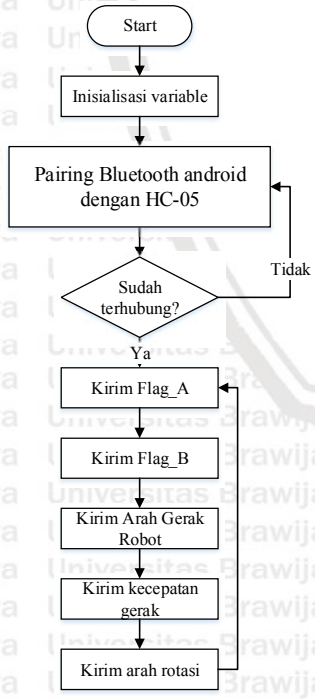
Sebelum sensor digunakan, sensor ini harus terlebih dahulu di kalibrasi untuk mengatur ulang penguatan sensor dan offset dari *magnetometer* dan *accelerometer*. Pertama-tama hal yang harus dilakukan adalah masuk ke mode kalibrasi sensor dengan mengirimkan urutan 3-*byte* yang terdiri dari 0xF0, 0xF5, dan 0xF6, setelah itu sensor digerakan ke segala arah, jika LED sudah tidak berkedip lagi, itu menandakan proses pengambilan data kalibrasi sudah selesai, dan selanjutnya adalah keluar dari mode kalibrasi dengan mengirim data 0xF8. Agar lebih jelasnya dapat dilihat pada gambar 14 di bawah ini yang merupakan diagram alir kalibrasi sensor kompas CMPS-11



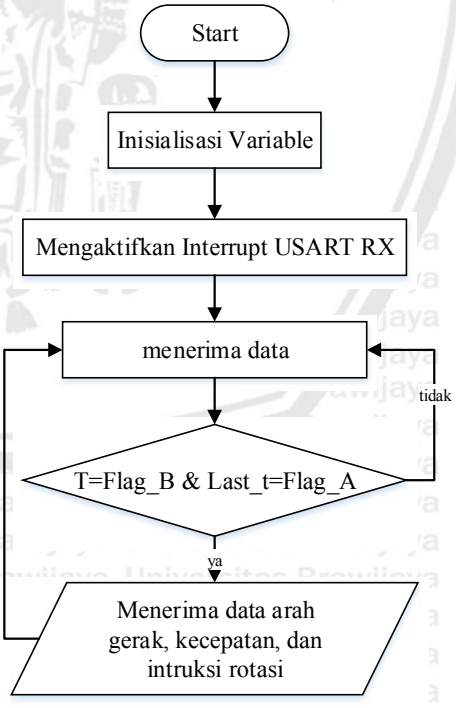
Gambar 3.4 Diagram alir kalibrasi sensor kompas CMPS-11

3.2.3.4. Perancangan Komunikasi Android dengan STM32F4

Proses komunikasi antara android dengan STM32F4 terbagi menjadi dua proses, yaitu proses pengiriman data pada android, dan proses penerimaan data pada STM32F4. Gambar 14 (a) dan (b) di bawah ini merupakan diagram alir proses pengiriman data pada android dan penerimaan data pada STM32F4



(a) proses pengiriman data



(b) proses penerimaan data

Gambar 3.5 Proses pengiriman (a) dan penerimaan (b) data

dari gambar 14 di atas, diketahui data yang pertama dikirim oleh android adalah data FLAG_A dan disusul oleh FLAG_B, setelah kedua data tersebut dikirim, barulah data

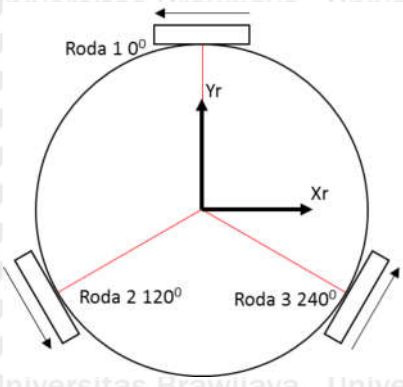
bergerakan robot dikirm. Pada STM32F4, proses penerimaan data merupakan sebuah looping yang akan terus berjalan selama STM32F4 menerima data. Pada siklus pertama, *variable* T akan menerima data FLAG_A dan *variable* LAST_T akan menyimpan data T yang nilainya adalah FLAG_A, lalu pada siklus kedua, nilai T akan berubah menjadi FLAG_B, saat T=FLAG_B dan LAST_T=FLAG_A, maka proses penerimaan data untuk pergerakan data dimulai, cara ini digunakan agar tidak ada pergeseran data untuk pergerakan robot. Gambar 15 di bawah ini menunjukan *source code* untuk proses penerimaan data

```
void USART3_IRQHandler(void){
    if( USART_GetITStatus(USART3, USART_IT_RXNE) ){
        char t = USART3->DR; // the character from the USART3 data register is saved in t
        char static last_data;
        Pin(GPIOD,TO,P_15);
        tanda_BT2++;
        switch(angka1)
        {
            case 0 : vxy=USART3->DR; angka1+=1; break;
            case 1 : GetSudut_v=USART3->DR; angka1+=1; break;
            case 2 : kec_w=USART3->DR; angka1+=1; break;
            case 3 : SetAksel_v=USART3->DR; angka1+=1; break;
            case 4 : SetAksel_v2=USART3->DR; angka1+=1; break;
            case 5 : re_center_flg=USART3->DR; angka1=100; break;
        }
        if(t==170&&last_data==85)angka1=0;
        last_data=t;
        if(kec_w == 40)kec_w= 100;
        else if(kec_w == 216)kec_w= -100;

        GetAksel_v = SetAksel_v+SetAksel_v2;
    }

}
```

Gambar 3.5 *Source Code* proses penerimaan data pada STM32F4

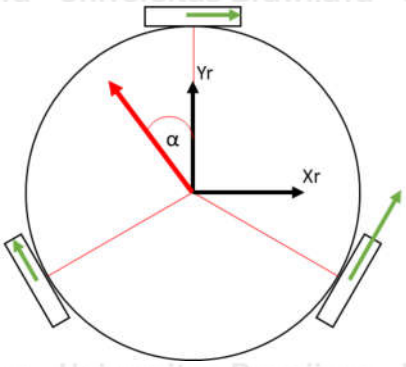
3.2.3.5.Persamaan untuk Memberi Nilai Kecepatan dan Arah Putaran Motor

Untuk memberi nilai kecepatan putaran dan arah putaran motor pada robot *omni* beroda tiga, pada perancangan ini digunakan rumus yang bersumber dari jurnal "*Three Omni-Directional Wheels Control On A Mobile Robot*" yang dibuat oleh F. Riberio dkk. Pada jurnal tersebut, setiap roda diletakan sedemikian rupa yang dimana titik rotasinya atau pusat roda menghadap ke titik tengah robot, dan terdapat sudut 120^0 antara roda. Untuk lebih jelasnya dapat dilihat pada gambar 3.6 dibawah ini



Gambar 3.6 Penempatan motor dan roda robot

Dari gambar 3.6 roda satu, dua, dan tiga pada posisi 0^0, 120^0, dan 240^0. Lalu diasumsikan bahwa arah depan robot adalah 0^0. Dan arah panah menunjukan arah positif setiap motor.



Gambar 3.7 Arah yang diinginkan serta perputaran motor

Gambar 3.7 menunjukan bagaimana respon robot jika diberi pergerakan yang diinginkan (anak panah merah) dengan α sebagai arah sudut, dan panjang anak panah merah merupakan kecepatan yang diinginkan. Lalu garis hijau adalah hasil dari masukan yang diberikan (arah dan kecepatan).

Karena arah depan robot diwakili Yr, masing-masing kecepatan motor terdiri dari sinus sudut α (arah yang diinginkan) diproyeksikan pada posisi setiap roda, dan dikalikan dengan kecepatan yang diinginkan. Maka dari itu didapatkan persamaan

Vn.SP=kecepatan x sin(posisi roda° + α) .....(1)

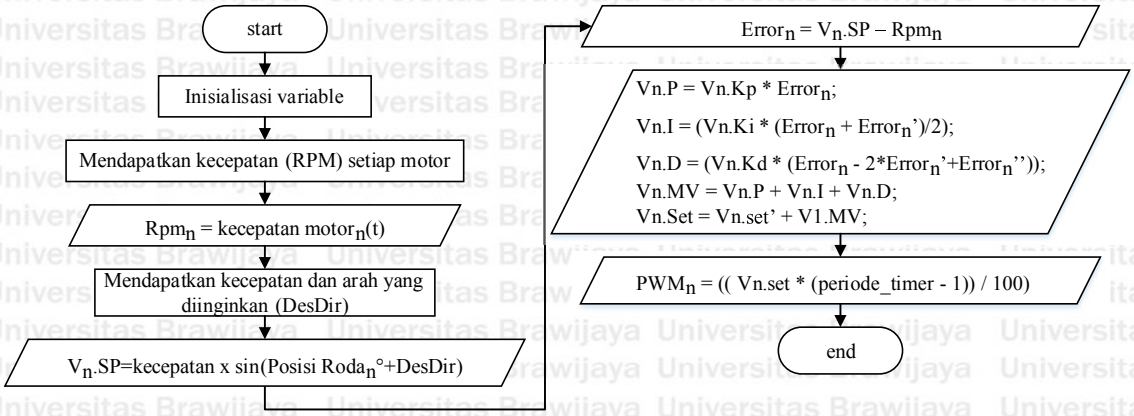
Dengan posisi setiap roda adalah 0^0, 120^0, dan 240^0, persamaan untuk masing-masing roda adalah sebagai berikut

V1.SP=kecepatan x sin(0°+α) .....(2)
V2.SP=kecepatan x sin(120°+ α) .....(3)
V3.SP=kecepatan x sin(240°+ α) .....(4)

ket :
Vn.SP : kecepatan roda ke-n(RPM)
α: arah gerak yang ingin dituju(derajat).

3.2.3.6.Perancangan Pemberian *Duty Cycle* Sinyal PWM untuk *Driver* L298N

Untuk memberi nilai atau mengatur besar *duty cycle* PWM yang akan menjadi masukan *driver* L298N, digunakan persamana *invers kinematic* yang sudah dijelaskan pada sub-bab sebelumnya yang masukannya adalah kecepatan roda yang diinginkan dan arah yang diinginkan dalam bentuk sudut. Lalu dari persamaan ini, didapatkan kecepatan setiap roda yang dibutuhkan agar dapat bergerakan ke arah yang diinginkan serta arah putaran rodanya. Pada Gambar 3.7 dapat dilihat bagaimana proses pemberian nilai *duty cycle* PWM untuk *driver* L298N.



Gambar 3.8 Diagram Alir Menghasilkan Nilai *Duty Cycle* PWM

Keterangan :

n : 1,2, dan 3.

Rpm_n : Kecepatan Motor ke-n aktual(RPM),

Kecepatan : Kecepatan gerak robot yang diinginkan (RPM),

α : arah gerak robot yang diinginkan (derajat),

Vn.SP : kecepatan motor ke-n sesuai dengan posisi penempatan roda ke-n(RPM),

$Error_n$: kesalahan kecepatan motor ke-n (RPM),

Vn.P : hasil kontroler *proportional* untuk motor ke-n.

Vn.I : hasil kontroler *Integral* untuk motor ke-n.

Vn.D : hasil kontroler *Derivative* untuk motor ke-n.

Vn.Set : keluaran kontroler PID untuk motor ke-n (%).

PWMn : nilai untuk *register timer* PWM motor ke-n.

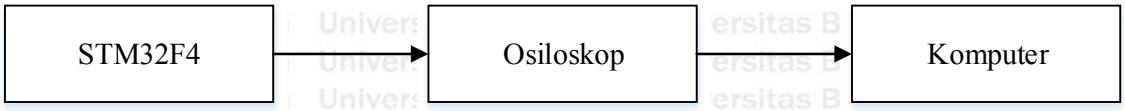
3.3. Pengujian Alat

Untuk mengetahui kinerja alat apakah sudah sesuai dengan yang direncanakan maka dilakukanlah pengujian alat. Pengujian dilakukan pada masing-masing bagian dan kemudian secara kesuluruhan sistem. Secara garis besar pengujian dilakukan adalah sebagai berikut :

3.3.1. Pengujian Keluaran *Duty Cycle* PWM STM32F4

Pengujian ini dilakukan untuk mengetahui apakah keluaran *duty cycle* PWM dari STM32F4 sudah sesuai yang diharapkan atau tidak. Pengujian dilakukan dengan menggunakan osiloskop digital dan PC untuk menampilkan hasilnya.

Prosedur pengujian dilakukan dengan cara menghubungkan STM32F4 dengan osiloskop, lalu osiloskop dihubungkan ke perangkat PC untuk dipantau hasilnya. PWM yang akan diuji sebesar 0%, 25%, 75%, dan 100%. Gambar 17 di bawah ini menunjukan skema pengujian *duty cycle* PWM yang dilakukan

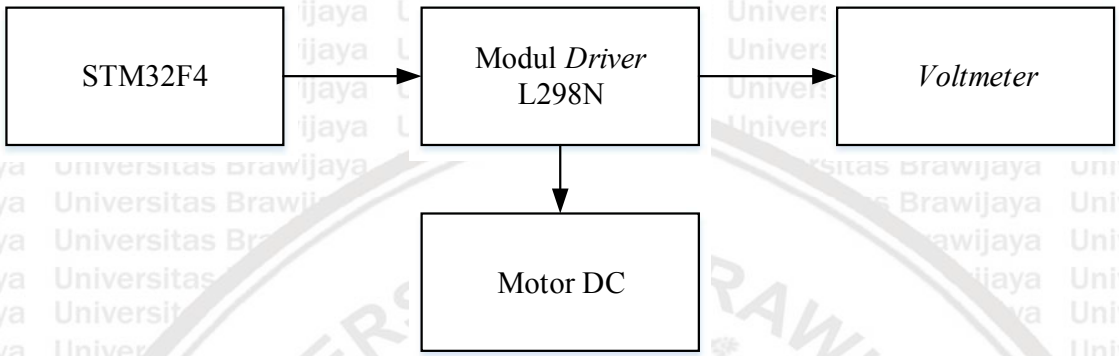


Gambar 3.9 Skema pengujian *duty cycle* PWM STM32F4

3.3.2. Pengujian Modul *Driver* Motor L298N

Pengujian ini dilakukan untuk mengetahui bagaimana keluaran tegangan dari masing masing *driver* motor saat diberikan tegangan masukan yang sama dan *duty cycle* PWM yang sama.

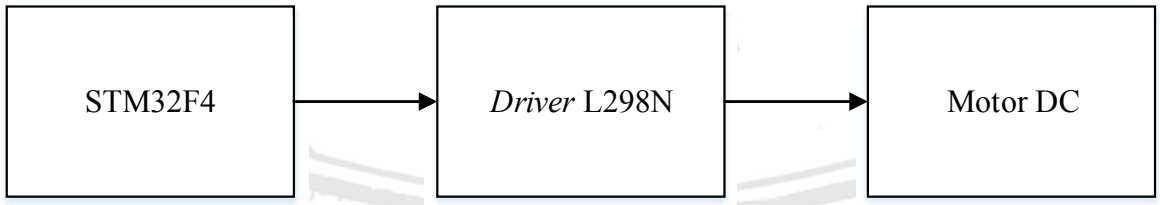
Prosedur pengujian dilakukan dengan menghubungkan STM32F4 dan modul *driver* motor L298N, pada pengujian ini motor DC juga dihubungkan ke keluaran dari L298N. lalu tegangan di cek menggunakan *voltmeter* seperti pada gambar 18 di bawah ini.



Gambar 3.10 Skema pengujian tegangan keluaran L298N

3.3.3. Pengujian Arah Putaran Motor

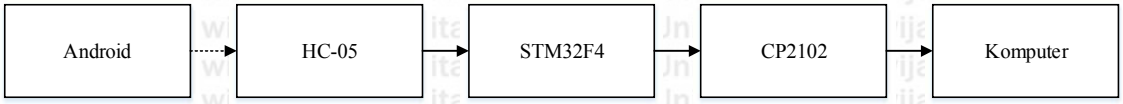
Pengujian ini dilakukan untuk mengetahui apakah arah putaran motor sesuai dengan masukan logika *driver* L298N. Alat yang digunakan dalam pengujian ini adalah STM32F4, *driver* L298N, motor DC, dan catu daya 5V dan 12V. Prosedur pengujian dilakukan dengan menghubungkan keluaran logika STM32F4 dengan masukan logika *driver* L298N, lalu keluaran tegangannya dihubungkan dengan motor DC sesaui pada gambar 19 d bawah ini



Gambar 3.11 Skema pengujian arah putaran motor

3.3.4. Pengujian Transmisi Data *Bluetooth* HC-05

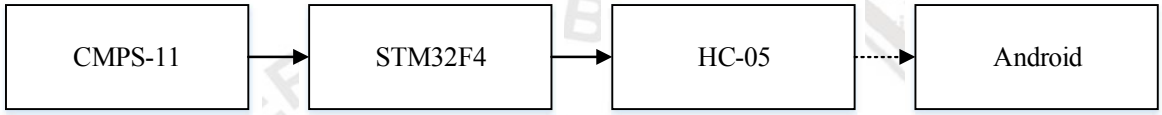
Pengujian ini dilakukan untuk mengetahui apakah data yang dikirim oleh android sudah diterima dengan baik di STM32F4. Prosdur pengujian ini dilakukan dengan mengatur HC-05 agar dapat terkoneksi dengan alamat *device Bluetooth* yang ada di sekitarnya dan mengatur *baudrate*nya menjadi 9600. Setelah itu dilakukan *pairing* antara *Bluetooth* android dengan HC-05, dan selanjutnya data yang diterima STM32F4 dapat dicek pada PC menggunakan modul CP2102. Untuk lebih jelasnya dapat dilihat pada gambar 20 di bawah ini



Gambar 3.12 Skema pengujian transmisi data

3.3.5. Pengujian Nilai *Heading* dari CMPS-11

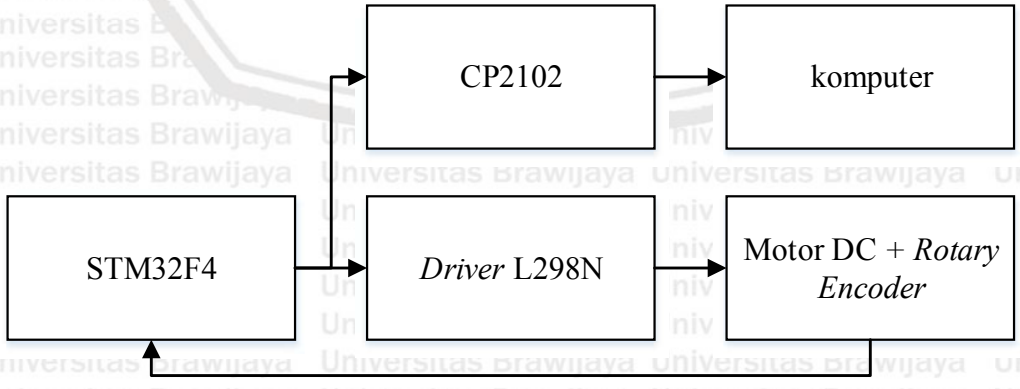
Pengujian ini dilakukan untuk mengetahui apakah data *heading* yang diterima dari sensor kompas CMPS-11 sudah sesuai dengan perputaran robot. Alat-alat yang diperlukan dalam pengujian ini antara lain sensor kompas CMPS-11, STM32F4, HC-05, android, serta catu 5V. Pengujian dilakukan dengan cara memutar robot setiap 45^0 lalu dibandingkan dengan nilai *heading* kompas CMPS-11, nilai *heading* akan dimonitoring melalui aplikasi android dengan *Bluetooth* HC-05 sebagai media pengirimannya dan hasilnya dicatat untuk dibuat persamaan jika nilai *heading* CMPS-11 tidak sesuai dengan perputaran robot yang sebenanya. Gambar 21 dibawah ini menunjukan skema pengujian CMPS-11



Gambar 3.13 Skema pengujian CMPS-11

3.3.6. Pengujian Kecepatan Motor

Pengujian ini dilakukan untuk memastikan kecepatan motor yang dihasilkan sesuai dengan kecepatan yang diinginkan. Alat-alat yang digunakan dalam pengujian ini adalah STM32F4, *driver* motor L298N, motor DC, CP2102, catu daya 5V dan 12V. Pengujian ini dilakukan dengan cara yang sama seperti pengujian arah putaran motor, hanya saja pada pengujian ini nilai yang ditetapkan adalah kecepatan motor dalam satuan RPM dan juga terdapat *feedback rotary encoder* untuk mengetahui kecepatan motor yang sesungguhnya, serta pada pengujian kali ini digunakan kontroller PID untuk mengontrol motor. Gambar 22 berikut menunjukan skema pengujian kecepatan motor

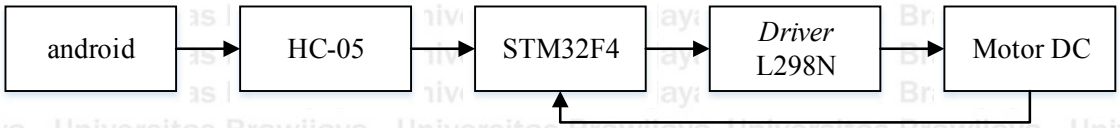


Gambar 3.14 Skema Pengujian Kecepatan Motor

3.3.7. Pengujian Arah Gerak Robot

Pengujian ini dilakukan untuk mengetahui apakah persamaan yang digunakan sudah benar atau masih memiliki kesalahan. Pengujian ini menggunakan robot yang

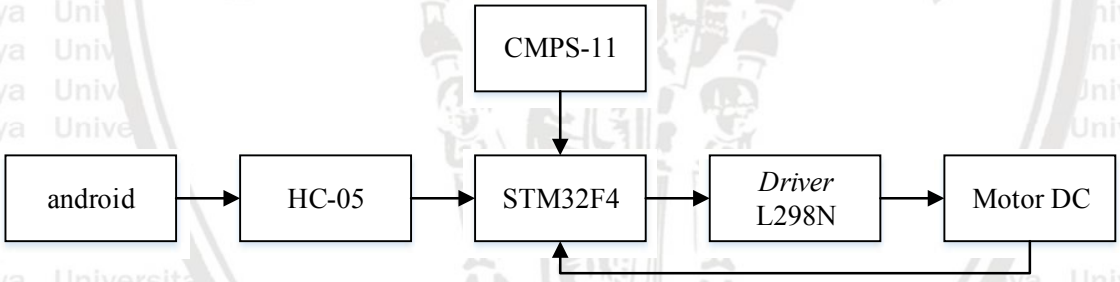
sudah jadi sepenuhnya. Lalu pengujian dilakukan dengan cara memberikan arah sudut yang ingin dituju, lalu respon robot diamati dan dicatat. Alat-alat yang dibutuhkan dalam pengujian ini antara lain, android, hc-05, STM32F4, *driver* motor L298N, motor DC, catu tegangan 5V dan 12V dan gambar 23 di bawah ini menunjukan skema pengujian



Gambar 3.15 Skema pengujian arah gerak robot

3.3.8. Pengujian Keseluruhan

Pengujian ini dilakukan untuk mengetahui apakah robot sudah bekerja dengan hasil yang diharapkan atau tidak. Pengujian ini dilakukan sama seperti pengujian arah gerak robot, akan tetapi pada pengujian kali ini digunakan CMPS-11 yang keluarannya merupakan *feedback heading* robot yang sesungguhnya. Pada pengujian kali ini akan dibandingkan pergerakan robot saat tidak menggunakan CMPS-11 dan saat menggunakan CMPS-11, yang dimana saat tidak menggunakan CMPS-11, robot tidak menggunakan fungsi untuk memperbaiki *heading*nya, dan saat menggunakan CMPS-11 robot memiliki fungsi yang diharapkan dapat memperbaiki *heading*nya secara otomatis jika *heading* robot sewaktu-waktu berubah. Gambar 24 di bawah ini menunjukan skema pengujian keseluruhan



Gamabr 3.16 Skema pengujian keseluruhan robot

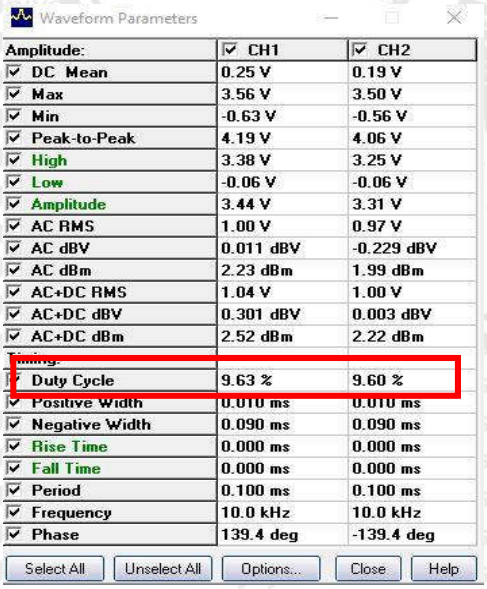
BAB IV
HASIL DAN PEMBAHASAN

Pengujian dan analisis dilakukan untuk menganalisis alat yang telah dirancang dan diimplementasikan apakah telah bekerja sesuai dengan perancangan. Pengujian dilakukan tiap-tiap blok dengan tujuan untuk mengamati apakah tiap rangkaian sudah sesuai dengan perancangan, kemudian dilanjutkan dengan pengujian secara keseluruhan sistem. Adapun pengujian yang telah dilakukan sebagai berikut:

1. Pengujian *duty cyle* PWM yang dihasilkan STM32F4
2. Pengujian *driver* L298N
3. Pengujian arah putaran motor
4. Pengujian transmisi data
5. Pengujian nilai *heading* CMPS-11
6. Pengujian kecepatan motor
7. Pengujian arah gerak robot
8. Pengujian keseluruhan

4.1 Pengujian *Duty Cycle* PWM yang Dihasilkan STM32F4

Pada pengujian, mikrokontroler menhasilkan *duty cycle* sinyal PWM pada *range* 0% sampai 100% dengan kenaikan bertahap sebesar 25%. Hasil pengujian sinyal PWM mikrokontroler diamati menggunakan osiloskop digital Vellemen PCLAB PCSU100. Adapun antarmuka PCSU100 di komputer dapat dilihat pada gambar 4.1 di bawah ini



Gambar 4.1 tampilan *waveform parameter* pada aplikasi PCSU1000

Hasil pengujian keseluruhan dapat dilihat pada tabel 4.1. berdasarkan tabel tersebut, dapat dilihat bahwa keluaran *duty cycle* sinyal PWM yang dihasilkan STM32F4 memiliki kesalahan rata-rata sebesar 0.26%, akan tetapi hasil pengujian ini sudah cukup baik untuk digunakan pada pengujian dan perancangan selanjutnya karena tidak memberikan pengaruh yang besar pada kinerja sistem yang akan dirancang. Maka dapat disimpulkan bahwa keluaran *duty cylcle* PWM yang dihasilkan STM32F4 sudah baik dan dapat digunakan.

Tabel 4.1 Hasil Pengujian *Duty Cycle* PWM dari STM32F4

Duty Cycle PWM yang diharapkan	*Duty Cycle* PWM pada Osiloskop	Selisih
0%	0%	0%
25%	24.6%	0.4%
50%	49.7%	0.3%
75%	74.6%	0.4%
100%	99.8%	0.2%

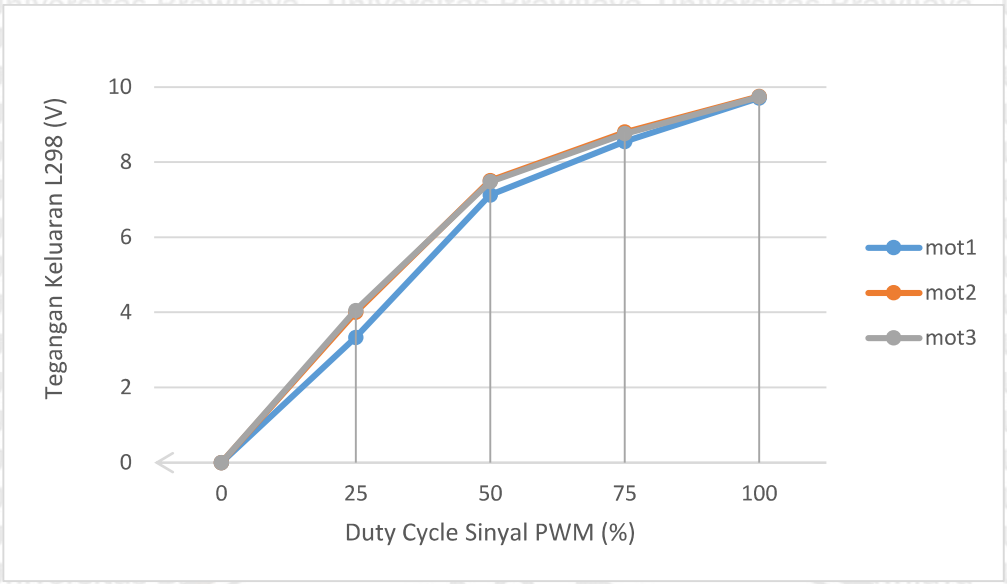
4.2 Pengujian Keluaran Tegangan *Driver* L298N

Pengujian keluaran tegangan *driver* L298N untuk masing masing diuji dengan dengan memberikan tegangan masukan dan logika putaran yang sama, begitu juga dengan masukan *duty cycle* PWMnya. Sinyal *duty cycle* PWM nilainya akan meningkat secara berkala sebesar 25% dimulai dari 0% dan diakhiri pada 100%, tabel 4.2 di bawah ini menunjukan hasil pengujian keluaran tegangan dari *driver* L298N untuk masing-masing motor

Tabel 4.2 hasil pengujian keluaran tegangan *driver* L298N

Duty Cycle PWM	Keluaran Tegangan (volt)		
(%)	Motor1	Motor2	Motor3
0	0	0	0
25	3.33	4	4.05
50	7.12	7.51	7.48
75	8.55	8.8	8.76
100	9.71	9.75	9.74

Pengujian diatas dilakukan saat tegangan masukan *driver* L298N adalah sebesar 11.41V. dari tabel di atas dapat dilihat bahwa, meskipun tegangan masukan dan *duty cycle* PWM sama, akan tetapi keluaran tegangan untuk setiap motor berbeda-beda. Gambar 4.2 di bawah ini menunjukan bagaimana linieritas keluaran tegangan driver L298



Gambar 4.2 keluaran tegangan driver L298N untuk setiap motor

Dari gambar 4.2 d atas, dapat dilihat bahwa kenaikan tegangan tidak linier, dan keluaran tegangan untuk motor satu selalu lebih kecil daripada keluaran untuk motor dua dan motor 3. Akan tetapi perbedaan keluaran pada gambar 4.2 di atas dapat dikatakan kecil, maka dari itu hal ini tidak akan menjadi masalah untuk perancangan kedepannya.

4.3 Pengujian Arah Putaran Motor

Pengujian arah gerak motor diatur dengan mengubah keadaan logika dari pin *direction* pada mikrokontroler. Hasil pengujian arah putaran motor dapat dilihat pada Tabel 4.3

Tabel 4.3 Hasil Pengujian Arah Putaran Motor

Motor 1		Motor 2		Motor 3		Respon Motor	
Logika 1	Logika 2	Logika 1	Logika 2	Logika 1	Logika 2	Yang diharapkan	Hasil Pengujian
1	0	1	0	1	0	CCW	CCW
0	1	0	1	0	1	CW	CW
0	0	0	0	0	0	Diam	Diam
1	1	1	1	1	1	Diam	Diam

Keterangan :

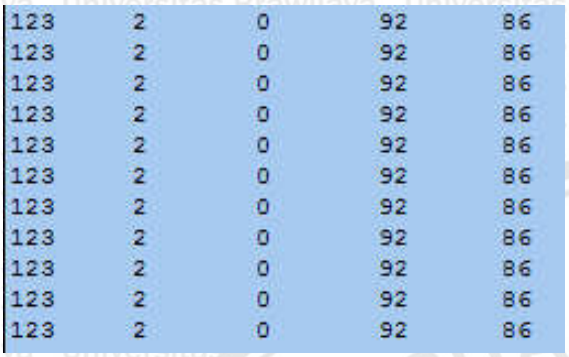
CCW : *Counter Clock Wise*
CW : *Clock Wise*

Dari Tabel 4.3 dapat diketahui bahwa respon *driver* motor L298N terhadap sinyal masukan arah dari mikrokontroler bekerja sesuai dengan perancangan yang diharapkan.

Sehingga dapat disimpulkan bahwa driver L298N dapat bekerja dengan baik saat mendapatkan sinyal logika arah dari mikrokontroler.

4.4 Pengujian Transmisi Data

Pengujian ini dilakukan dengan cara mengirimkan data dari android ke STM32F4 dengan media *Bluetooth*. Data yang dikirim merupakan data untuk pergerakan robot, adapun hasil pengiriman data dapat dilihat pada Gambar 4.3 di bawah ini



Gambar 4.3 Hasil Penerimaan Data pada STM32F4

Dari gambar di atas dapat dilihat bahwa hasil penerimaan data dengan metode yang sudah dijelaskan pada bab sebelumnya memiliki hasil yang baik karena tidak ada pergeseran data, ataupun kesalahan.

Selain itu, jarak transmisi antara android dan robot juga diuji, dan hasilnya dapat dilihat pada Tabel 4.4 di bawah ini

Tabel 4.4 Hasil Pengujian Jarak Transmisi Data

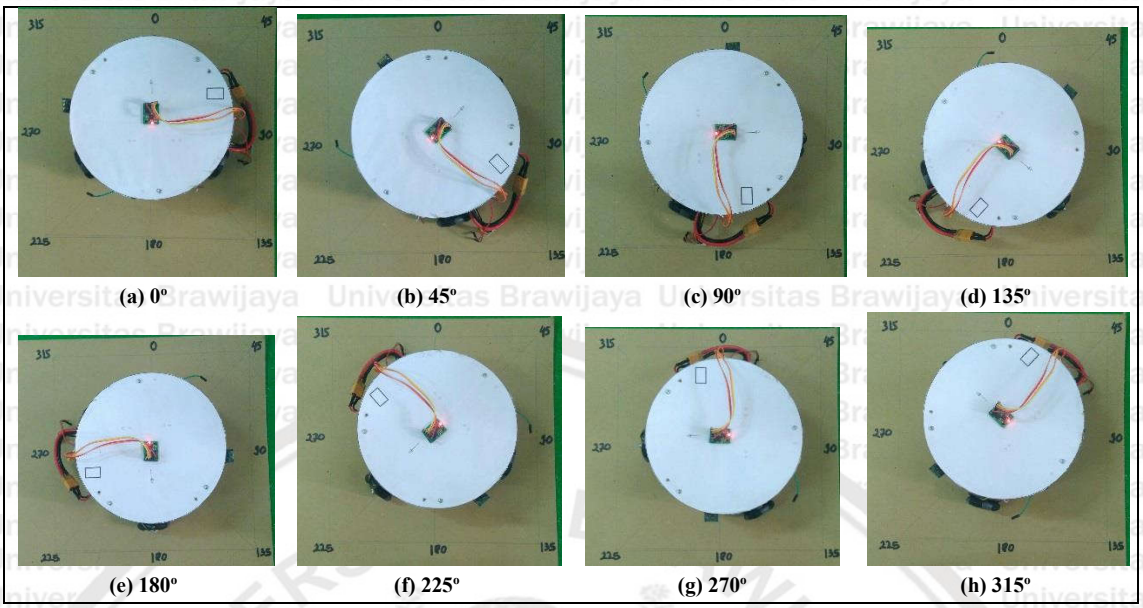
Jarak (Meter)	Kondisi Komunikasi	Data yang Diterima
1-7	Terhubung	Benar
8	Terhubung	Benar
9	Terhubung	Benar
10	Terhubung	Benar
11	Terhubung	Benar
12	Terhubung	Benar
13	Terhubung	Benar
14	Terhubung	Benar
15	Tidak Terhubung	Tidak Menerima Data
16	Tidak Terhubung	Tidak Menerima Data
17	Tidak Terhubung	Tidak Menerima Data
18	Tidak Terhubung	Tidak Menerima Data

Dari Tabel 4.4 di atas dapat dilihat, jarak transmisi terjauh adalah 14 meter, dan sampai jarak tersebut data yang diterima masih benar. Saat jarak transmisi sudah mendekati 15 meter, *Bluetooth* sudah tidak terhubung dan transmisi data tidak berjalan.

4.5 Pengujian Nilai *Heading* CMPS-11

Pengujian ini dilakukan dengan cara membaca *heading* sensor kompar CMPS-11, lalu datanya dimonitoring pada computer dan dicatat. Pada pengujian ini, pertama-tama robot

akan dihadapkan sampai nilai dari CMPS-11 samadengan 0, setelah itu robot akan diputar setiap 45^0, setelah itu diamati bagaimana respon dari CMPS-11. Untuk lebih jelasnya dapat dilihat pada Gambar 4.4 di bawah ini



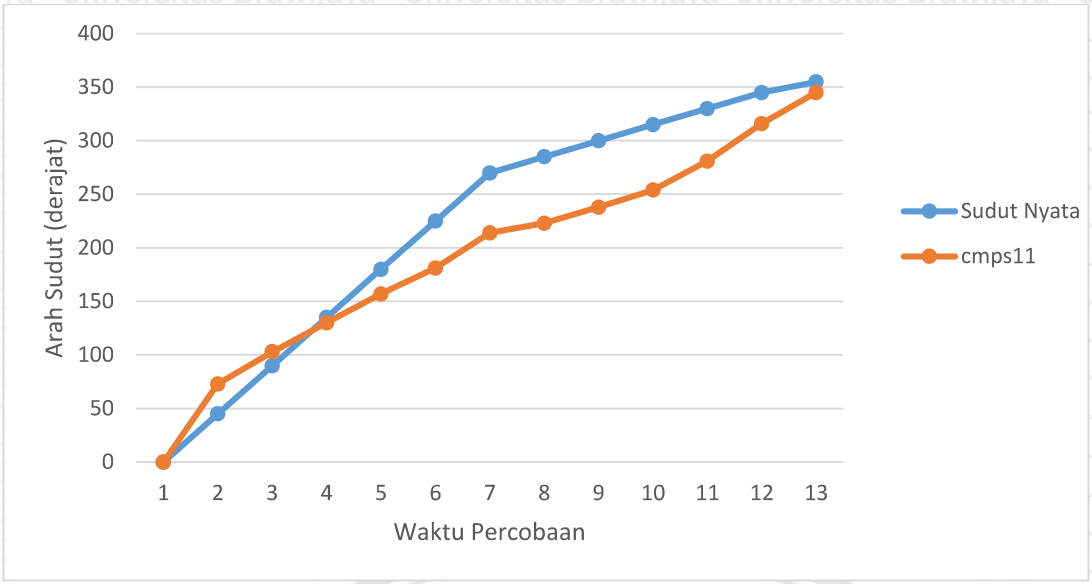
Gambar 4.4 Pengujian Data CMPS-11

Adapun hasil pengujian ini dapat dilihat pada Tabel 4.5 di bawah ini

Tabel 4.5 Hasil Pengujian Data CMPS-11

Perputaran Robot(derajat)	CMPS-11
0	0
45	73
90	103
135	130
180	157
225	181
270	214
285	223
300	238
315	254
330	281
345	316
355	345

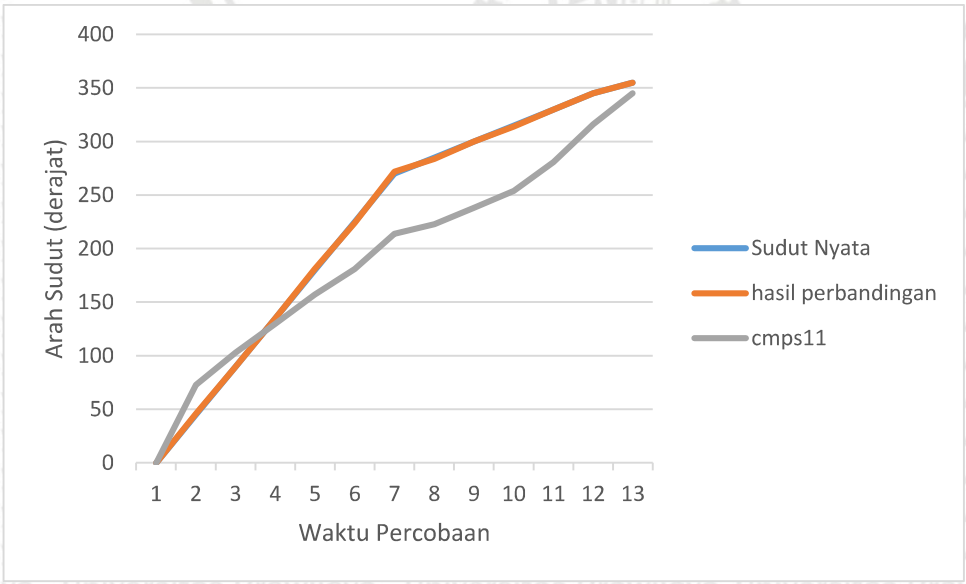
Dari Tabel 4.5 di atas dapat dilihat bahwa data yang dihasilkan oleh CMPS-11 tidak sesuai dengan perputaran robot. Untuk melihat lebih jelas bagaimana respon CMPS-11 dapat dilihat pada Gambar 4.4 di bawah ini



Gambar 4.4 Grafik Hasil Pengujian CMPS-11

Dari Gambar 4.4 di atas dapat dilihat bahwa nilai CMPS-11 tidak linier, saat arah hadapnya antara 0 sampai 135 derajat, nilai hasil CMPS-11 selalu lebih besar dari nilai perputaran yang sesungguhnya, akan tetapi setelah melewati batas 135 derajat, nilai CMPS-11 menjadi jauh lebih kecil dibanding nilai perputaran yang sesungguhnya sampai kembali ke arah hadap 0 derajat.

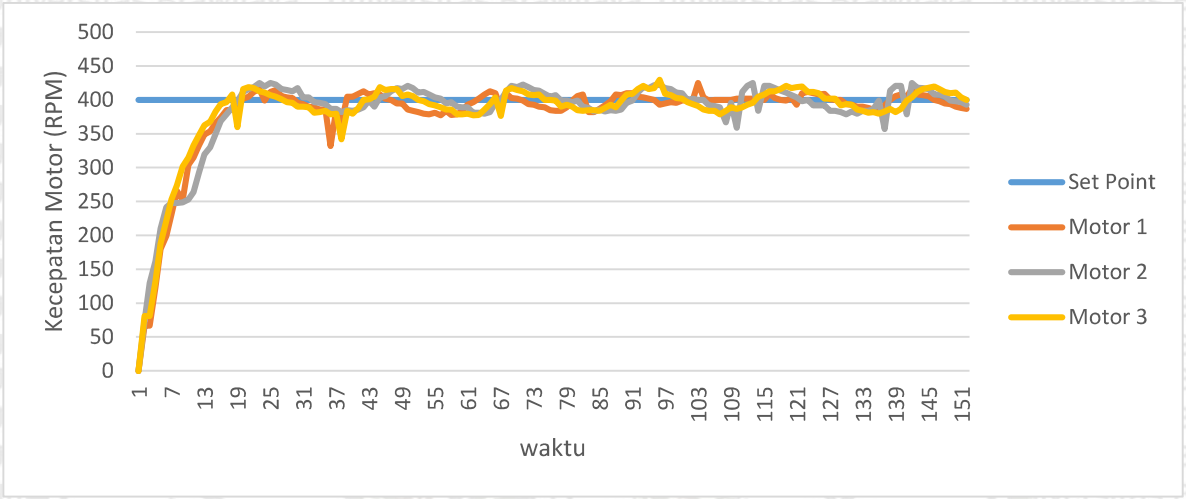
Maka dari itu, penulis membuat persamaan untuk membuat nilai hasil CMPS-11 sesuai dengan arah putarannya menggunakan persamaan perbandingan nilai terkecil dan terbesar setiap perputaran 45 derajat sampai pada sudut 270 derajat, lalu setelah itu perbadingan dilakukan setiap perputaran 15 derajat karena perubahan nilainya semakin besar saat perubahan arah sudutnya kecil, untuk hasilnya dapat dilihat pada ganbar 4.5 di bawah ini



Gambar 4.5 Hasil CMPS-11 Setelah Menggunakan Perbandingan

4.6 Pengujian Kecepatan Motor

Pengujian ini dilakukan dengan cara memberi *set point* berupa kecepatan dalam satuan RPM ke STM32F4. Setelah itu STM32F4 akan menghasilkan *duty cycle* PWM ke *driver* L298N untuk memutar motor, lalu kecepatan motor yang sesungguhnya didapat dengan menggunakan sensor *rotary encoder* yang sudah terhubung dengan motornya dan data *rotary encoder* akan diolah menjadi RPM pada STM32F4 Adapun hasilnya dapat dilihat pada Gambar 4.6 di bawah ini

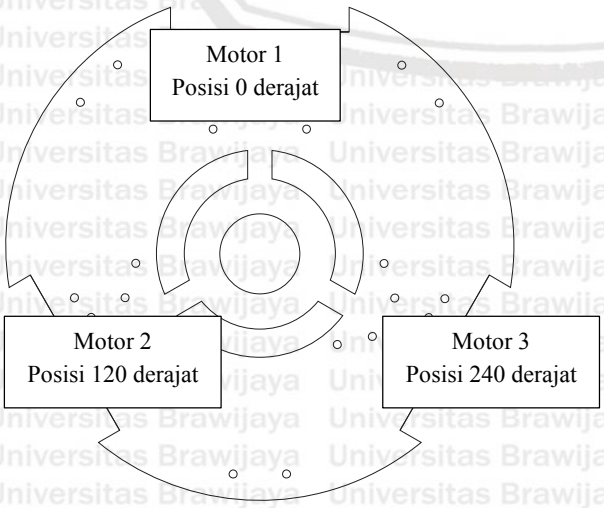


Gambar 4.6 Hasil Pengujian Kecepatan Motor

Dari Gambar 4.6 dapat dilihat respon kecepatan untuk setiap motor sudah cukup baik karena kecepatan motor berosilasi dekat dengan garis *set point*(garis warna biru). Untuk motor 1 kesalahan rata-ratanya adalah ±8.526, lalu untuk motor 2 kesalahan rata-ratanya adalah ±12.706, dan rata-rata kesalahan motor 3 adalah ±12.308.

4.7 Pengujian Arah Gerak Robot

Pengujian ini dilakukan dengan cara yang sama seperti pengujian kecepatan motor, akan tetapi pada pengujian ini, masukan yang diberikan ke STM32F4 adalah kecepatan yang diinginkan serta arah yang diinginkan. Adapun penempatan roda dan motor dapat dilihat pada Gambar 4.7



Gambar 4.7 Penempatan Motor dan Roda

Adapun hasil pengujian ini dapat dilihat pada tabel 4.6

Tabel 4.6 Hasil Pengujian Arah Gerak Robot

Arah	RPM	SP Mot1	SP Mot2	SP Mot3	Arah Gerak Robot	
					Yang diharapkan	Hasil Pengujian
0	400	0	346.4102	-346.41	Maju	Maju
45	400	282.8427	103.5276	-386.37	Serong atas kiri	Serong atas kiri
90	400	400	-200	-200	Kiri	Kiri
135	400	282.8427	-386.37	103.5276	Serong bawah kiri	Serong bawah kiri
180	400	0	-346.41	346.4102	Mundur	Mundur
225	400	-282.843	-103.528	386.3703	Serong bawah kanan	Serong bawah kanan
270	400	-400	200	200	Kanan	Kanan
315	400	-282.843	386.3703	-103.528	Serong atas kanan	Serong atas kanan

Dari tabel di atas dapat dilihat kecepatan motor ada yang tidak sesuai dengan nilai RPM yang diberikan karena penempatan posisi roda dimana jika nilainya positif maka arah putaran motornya adalah searah jarum jam, dan saat nilainya negative arah putarannya adalah tidak searah jarum jam, dan arah yang diharapkan dengan hasil pengujian adalah sama.

4.8 Pengujian Sistem Keseluruhan

Pengujian dilakukan sama seperti pengujian arah gerak robot, akan tetapi pada pengujian ini *heading* robot dari CMPS-11 digunakan. Pada pengujian ini robot akan berjalan dengan kecepatan yang sudah ditentukan. Saat robot sampai pada tujuan, posisi robot akan dicatat. pada pengujian ini, kecepatan yang diberikan adalah 200 RPM dengan arah sudut 0^0.Adapun hasilnya dapat dilihat pada Tabel 4.7

Tabel 4.7 Pengujian Keseluruhan Robot

Pengujian ke	*Heading* Robot
1	Bergeser 4^0
2	Bergeser 5^0
3	Bergeser 5^0
4	Bergeser 4^0

Pengujian dilakukan sebanyak 4 kali. Berdasarkan tabel di atas, robot masih memiliki pergeseran sebesar 5^0 dan kesalah yang terkecil adalah 4^0.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil analisis dan pengujian setiap bagian dan keseluruhan sistem yang telah dilakukan, maka dapat dilambil kesimpulan sebagai berikut :

1. Untuk membuat robot beroda tiga *omni*, penempatan masing-masing motor berjarak 120^0. Pada perancangan ini penempatan roda 1 berada pada posisi 0^0, roda 2 pada 120^0, dan roda 3 pada posisi 240^0, dengan penempatan seperti ini robot dapat berjalan sesuai dengan yang diharapkan
2. Pengiriman data dari android ke STM32F4 berjalan dengan lancar sampai pada jarak 14 meter, setelah memasuki jarak 15 meter, koneksi *Bluetooth* terputus dan data tidak terkirim.
3. Data *heading* dari CMPS-11 tidak linier, maka dari itu dirancang persamaan dengan menggunakan rumus perbandingan nilai terkecil dan terbesar CMPS-11 setiap 45^0 karena saat arah hadapnya 0^0 sampai 135^0 niai CMPS-11 lebih besar dari arah hadap sesungguhnya, dan saat melebihi 135^0 nilai CMPS-11 selalu lebih kecil dari arah hadap sesungguhnya.
4. Kecepatan motor yang dihasilkan menggunakan kontroler PID dengan metode *hand-tunning* sudah cukup baik.
5. Dengan menggunakan persamaan *omni-drive* umum, hasil pergerakan robot sudah sesuai dengan arah yang diharapkan.

5.2 Saran

1 Agar pegerakan robot lebih baik dan stabil, disarankan untuk perancangan kedepannya digunakan empat roda dan empat motor.

2 Disarankan mencari sensor kompas yang hasilnya sudah linier.

3 Untuk penggunaan *driver* motor dianjurkan untuk menggunakan *driver* motor yang dapat menahan beban arus lebih dari 2A.

4 Menggunakan Algoritma program yang lebih baik untuk memperbaiki arah hadap robot.

DAFTAR PUSTAKA

Braunl, T. (2008). Embedded Robotics. In *Embedded Robotics*. Crawley, Perth: Springer .

Devantech. (n.d.). CMPS-11.

F. Ribeiro, I. M. (2002). THREE OMNI-DIRECTIONAL WHEELS CONTROL ON A MOBILE ROBOT. 1.

Harashima, F. (1999). *Kinematic Correction of Differential Drive Mobile Robot and Design for Velocity Trajectory with Acceleration Constraints on Motor Controller.* Proceeding of The 1999 IEEE/RSJ.

Ogata, K. (2010). *Modern Control Engineering (Fifth Edition).* New Jersey.

Ozyagcilar, T. (2015). *Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors.* Texas: Freescale Semiconductor.

Purnama, A. (2012, July 4). *Teori Motor DC Dan Jenis-Jenis Motor DC*. Retrieved from ELEKTRONIKA DASAR: http://elektronika-dasar.web.id/teori-motor-dc-dan-jenis-jenis-motor-dc/

Riberio, M. I. (2002). *Kinematics Models of Mobile Robots.* Portugal: Instituto Superior Tecnico.

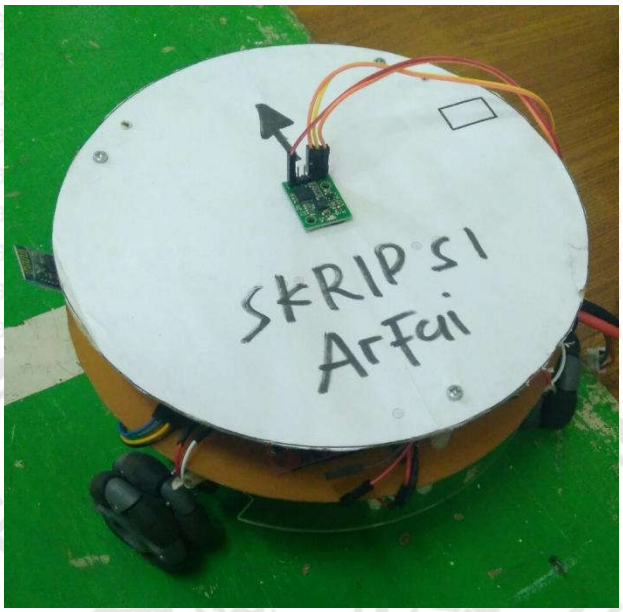
Rojas, R. (2005). Omnidirectional Control. 2.

Sparkfun. (2017, may 24). *I2C*. Retrieved from Sparkfun: https://learn.sparkfun.com/tutorials/i2c

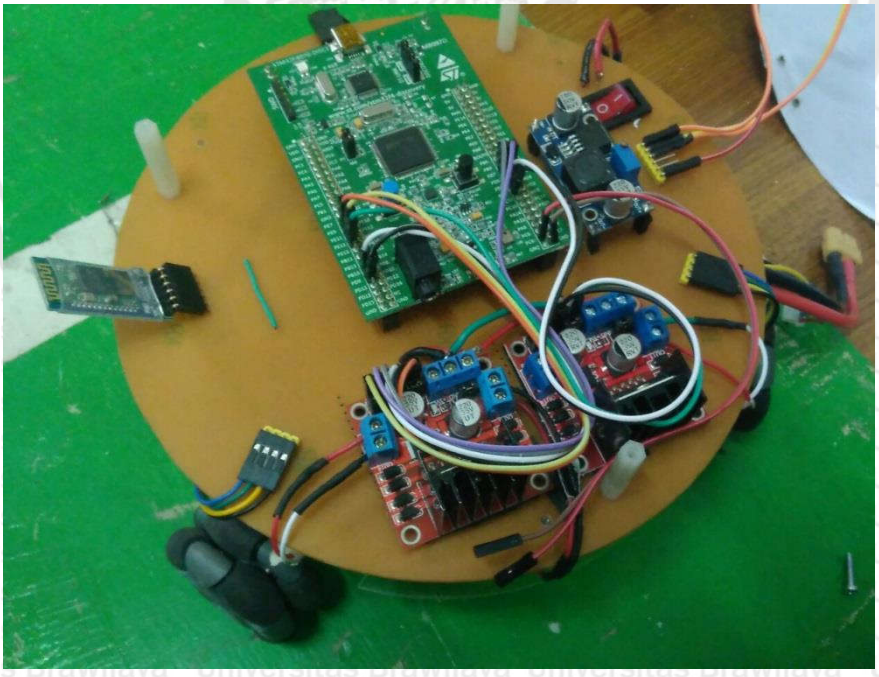
STMicroelectronics. (2013). *LSM9DS0.* STMicroelectronics.

STMicroelectronics. (2016). *Discovery kit with STM32F407VG MCU.* Switzerland: STMicroelectronics.

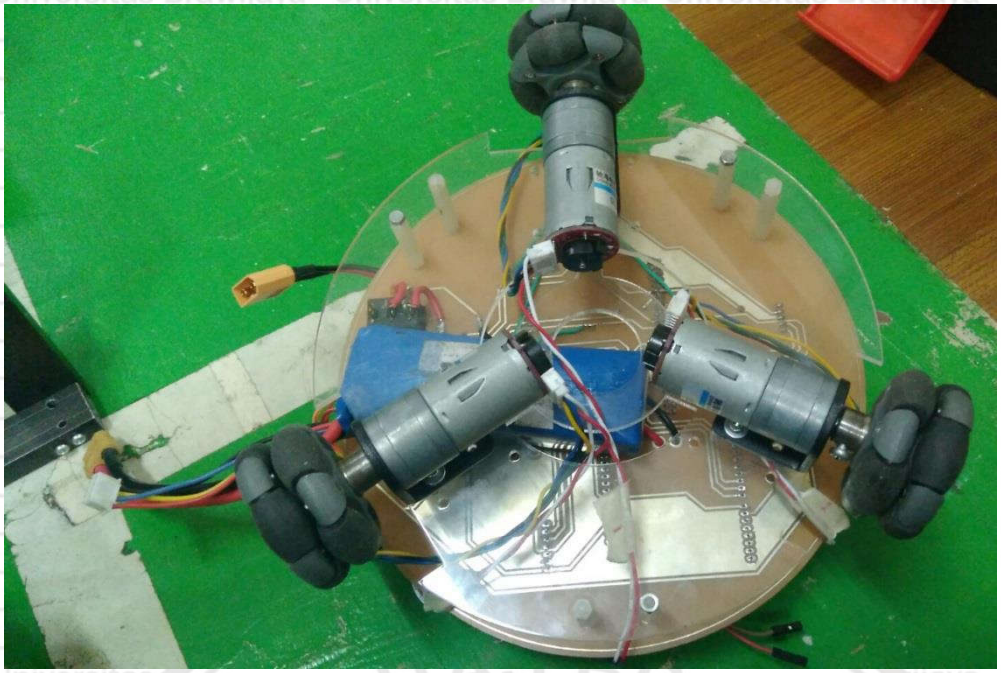
LAMPIRAN 1
DOKUMENTASI ALAT



Gambar 1.1 Robot Tampak Atas

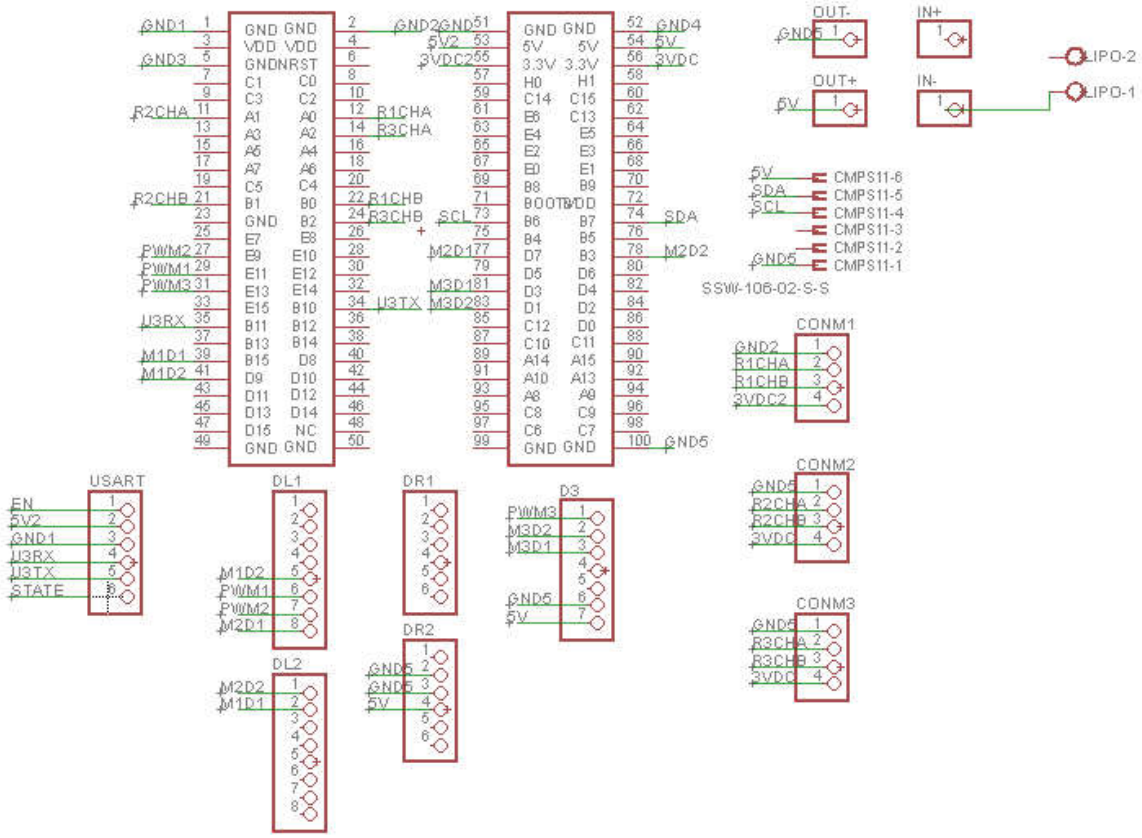


Gambar 1.2 Rangkaian Elektrik Robot

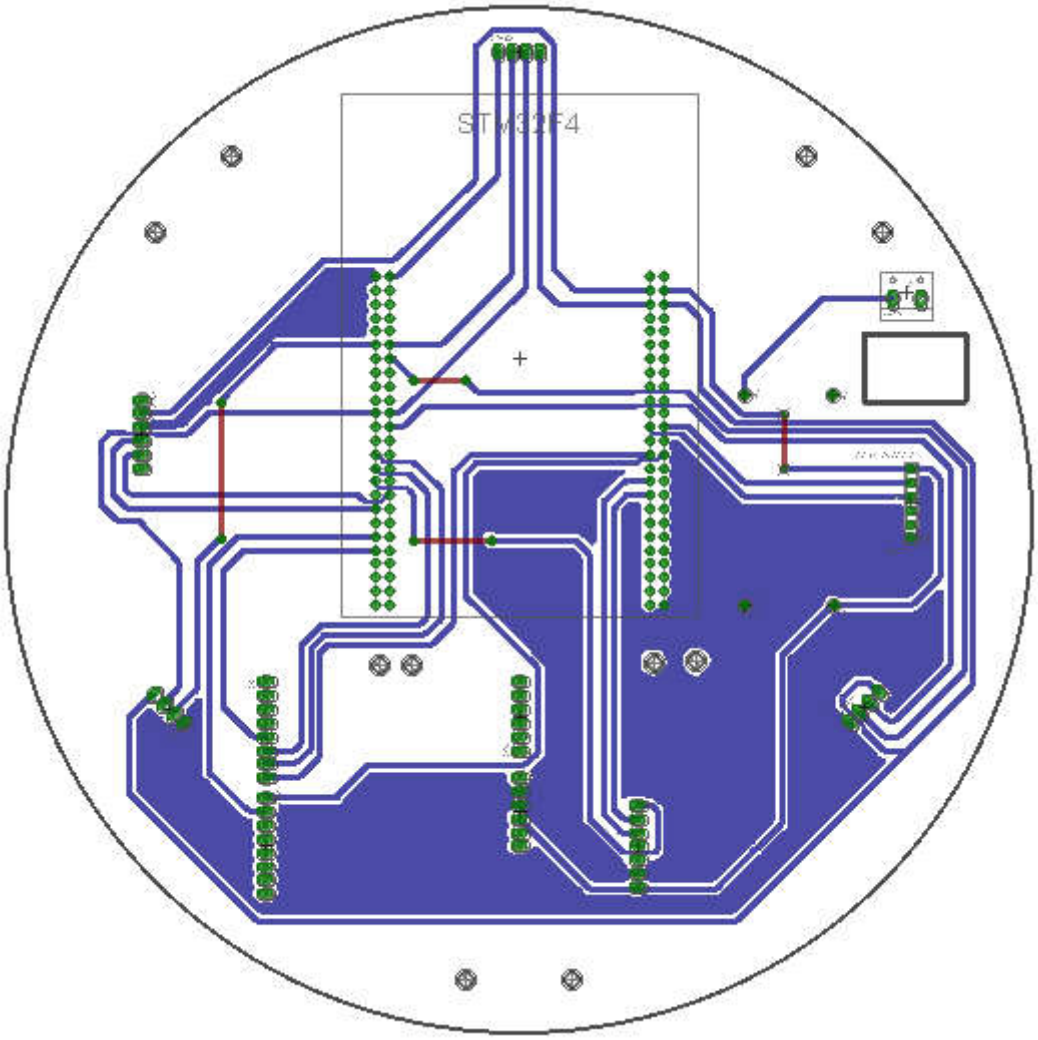


Gambar 1.3 Tampak Bawah Robot

LAMPIRAN 2
SKEMATIK RANGKAIAN



Gambar 2.1 Skematik *Board* Utama



Gambar 2.2 *Layout Board* Utama

LAMPIRAN 3
LISTING PROGRAM

```
#include "i2c.h"
#include "TEUB_GPIO.h"
#include "TEUB_USART.h"
#include "misc.h"
#include "stm32f4xx_tim.h"
#include "stm32f4xx_exti.h"
#include "stm32f4xx.h"
#include "stm32f4xx_syscfg.h"
#include "stm32f4xx_rcc.h"

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>

#define ON    0x01
#define OFF 0x02
#define TO    0x04

#define motor1                4
#define motor2                5
#define motor3                6

#define maju 0x01
#define mundur 0x02
#define diam 0x04

#define pi 3.14159265358979323846264338327795

#define Out GPIO_ReadOutputDataBit
#define In GPIO_ReadInputDataBit

#define CMPS11_ADDRESS                    0xC0 //Default CMPS-11 DEVICE
ADDRESS
#define CMPS11_CMD_REGISTER          0x00 // Command register (write) /
Software version (read)

#define CMPS11_BEARING_8BIT          0x01 // 0-255
#define CMPS11_BEARING_16BIT_H       0x02 // HIGH BYTE 0-3599
representing 0-355.9 degrees
#define CMPS11_BEARING_16BIT_L       0x03 // LOW BYTE
#define CMPS11_PITCH_KF                   0x04 // PITCH angle Kalman
Filtered
#define CMPS11_ROLL_KF               0x05 // ROLL angle Kalman Filtered
#define CMPS11_PITCH_NO_KF      0x20
#define CMPS11_ROLL_NO_KF       0x21

#define CMPS11_MAG_XH                0x06
#define CMPS11_MAG_XL                0x07
```

```
#define CMPS11_MAG_YH                   0x08
#define CMPS11_MAG_YL                   0x09
#define CMPS11_MAG_ZH                   0x0A
#define CMPS11_MAG_ZL                   0x0B

#define CMPS11_ACC_XH                   0x0C
#define CMPS11_ACC_XL                   0x0D
#define CMPS11_ACC_YH                   0x0E
#define CMPS11_ACC_YL                   0x0F
#define CMPS11_ACC_ZH                   0x10
#define CMPS11_ACC_ZL                   0x11

#define CMPS11_GYRO_XH                  0x12
#define CMPS11_GYRO_XL                  0x13
#define CMPS11_GYRO_YH                  0x14
#define CMPS11_GYRO_YL                  0x15
#define CMPS11_GYRO_ZH                  0x16
#define CMPS11_GYRO_ZL                  0x17

#define CMPS11_TEMP_H                   0x18
#define CMPS11_TEMP_L                   0x19

#define CMPS11_CAL_1                    0xF0
#define CMPS11_CAL_2                    0xF5
#define CMPS11_CAL_3                    0xF6
#define CMPS11_CAL_EXIT                 0xF8

uint16_t PWM_1 = 0;
uint16_t PWM_2 = 0;
uint16_t PWM_3 = 0;

int16_t yaw,yaw_h,yaw_l, pitch,roll,tempt,yaw_mag, imu_take, fst_yaw;
int16_t mag_x,mag_y,mag_z;
int tanda_BT2, angka1, hasil[4];
int vxy, SetAksel_v, SetAksel_v2, GetAksel_v, GetSudut_v, sudut_smt,kec_w,
aksel_v, count1, count2;
float sudut_v;
int re_center_flg, yaw_flag, flag1, start_flag,new_yaw,new_yaw1;

GPIO_InitTypeDef                GPIO_InitStruct;
GPIO_InitTypeDef                GPIO_InitStructure;
EXTI_InitTypeDef                EXTI_InitStruct;
NVIC_InitTypeDef                NVIC_InitStruct;
NVIC_InitTypeDef                NVIC_InitStructure;
TIM_ICInitTypeDef               TIM_ICInitStructure;
TIM_TimeBaseInitTypeDef         TIM_TimeBaseStructure;
TIM_OCInitTypeDef               TIM_OCInitStructure;

typedef struct{
    uint32_t EXTI_ReadValue1; //1 --> Baca data pertama
    uint32_t EXTI_ReadValue2; //2 --> Baca data kedua
    uint8_t CaptureNumber;      //3 --> Penanda
    uint32_t Capture;           //4 --> Data hasil kalkulasi
    float Freq;                 //5 --> Frekuensi putaran
    int Rpm;                    //6 --> Kecepatan putaran
    int encoder;              //7 --> Data encoder
    int sudut;                //8 --> Data sudut
```

```
    int ppr;                                    //9 --> Data pulse per rotation
encoder
    }
    Rotary;
//          1 2 3 4 5 6 7 8 9
Rotary rt_1={0,0,0,0,0,0,0,0,225};
Rotary rt_2={0,0,0,0,0,0,0,0,225};
Rotary rt_3={0,0,0,0,0,0,0,0,225};

uint32_t tim_period;

typedef struct{
    float P;
    float I;
    float D;
    float MV;
    float error;
    float last1_error;
    float last2_error;
    int temp; //variabel penyimpan speed sementara
    float set;
    float Kp;
    float Kd; //konstanta d tidak dipakai
    float Ki; //konstanta i tidak dipakai
    float SP; //setpoint kecepatan motor
    int MAX;
    int MIN;
    unsigned char arah;
}PID;

//P, I, D, MV,error, last1_error,last2_error, temp, set, Kp, Kd, Ki, SP , MAX,
MIN, arah
PID V1 ={0,0,0,0,0,0,0,0,0,0.449,0,0.013,0,100,0,0};
PID V2 ={0,0,0,0,0,0,0,0,0,0.449,0,0.013,0,100,0,0};
PID V3 ={0,0,0,0,0,0,0,0,0,0.449,0,0.013,0,100,0,0};
PID S1 ={0,0,0,0,0,0,0,0,0,2.5,1.2,1,0,100,0,0};
typedef struct
{
    int prt;
}p_rt;

p_rt prt_1 = {0};
p_rt prt_2 = {0};
p_rt prt_3 = {0};

int speed1, speed2, speed3;
uint32_t CCR1_Val = 5000;
uint32_t CCR2_Val = 50000;
uint32_t CCR3_Val = 1000;
uint32_t CCR4_Val = 65535;
int PrescalerValue, capture;

void pwm_motor_config(uint16_t frek);
void EXTI_config(void);
void omni_kinematik(int speed, float sudut, int W);
void aksel (void);
```

```
void motor_pwm(uint16_t motor, uint8_t arah, uint16_t pwm);
void PID_Kecepatan1(int RPM, int arah);
void PID_Kecepatan2(int RPM, int arah);
void PID_Kecepatan3(int RPM, int arah);
void PID_Sudut_rotasi(void);
void UU_PutChar(USART_TypeDef* USARTx, uint8_t ch);
void UU_PutString(USART_TypeDef* USARTx, uint8_t * str);
void UU_PutNumber(USART_TypeDef* USARTx, uint32_t x);

int main(void)
{
    SystemInit();
    SysTick_Config(SystemCoreClock/1000);

    //dir motor 1
    init_IO(RCC_AHB1Periph_GPIOB, P_15, GPIO_Mode_OUT, GPIO_OType_PP,
GPIO_PuPd_NOPULL );
    init_IO(RCC_AHB1Periph_GPIOD, P_9, GPIO_Mode_OUT, GPIO_OType_PP,
GPIO_PuPd_NOPULL );

    //dir motor 2
    init_IO(RCC_AHB1Periph_GPIOB, P_3, GPIO_Mode_OUT, GPIO_OType_PP,
GPIO_PuPd_NOPULL );
    init_IO(RCC_AHB1Periph_GPIOD, P_7, GPIO_Mode_OUT, GPIO_OType_PP,
GPIO_PuPd_NOPULL );

    //dir motor 3
    init_IO(RCC_AHB1Periph_GPIOD, P_3|P_1, GPIO_Mode_OUT, GPIO_OType_PP,
GPIO_PuPd_NOPULL );

    init_IO(RCC_AHB1Periph_GPIOD, P_12|P_13|P_14|P_15, GPIO_Mode_OUT,
GPIO_OType_PP, GPIO_PuPd_NOPULL );
    init_IO(RCC_AHB1Periph_GPIOB, P_0|P_1|P_4, GPIO_Mode_IN, GPIO_OType_PP,
GPIO_PuPd_UP );
    init_IO(RCC_AHB1Periph_GPIOA, P_0|P_1|P_2, GPIO_Mode_IN, GPIO_OType_PP,
GPIO_PuPd_UP );

    init_USART(RCC_APB2Periph_USART6, 9600, RCC_AHB1Periph_GPIOC, P_6, TX);
//cek cek
    init_USART(RCC_APB1Periph_USART3, 9600, RCC_AHB1Periph_GPIOB, P_10|P_11,
TX_RX); //bluetooth

    pwm_motor_config(1000);
    EXTI_config();
    TIM3_CC();
    //init i2c B6 SCL | B7 SDA
    I2C_Inits();
  while(1)
  {
    cetak(USART6,"%d\t%d\t%d\t||\t%d\r",rt_1.Rpm,rt_2.Rpm,rt_3.Rpm,yaw);

    UU_PutNumber(USART3,aksel_v);
    UU_PutString(USART3,"\t\t");
    UU_PutNumber(USART3,S1.SP);
    UU_PutString(USART3,"\t\t");
```

```
        UU_PutNumber(USART3,new_yaw1);
        UU_PutString(USART3,"\n");
    }
}

void USART3_IRQHandler(void){
    if( USART_GetITStatus(USART3, USART_IT_RXNE) ){
        char t = USART3->DR; // the character from the USART3 data register
is saved in t
        char static last_data;
        Pin(GPIOD,TO,P_15);
        tanda_BT2++;
        switch(angka1)
        {
            case 0 : vxy=USART3->DR; angka1+=1; break;
            case 1 : GetSudut_v=USART3->DR; angka1+=1; break;
            case 2 : kec_w=USART3->DR; angka1+=1; break;
            case 3 : SetAksel_v=USART3->DR; angka1+=1; break;
            case 4 : SetAksel_v2=USART3->DR; angka1+=1; break;
            case 5 : re_center_flg=USART3->DR; angka1=100; break;
        }
        if(t==170&&last_data==85)angka1=0;
        last_data=t;
        if(kec_w == 40)kec_w= 100;
        else if(kec_w == 216)kec_w= -100;

        GetAksel_v = SetAksel_v+SetAksel_v2;
    }

}

void omni_kinematik(int speed, float sudut, int W)
{
    sudut = (sudut/180)*pi;

    V1.SP = (int) (speed*(-1)*sin(sudut)+W);
    V2.SP = (int) (speed*(-1)*sin(sudut+(pi/3)*2)+W);
    V3.SP = (int) (speed*(-1)*sin(sudut+(pi/3)*4)+W);

        if (V1.SP<0) {      V1.SP = -V1.SP;     V1.arah = mundur;   }
        else if(V1.SP>0){   V1.SP = V1.SP;          V1.arah = maju;
    }
        else                {       V1.SP = 0;              V1.arah=diam;
        }

        if (V2.SP<0) {      V2.SP = -V2.SP;     V2.arah = mundur;   }
        else if(V2.SP>0){   V2.SP = V2.SP;          V2.arah = maju;
    }
        else                {       V2.SP = 0;              V2.arah = diam;
        }

        if (V3.SP<0) {      V3.SP = -V3.SP;     V3.arah = mundur;   }
        else if(V3.SP>0){   V3.SP = V3.SP;          V3.arah = maju;
    }
        else                {       V3.SP = 0;              V3.arah = diam;
        }
```

```
            if(W==0){
                if((V1.SP>=V2.SP)&&(V1.SP>=V3.SP)){
                    V2.SP=(speed/V1.SP)*V2.SP;
                    V3.SP=(speed/V1.SP)*V3.SP;
                    V1.SP=(speed/V1.SP)*V1.SP;
                }
                else if((V2.SP>=V1.SP)&&(V2.SP>=V3.SP)){
                    V1.SP=(speed/V2.SP)*V1.SP;
                    V3.SP=(speed/V2.SP)*V3.SP;
                    V2.SP=(speed/V2.SP)*V2.SP;
                }
                else if((V3.SP>=V1.SP)&&(V3.SP>=V2.SP)){
                    V1.SP=(speed/V3.SP)*V1.SP;
                    V2.SP=(speed/V3.SP)*V2.SP;
                    V3.SP=(speed/V3.SP)*V3.SP;
                }
            }

            if(V1.SP<3)V1.SP=0;
            if(V2.SP<3)V2.SP=0;
            if(V3.SP<3)V3.SP=0;
}

void TIM3_CC(void)
{
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
        PrescalerValue = (uint32_t) ((SystemCoreClock / 2) / 5000000) - 1;

        /* Time base configuration */
        TIM_TimeBaseStructure.TIM_Period = 50000; //65535
        TIM_TimeBaseStructure.TIM_Prescaler = 0;
        TIM_TimeBaseStructure.TIM_ClockDivision = 0;
        TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
        TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

        /* Prescaler configuration */
        TIM_PrescalerConfig(TIM3, PrescalerValue, TIM_PSCReloadMode_Immediate);

        /* Output Compare Timing Mode configuration: Channel1 */
        TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
        TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
        TIM_OCInitStructure.TIM_Pulse = CCR1_Val;
        TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
        TIM_OC1Init(TIM3, &TIM_OCInitStructure);

        TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Disable);

        TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
        TIM_OCInitStructure.TIM_Pulse = CCR2_Val;
        TIM_OC2Init(TIM3, &TIM_OCInitStructure);

        TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Disable);
```

```
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = CCR3_Val;
    TIM_OC3Init(TIM3, &TIM_OCInitStructure);

    TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Disable);

    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = CCR4_Val;
    TIM_OC4Init(TIM3, &TIM_OCInitStructure);

    TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Disable);

    /* TIM Interrupts enable */
    TIM_ITConfig(TIM3, TIM_IT_CC1 | TIM_IT_CC2 | TIM_IT_CC3, ENABLE);
    TIM_ITConfig(TIM3, TIM_IT_CC4, ENABLE);

    /* TIM3 enable counter */
    TIM_Cmd(TIM3, ENABLE);

    NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

}

void TIM3_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM3, TIM_IT_CC1) == SET)
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);
        capture = TIM_GetCapture1(TIM3);
        TIM_SetCompare1(TIM3, capture + CCR1_Val);

        count1++;
        if(GetSudut_v==1)sudut_v=0;
        else if(GetSudut_v==8)sudut_v=45;
        else if(GetSudut_v==7)sudut_v=45*2;
        else if(GetSudut_v==6)sudut_v=45*3;
        else if(GetSudut_v==5)sudut_v=45*4;
        else if(GetSudut_v==4)sudut_v=45*5;
        else if(GetSudut_v==3)sudut_v=45*6;
        else if(GetSudut_v==2)sudut_v=45*7;
        else sudut_v=0;

//          if(GetSudut_v==1){
//              motor_pwm(motor1, maju, 0);
//              motor_pwm(motor2, maju, 0);
//              motor_pwm(motor3, maju, 0);
//          }
//          else if(GetSudut_v==2){
//              motor_pwm(motor1, maju, 25);
//              motor_pwm(motor2, maju, 25);
```

```
//                motor_pwm(motor3, maju, 25);
//          }
//          else if(GetSudut_v==3){
//                motor_pwm(motor1, maju, 50);
//                motor_pwm(motor2, maju, 50);
//                motor_pwm(motor3, maju, 50);
//          }
//          else if(GetSudut_v==4){
//                motor_pwm(motor1, maju, 75);
//                motor_pwm(motor2, maju, 75);
//                motor_pwm(motor3, maju, 75);
//          }
//          else if(GetSudut_v==5){
//                motor_pwm(motor1, maju, 100);
//                motor_pwm(motor2, maju, 100);
//                motor_pwm(motor3, maju, 100);
//          }

          switch(flag1){
          case 0 : if(re_center_flg==11) flag1=1;
                        else flag1=0; break;
          case 1 : if(re_center_flg==11) { yaw_flag=1; flag1=2; }
                        else flag1=0; break;
          case 2 : if(re_center_flg==11) { flag1=2;}
                        else flag1=0; break;
          }

          if(yaw_flag==1){
                S1.SP=new_yaw1;
                yaw_flag=2;
          }
          else {
                if(kec_w!=0)S1.SP=new_yaw1;
                else S1.SP=S1.SP;
          }
    }
    else if (TIM_GetITStatus(TIM3, TIM_IT_CC2) == SET)
    {
          TIM_ClearITPendingBit(TIM3, TIM_IT_CC2);
          capture = TIM_GetCapture2(TIM3);
          TIM_SetCompare2(TIM3, capture + CCR2_Val);

          prt_1.prt = speed1*(6000/225);
          speed1=0;
          if(prt_1.prt==0)rt_1.Rpm=0;

          prt_2.prt = speed2*(6000/225);
          speed2=0;
          if (prt_2.prt==0)rt_2.Rpm=0;

          prt_3.prt = speed3*(6000/225);
          speed3=0;
          if (prt_3.prt==0)rt_3.Rpm=0;
    }
    else if (TIM_GetITStatus(TIM3, TIM_IT_CC3) == SET){
```

```
        TIM_ClearITPendingBit(TIM3, TIM_IT_CC3);
        capture = TIM_GetCapture3(TIM3);
        TIM_SetCompare3(TIM3, capture + CCR3_Val);

        if(vxy==123)start_flag=1;

        if(start_flag==1){
            if(kec_w==100)
            {
                motor_pwm(motor1, maju, 30);
                motor_pwm(motor2, maju, 30);
                motor_pwm(motor3, maju, 30);
            }
            else if(kec_w==-100){
                motor_pwm(motor1, mundur, 30);
                motor_pwm(motor2, mundur, 30);
                motor_pwm(motor3, mundur, 30);
            }
            else{
                PID_Sudut_rotasi();
                aksel();
                omni_kinematik(aksel_v,sudut_v,S1.set);

                PID_Kecepatan1(V1.SP,V1.arah);
                PID_Kecepatan2(V2.SP,V2.arah);
                PID_Kecepatan3(V3.SP,V3.arah);

                motor_pwm(motor1, V1.arah, (int)V1.set);
                motor_pwm(motor2, V2.arah, (int)V2.set);
                motor_pwm(motor3, V3.arah, (int)V3.set);
            }
        }


        PWM_1 = (uint16_t) (( V1.set * (tim_period - 1)) / 100);
        TIM1->CCR1 = PWM_1;
        PWM_2 = (uint16_t) (( V2.set * (tim_period - 1)) / 100);
        TIM1->CCR2 = PWM_2;
        PWM_3 = (uint16_t) (( V3.set * (tim_period - 1)) / 100);
        TIM1->CCR3 = PWM_3;

    }
    else
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_CC4);
        capture = TIM_GetCapture4(TIM3);
        TIM_SetCompare4(TIM3, capture + CCR4_Val);

        I2C_ReadMultiByte(CMPS11_ADDRESS,CMPS11_BEARING_16BIT_H,2);
        yaw = (int16_t)((int16_t)I2C1_DATA[0]<<8)|I2C1_DATA[1];
        yaw/=10;

    if(imu_take==0){
        fst_yaw = (int16_t)((int16_t)I2C1_DATA[0]<<8)|I2C1_DATA[1];
```

```
            fst_yaw/=10;
            if((fst_yaw>=0)&&(fst_yaw<=74)) new_yaw = (45*fst_yaw)/74;
            else if((fst_yaw>66)&&(fst_yaw<=103)) new_yaw = (90*fst_yaw)/103;
            else if((fst_yaw>103)&&(fst_yaw<=130)) new_yaw = (135*fst_yaw)/130;
            else if((fst_yaw>130)&&(fst_yaw<=157)) new_yaw = (180*fst_yaw)/157;
            else if((fst_yaw>157)&&(fst_yaw<=181)) new_yaw = (225*fst_yaw)/181;
            else if((fst_yaw>181)&&(fst_yaw<=214)) new_yaw = (270*fst_yaw)/214;
            else if((fst_yaw>214)&&(fst_yaw<=223)) new_yaw = (285*fst_yaw)/223;
            else if((fst_yaw>223)&&(fst_yaw<=238)) new_yaw = (300*fst_yaw)/238;
            else if((fst_yaw>238)&&(fst_yaw<=254)) new_yaw = (315*fst_yaw)/254;
            else if((fst_yaw>254)&&(fst_yaw<=281)) new_yaw = (330*fst_yaw)/281;
            else if((fst_yaw>281)&&(fst_yaw<=316)) new_yaw = (345*fst_yaw)/316;
            else if((fst_yaw>316)&&(fst_yaw<=345)) new_yaw = (355*fst_yaw)/345;
            else new_yaw=fst_yaw;
            imu_take=101;
        }
        else{
            yaw = (int16_t)((int16_t)I2C1_DATA[0]<<8)|I2C1_DATA[1];
            yaw/=10;
            if((yaw>=0)&&(yaw<=74)) new_yaw1 = (45*yaw)/74;
            else if((yaw>66)&&(yaw<=103)) new_yaw1 = (90*yaw)/103;
            else if((yaw>103)&&(yaw<=130)) new_yaw1 = (135*yaw)/130;
            else if((yaw>130)&&(yaw<=157)) new_yaw1 = (180*yaw)/157;
            else if((yaw>157)&&(yaw<=181)) new_yaw1 = (225*yaw)/181;
            else if((yaw>181)&&(yaw<=214)) new_yaw1 = (270*yaw)/214;
            else if((yaw>214)&&(yaw<=223)) new_yaw1 = (285*yaw)/223;
            else if((yaw>223)&&(yaw<=238)) new_yaw1 = (300*yaw)/238;
            else if((yaw>238)&&(yaw<=254)) new_yaw1 = (315*yaw)/254;
            else if((yaw>254)&&(yaw<=281)) new_yaw1 = (330*yaw)/281;
            else if((yaw>281)&&(yaw<=316)) new_yaw1 = (345*yaw)/316;
            else if((yaw>316)&&(yaw<=345)) new_yaw1 = (355*yaw)/345;
            else new_yaw1=yaw;

            new_yaw1 = new_yaw1-new_yaw;

                if(new_yaw1 < 0) new_yaw1 = new_yaw1 + 360;
                if(new_yaw1 > 360) new_yaw1 = new_yaw1 - 360;
        }
        }
}

void aksel (void)
{
    if((vxy>50)&&(GetAksel_v!=0))
    {
        if(aksel_v<5) aksel_v=20;
        else if(aksel_v>=GetAksel_v)aksel_v=GetAksel_v;
        else
        {
                if(count1>=1)
                {
                    aksel_v+=3;
                    count1=0;
                }
        }
        sudut_smt=sudut_v;
    }
    else
    {
```

```
            if(aksel_v>40)
            {
                aksel_v--;
                sudut_v=sudut_smt;
            }
            else aksel_v=0;
        }
}

void pwm_motor_config(uint16_t frek){
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_OCInitTypeDef TIM_OCInitStructure;
    TIM_BDTRInitTypeDef TIM_BDTRInitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
    init_IO(RCC_AHB1Periph_GPIOE, P_9|P_11|P_13, GPIO_Mode_AF, GPIO_OType_PP,
GPIO_PuPd_UP); //pwm

    GPIO_PinAFConfig(GPIOE, GPIO_PinSource9, GPIO_AF_TIM1);
    GPIO_PinAFConfig(GPIOE, GPIO_PinSource11, GPIO_AF_TIM1);
    GPIO_PinAFConfig(GPIOE, GPIO_PinSource13, GPIO_AF_TIM1);

    tim_period = (1000000/frek)-1;

    TIM_TimeBaseStructure.TIM_Prescaler = 167;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseStructure.TIM_Period = tim_period;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);

    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 0;
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
    TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_Low;
    TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
    TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;
    TIM_OC1Init(TIM1, &TIM_OCInitStructure);

    TIM_BDTRInitStructure.TIM_OSSRState = TIM_OSSRState_Enable;
    TIM_BDTRInitStructure.TIM_OSSIState = TIM_OSSIState_Enable;
    TIM_BDTRInitStructure.TIM_LOCKLevel = TIM_LOCKLevel_1;
    TIM_BDTRInitStructure.TIM_DeadTime = 11;
    TIM_BDTRInitStructure.TIM_Break = TIM_Break_Enable;
    TIM_BDTRInitStructure.TIM_BreakPolarity = TIM_BreakPolarity_High;
    TIM_BDTRInitStructure.TIM_AutomaticOutput = TIM_AutomaticOutput_Enable;
    TIM_BDTRConfig(TIM1, &TIM_BDTRInitStructure);

    TIM_OC1Init(TIM1, &TIM_OCInitStructure);
    TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);
```

```
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 0;

    TIM_OC2Init(TIM1, &TIM_OCInitStructure);
    TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);

    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 0;

    TIM_OC3Init(TIM1, &TIM_OCInitStructure);
    TIM_OC3PreloadConfig(TIM1, TIM_OCPreload_Enable);

    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse = 0;

    TIM_ARRPreloadConfig(TIM1, ENABLE);

    TIM_Cmd(TIM1, ENABLE);

    TIM_CtrlPWMOutputs(TIM1, ENABLE);
}

void EXTI_config(void){
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    TIM_TimeBaseStructure.TIM_Period = 1000000;
    TIM_TimeBaseStructure.TIM_Prescaler = 83;
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV2;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

    TIM_Cmd(TIM2, ENABLE);

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

  EXTI_InitStruct.EXTI_Line = EXTI_Line0|EXTI_Line1|EXTI_Line2;
  EXTI_InitStruct.EXTI_LineCmd = ENABLE;
  EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_Init(&EXTI_InitStruct);

  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
  NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
  NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x00;
  NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
  NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStruct);

  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource1);
  NVIC_InitStruct.NVIC_IRQChannel = EXTI1_IRQn ;
  NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x01;
```

```
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x01;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);

    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource2);
    NVIC_InitStruct.NVIC_IRQChannel = EXTI2_IRQn ;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x02;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x02;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
}

void EXTI0_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line0) != RESET) {
      speed1++;
      if(GPIO_ReadInputDataBit(GPIOB, P_0) == 0)rt_1.encoder--;
      else rt_1.encoder++;

      if(rt_1.CaptureNumber == 0){
            rt_1.EXTI_ReadValue1 = TIM2->CNT;
            rt_1.CaptureNumber = 1;
            }
      else if(rt_1.CaptureNumber == 1){
            rt_1.EXTI_ReadValue2 = TIM2->CNT;
    // Hitung Data Periode
            if (rt_1.EXTI_ReadValue2 > rt_1.EXTI_ReadValue1)rt_1.Capture =
(rt_1.EXTI_ReadValue2 - rt_1.EXTI_ReadValue1);
            else if (rt_1.EXTI_ReadValue2 < rt_1.EXTI_ReadValue1)rt_1.Capture =
((TIM2 -> ARR - rt_1.EXTI_ReadValue1) + rt_1.EXTI_ReadValue2);
            else rt_1.Capture = 0;
    // Hitung Frekuensi
            rt_1.Freq = 1000000 / rt_1.Capture;
            rt_1.Rpm= (rt_1.Freq*60)/rt_1.ppr;
            rt_1.CaptureNumber = 0;
      }
       EXTI_ClearITPendingBit(EXTI_Line0);
    }
}

void EXTI1_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line1) != RESET) {
      speed2++;
      if(GPIO_ReadInputDataBit(GPIOB, P_1) == 0)rt_2.encoder--;
      else rt_2.encoder++;

      if(rt_2.CaptureNumber == 0){
            rt_2.EXTI_ReadValue1 = TIM2->CNT;
            rt_2.CaptureNumber = 1;
            }
      else if(rt_2.CaptureNumber == 1){
            rt_2.EXTI_ReadValue2 = TIM2->CNT;
    // Hitung Data Periode
```

```
            if (rt_2.EXTI_ReadValue2 > rt_2.EXTI_ReadValue1)rt_2.Capture =
(rt_2.EXTI_ReadValue2 - rt_2.EXTI_ReadValue1);
            else if (rt_2.EXTI_ReadValue2 < rt_2.EXTI_ReadValue1)rt_2.Capture =
((TIM2 -> ARR - rt_2.EXTI_ReadValue1) + rt_2.EXTI_ReadValue2);
            else rt_2.Capture = 0;
    // Hitung Frekuensi
            rt_2.Freq = 1000000 / rt_2.Capture;
            rt_2.Rpm= (rt_2.Freq*60)/rt_2.ppr;
            rt_2.CaptureNumber = 0;
      }
        EXTI_ClearITPendingBit(EXTI_Line1);
    }
}

void EXTI2_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line2) != RESET) {
      speed3++;
      if(GPIO_ReadInputDataBit(GPIOB, P_4) == 0)rt_3.encoder--;
      else rt_3.encoder++;

      if(rt_3.CaptureNumber == 0){
            rt_3.EXTI_ReadValue1 = TIM2->CNT;
            rt_3.CaptureNumber = 1;
            }
      else if(rt_3.CaptureNumber == 1){
            rt_3.EXTI_ReadValue2 = TIM2->CNT;
    // Hitung Data Periode
            if (rt_3.EXTI_ReadValue2 > rt_3.EXTI_ReadValue1)rt_3.Capture =
(rt_3.EXTI_ReadValue2 - rt_3.EXTI_ReadValue1);
            else if (rt_3.EXTI_ReadValue2 < rt_3.EXTI_ReadValue1)rt_3.Capture =
((TIM2 -> ARR - rt_3.EXTI_ReadValue1) + rt_3.EXTI_ReadValue2);
            else rt_3.Capture = 0;
    // Hitung Frekuensi
            rt_3.Freq = 1000000 / rt_3.Capture;
            rt_3.Rpm= (rt_3.Freq*60)/rt_3.ppr;
            rt_3.CaptureNumber = 0;
      }
        EXTI_ClearITPendingBit(EXTI_Line2);
    }
}

void motor_pwm(uint16_t motor, uint8_t arah, uint16_t pwm)
{
      if(motor==motor1){
            if(arah==maju){
                  Pin(GPIOB, ON, P_15);
                  Pin(GPIOD, OFF, P_9);
                  V1.set=pwm;
            }
            else if(arah==mundur){
                  Pin(GPIOB, OFF, P_15);
                  Pin(GPIOD, ON, P_9);
```

```
                V1.set=pwm;
            }
            else if(arah==diam){
                Pin(GPIOB, ON, P_15);
                Pin(GPIOD, ON, P_9);
                V1.set=100;
            }
        }
        if(motor==motor2){
            if(arah==mundur){
                Pin(GPIOD, ON, P_7);
                Pin(GPIOB, OFF, P_3);
                V2.set=pwm;
            }
            else if(arah==maju){
                Pin(GPIOD, OFF, P_7);
                Pin(GPIOB, ON, P_3);
                V2.set=pwm;
            }
            else if(arah==diam){
                Pin(GPIOD, ON, P_7);
                Pin(GPIOB, ON, P_3);
                V2.set=100;
            }
        }

        if(motor==motor3){
            if(arah==maju){
                Pin(GPIOD, ON, P_1);
                Pin(GPIOD, OFF, P_3);
                V3.set=pwm;
            }
            else if(arah==mundur){
                Pin(GPIOD, OFF, P_1);
                Pin(GPIOD, ON, P_3);
                V3.set=pwm;
            }
            else if(arah==diam){
                Pin(GPIOD, ON, P_1);
                Pin(GPIOD, ON, P_3);
                V3.set=100;
            }
        }
    }

    void PID_Kecepatan1(int RPM, int arah)
    {
        V1.SP = RPM;
        V1.arah = arah;
        V1.error = V1.SP - rt_1.Rpm;
        V1.P = V1.Kp * V1.error;
        V1.I = (V1.Ki * (V1.error + V1.last1_error)/2);
        V1.D = (V1.Kd * (V1.error - 2*V1.last1_error+V1.last2_error));
```

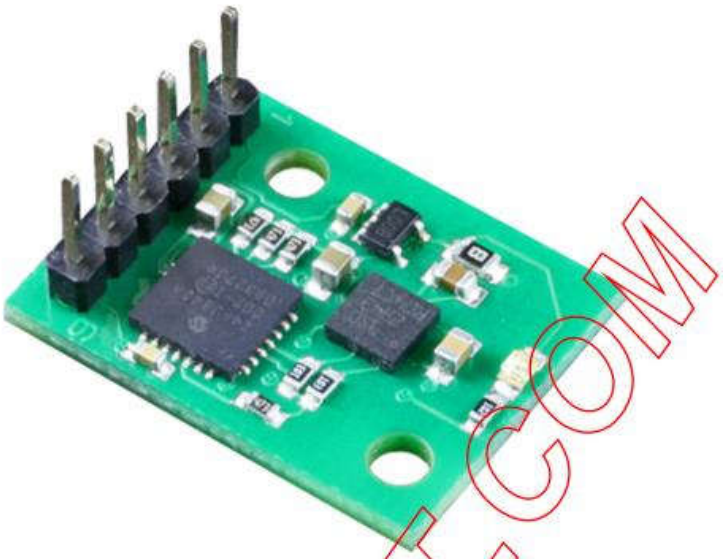
```
    V1.MV = V1.P + V1.I + V1.D;
    V1.temp = V1.set + V1.MV;

    if(V1.temp>=V1.MAX)V1.set=V1.MAX;
    else if(V1.temp<=V1.MIN)V1.set=V1.MIN;
    else V1.set=V1.temp;

    V1.last2_error = V1.last1_error;
    V1.last1_error = V1.error;

}
void PID_Kecepatan2(int RPM, int arah)
{
    V2.SP = RPM;
    V2.arah = arah;
    V2.error = V2.SP - rt_2.Rpm;
    V2.P = V2.Kp * V2.error;
    V2.I = (V2.Ki * (V2.error + V2.last1_error)/2);
    V2.D = (V2.Kd * (V2.error - 2*V2.last1_error + V2.last2_error));

    V2.MV = V2.P + V2.I + V2.D;
    V2.temp = V2.set + V2.MV;

    V2.last2_error = V2.last1_error;
    V2.last1_error = V2.error;

    if(V2.temp>=V2.MAX)V2.set=V2.MAX;
    else if(V2.temp<=V2.MIN)V2.set=V2.MIN;
    else V2.set=V2.temp;

}
void PID_Kecepatan3(int RPM, int arah)
{
    V3.SP = RPM;
    V3.arah = arah;

    V3.error = V3.SP - rt_3.Rpm;

    V3.P = V3.Kp * V3.error;
    V3.I = (V3.Ki * (V3.error + V3.last1_error)/2);
    V3.D = (V3.Kd * (V3.error - 2*V3.last1_error + V3.last2_error));
    V3.MV = V3.P + V3.I + V3.D;
    V3.temp = V3.set + V3.MV;

    V3.last2_error = V3.last1_error;
    V3.last1_error = V3.error;

    if(V3.temp>=V3.MAX)V3.set=V3.MAX;
    else if(V3.temp<=V3.MIN)V3.set=V3.MIN;
    else V3.set=V3.temp;
}
```

```
void PID_Sudut_rotasi(void){
    S1.error = S1.SP - new_yaw1;
    S1.P = S1.Kp * (S1.error - S1.last1_error);
    S1.I = (S1.Ki * (S1.error + S1.last1_error))/3;
    S1.D = (S1.Kd * (S1.error - 2*S1.last1_error + S1.last2_error));
    S1.MV = S1.P + S1.I + S1.D;
    S1.temp = S1.MV;
    S1.last2_error = S1.last1_error;
    S1.last1_error = S1.error;

    if(S1.temp<-8){
    if((-1*S1.temp)>=S1.MAX)S1.set=S1.MAX;
    else S1.set=-1*S1.temp;
    }
    else if(S1.temp>=8){
    if(S1.temp>=S1.MAX)S1.set=S1.MAX;
    else S1.set=-S1.temp;
    }
    else {
    S1.set=0;
    }
}

void UU_PutChar(USART_TypeDef* USARTx, uint8_t ch)
{
    while(!(USARTx->SR & USART_SR_TXE));
    USARTx->DR = ch;
}

void UU_PutString(USART_TypeDef* USARTx, uint8_t * str)
{
  while(*str != 0)
  {
    UU_PutChar(USARTx, *str);
    str++;
  }
}

void UU_PutNumber(USART_TypeDef* USARTx, uint32_t x)
{
    char value[10]; //a temp array to hold results of conversion
    int i = 0; //loop index

    do
    {
        value[i++] = (char)(x % 10) + '0'; //convert integer to character
        x /= 10;
    } while(x);

    while(i) //send data
    {
        UU_PutChar(USARTx, value[--i]);
    }
}
```

LAMPIRAN 4
DATASHEET

1. *Datasheet* CMPS-11

CMPS11 - Tilt Compensated Compass Module



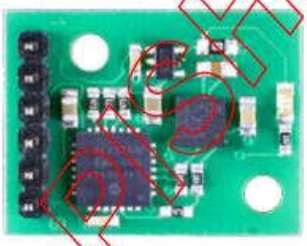
Introduction

The CMPS11 is our 3rd generation tilt compensated compass. Employing a 3-axis magnetometer, a 3-axis gyro and a 3-axis accelerometer. A Kalman filter combines the gyro and accelerometer to remove the errors caused by tilting of the PCB. The CMPS11 produces a result of 0-3599 representing 0-359.9 or 0 to 255. The output of the three sensors measuring x, y and z components of the magnetic field, together with the pitch and roll are used to calculate the bearing, each of these components are also made available in there raw form. The CMPS11 module requires a power supply at 3.6 - 5v and draws a nominal 25mA of current. A choice of serial or I2C interfaces are provided.

Mode selection

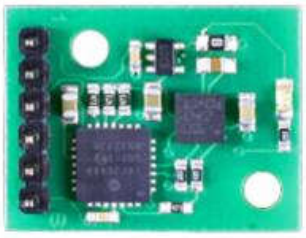
For data on each mode please click the mode heading. Note the CMPS11 looks at the mode selection pin at power-up only.

I2C mode



To enter the I2C mode of operation leave the mode pin unconnected

Serial mode



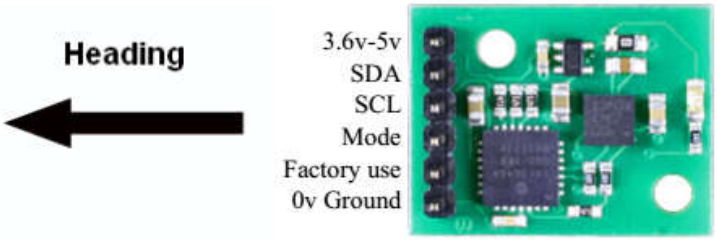
To enter the serial mode of operation connect the mode pin to ground

The **Factory use** pin is for our internal testing only, this pin should remain unconnected.

CMPS11 - Tilt Compensated Compass Module

I2C mode

Connections



To enter the I2C mode of operation leave the mode pin unconnected

I2C Communication

I2C communication protocol with the compass module is the same as popular eeprom's such as the 24C04. First send a start bit, the module address with the read/write bit low, then the register number you wish to read. This is followed by a repeated start and the module address again with the read/write bit high. You now read one or two bytes for 8bit or 16bit registers respectively. 16bit registers are read high byte first. The compass has a 28 byte array of registers. organized as below:

Register	Function
0	Command register (write) / Software version (read)
1	Compass Bearing 8 bit, i.e. 0-255 for a full circle
2,3	Compass Bearing 16 bit, i.e. 0-3599, representing 0-359.9 degrees. register 2 being the high byte
4	Pitch angle - signed byte giving angle in degrees from the horizontal plane, Kalman filtered with Gyro
5	Roll angle - signed byte giving angle in degrees from the horizontal plane, Kalman filtered with Gyro
6,7	Magnetometer X axis raw output, 16 bit signed integer with register 6 being the upper 8 bits
8,9	Magnetometer Y axis raw output, 16 bit signed integer with register 8 being the upper 8 bits
10,11	Magnetometer Z axis raw output, 16 bit signed integer with register 10 being the upper 8 bits
12,13	Accelerometer X axis raw output, 16 bit signed integer with register 12 being the upper 8 bits
14,15	Accelerometer Y axis raw output, 16 bit signed integer with register 14 being the upper 8 bits
16,17	Accelerometer Z axis raw output, 16 bit signed integer with register 16 being the upper 8 bits
18,19	Gyro X axis raw output, 16 bit signed integer with register 18 being the upper 8 bits
20,21	Gyro Y axis raw output, 16 bit signed integer with register 20 being the upper 8 bits
22,23	Gyro Z axis raw output, 16 bit signed integer with register 22 being the upper 8 bits

24,25	Temperature raw output, 16 bit signed integer with register 24 being the upper 8 bits
26	Pitch angle - signed byte giving angle in degrees from the horizontal plane (no Kalman filter)
27	Roll angle - signed byte giving angle in degrees from the horizontal plane (no Kalman filter)

Register 0 is a dual action register, in the event of a read the CMPS11 will reply with the software version, for a write it acts as the command register and is used to calibrate the compass, change address and if necessary restore the factory default calibration. Register 1 is the bearing converted to a 0-255 value, this may be easier for some applications than 0-3599 which requires two bytes. For those who require better resolution registers 2 and 3 (high byte first) form a 16 bit unsigned integer in the range 0-3599. This represents 0-359.9°. Register 4 is the pitch angle, giving an angle of 0 when the board is flat and up to +/- 85° at maximum tilt in either direction and also features a Kalman filter that uses the gyro sensor to reduce errors caused by unwanted acceleration effects (such as shake). Register 5 works the same way but with results for the Roll angle. There is then an array of registers (6-25) providing all the raw sensor data from the magnetic and acceleration sensors. Finally we have included the pitch and roll angle with no Kalman filter in registers 26 and 27. The raw output values in registers 6-25 are exactly what we get from the LSM9DS0 sensor chip, customers wishing to make use of these should consult the ST data sheet for further information.

Calibration of the CMPS11

Please do not do this until you have I2C communication fully working. I would recommend evaluating the CMPS11 performance first before implementing this function. Its purpose is to remove sensor gain and offset of both magnetometer and accelerometer and achieves this by looking for maximum sensor outputs. First of all you need to enter the calibration mode by sending a 3 byte sequence of 0xF0,0xF5 and then 0xF6 to the command register, these MUST be sent in 3 separate I2C frames, you cannot send them all at once. There MUST be a minimum of 20ms between each I2C frame. An I2C frame is [start sequence] [I2C address] [register address] [command byte] [stop sequence]. The LED will then extinguish and the CMPS11 should now be rotated in all directions in 3 dimensions, if a new maximum for any of the sensors is detected then the LED will flash, when you cannot get any further LED flashes in any direction then exit the calibration mode with a command of 0xF8. Please make sure that the CMPS11 is not located near to ferrous objects as this will distort the magnetic field and induce errors in the reading. While calibrating rotate the compass slowly. Remember the axis of the magnetic field is unlikely to be horizontal, it dips into the earth at an angle which varies depending on your location. At our offices in the UK it dips into the earth at 67 degrees and that is the orientation each axis of the compass needs to be to find the maximums. You need to find both positive and negative maximums for each axis so there are 6 points to calibrate. The accelerometer is also calibrated at the same time, so the module should also be positioned horizontal, inverted, and on all 4 sides to calibrate the 6 accelerometer points. Each accelerometer point needs to be stable for 200mS for its reading to be used for calibration. This delay is deliberate so that light taps to the module do not produces disruptive accelerometer readings which would mess up the pitch and roll angles. There is no delay for the magnetic points. The performance of the module is directly related to how well you perform calibration so do this slowly and carefully.

2. *Datasheet* L298N

L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)

Multiwatt15

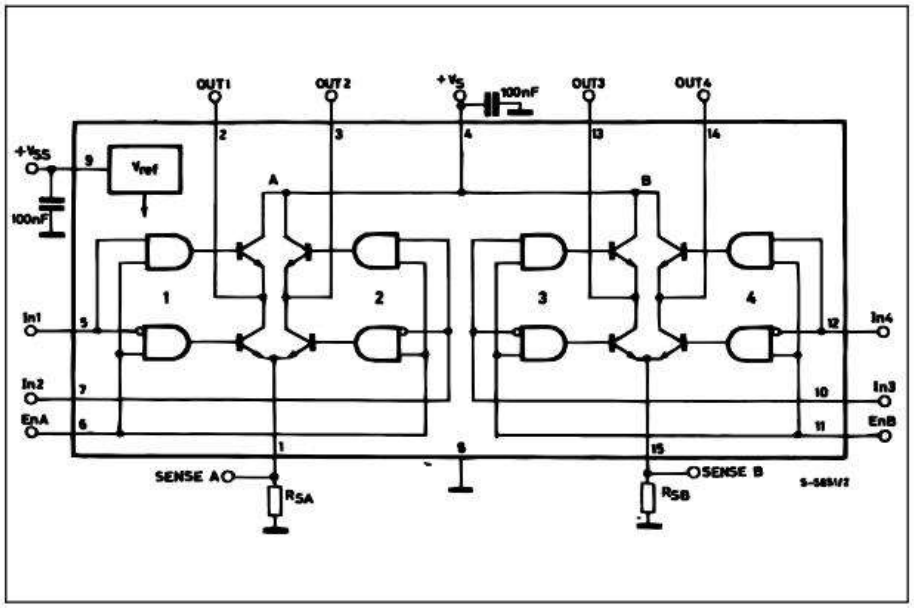
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

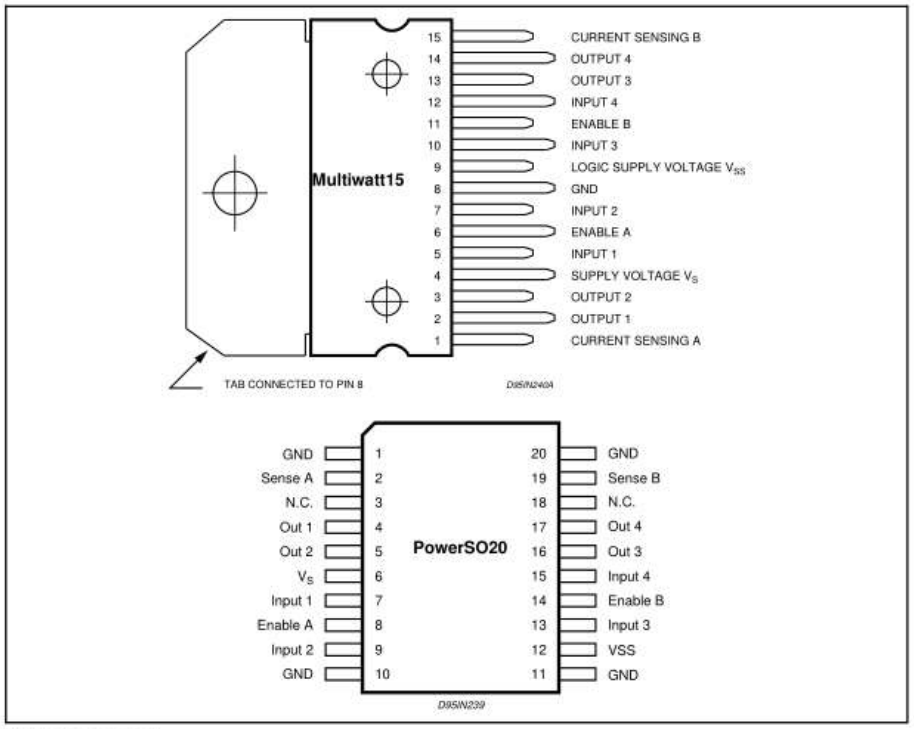
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_i,V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminum substrate

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition		V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)			4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0	V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L	V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0	V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L	V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)			–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)			2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L				–10	µA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	V_i = H ≤ V_{SS} –0.6V			30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)			–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)			2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L				–10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} –0.6V			30	100	µA
$V_{CEsat\ (H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A		0.95	1.35 2	1.7 2.7	V V
$V_{CEsat\ (L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)		0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)		1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)			–1 (1)		2	V

3. *Datasheet Bluetooth* HC-05

ITead Studio
Make Innovation easier

Tech Support: info@iteadstudio.com

HC-05

-Bluetooth to Serial Port Module

Overview

HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup.

Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH(Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm. Hope it will simplify your overall design/development cycle.

Specifications

Hardware features

- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector

Software features

- Default Baud rate: 38400, Data bits:8, Stop bit:1,Parity:No parity, Data control: has. Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected;
- PIO10 and PIO11 can be connected to red and blue led separately. When master and slave are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2times/s.
- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:"0000" as default
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.

PIN Name	PIN #	Pad type	Description	Note
GND	13 21 22	VSS	Ground pot	
3.3 VCC	12	3.3V	Integrated 3.3V (+) supply with On-chip linear regulator output within 3.15-3.3V	
AIO0	9	Bi-Directional	Programmable input/output line	
AIO1	10	Bi-Directional	Programmable input/output line	
PIO0	23	Bi-Directional RX EN	Programmable input/output line, control output for LNA(if fitted)	
PIO1	24	Bi-Directional TX EN	Programmable input/output line, control output for PA(if fitted)	

4. *Datasheet* Mikrokontroler STM32F407VG

lite.augmented

UM1472
User manual

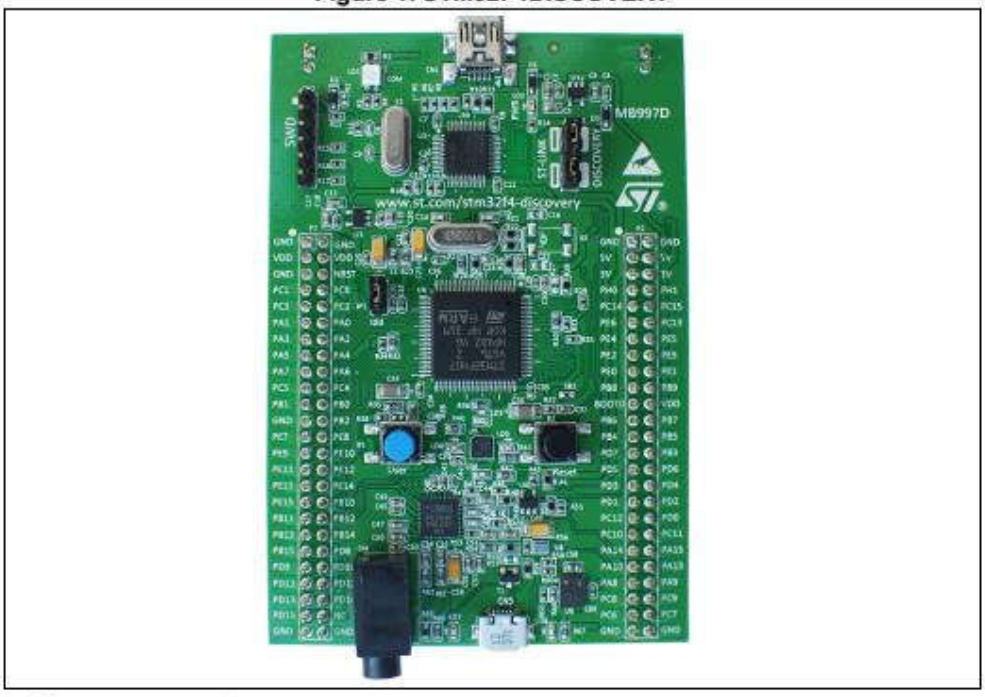
Discovery kit with STM32F407VG MCU

Introduction

The STM32F4DISCOVERY Discovery kit allows users to easily develop applications with the STM32F407 high performance microcontroller with ARM® Cortex®-M4 32-bit core. It includes everything required either for beginners or for experienced users to get quickly started.

Based on the STM32F407VGT6, it includes an ST-LINK/V2 or ST-LINK/V2-A embedded debug tool, two ST MEMS digital accelerometers, a digital microphone, one audio DAC with integrated class D speaker driver, LEDs and push buttons and an USB OTG micro-AB connector. To expand the functionality of the STM32F4DISCOVERY Discovery kit with the Ethernet connectivity, LCD display and more, visit the www.st.com/stm32f4dis-expansion webpage. The STM32F4DISCOVERY Discovery kit comes with the STM32 comprehensive software HAL library, together with various packaged software examples, as well as a direct access to the ARM® mbed™ on-line resources at http://mbed.org.

Figure 1. STM32F4DISCOVERY



1. Picture not contractual

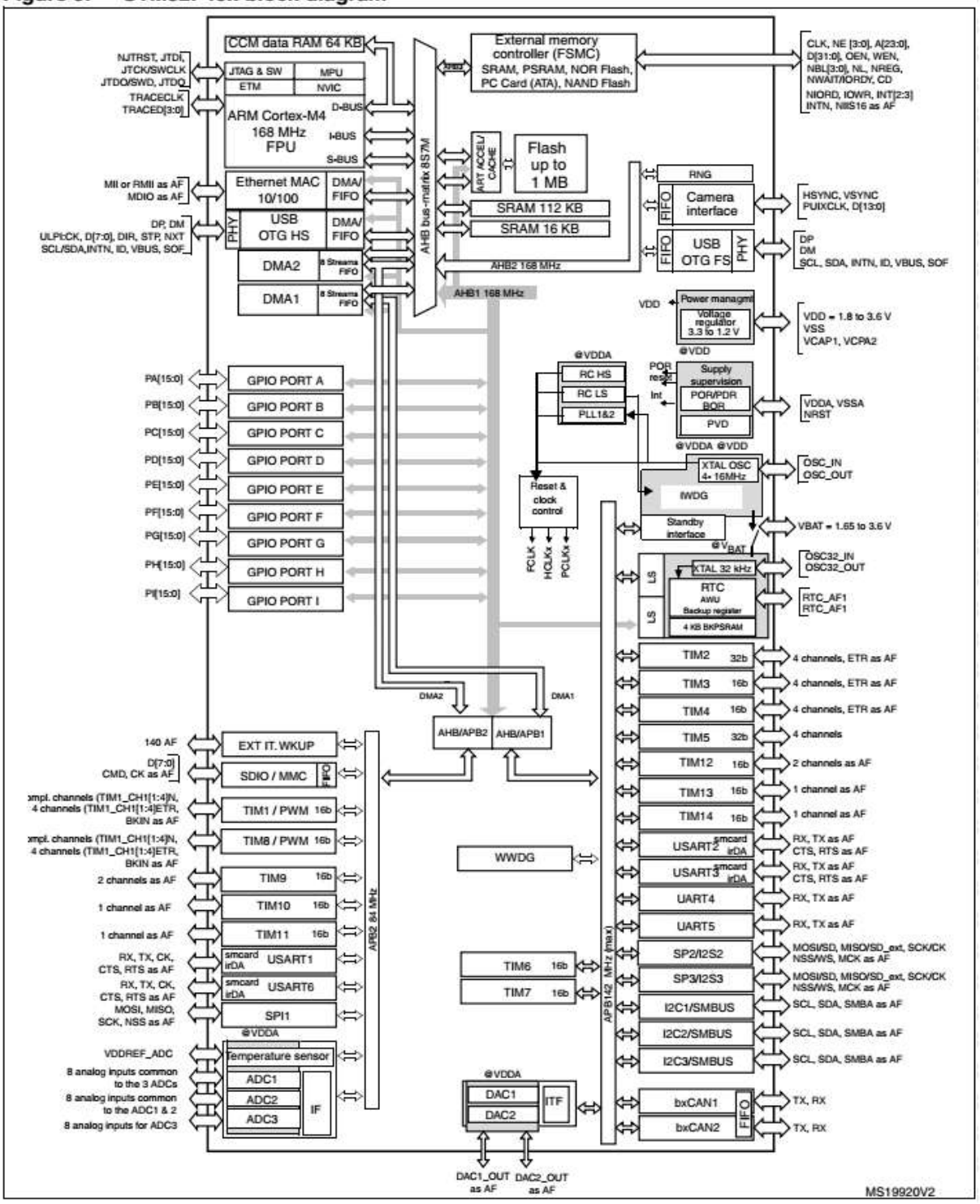
ARM mbed enabled

Features

The STM32F4DISCOVERY offers the following features:

- STM32F407VGT6 microcontroller featuring 32-bit ARM Cortex® -M4 with FPU core, 1-Mbyte Flash memory, 192-Kbyte RAM in an LQFP100 package
- On-board ST-LINK/V2 on STM32F4DISCOVERY or ST-LINK/V2-A on STM32F407G-DISC1
- ARM® mbed™ -enabled (http://mbed.org) with ST-LINK/V2-A only
- USB ST-LINK with re-enumeration capability and three different interfaces:
  - virtual com port (with ST-LINK/V2-A only)
  - mass storage (with ST-LINK/V2-A only)
  - debug port
- Board power supply:
  - Through USB bus
  - External power sources:
    3 V and 5 V
- LIS302DL or LIS3DSH ST MEMS 3-axis accelerometer
- MP45DT02 ST MEMS audio sensor omni-directional digital microphone
- CS43L22 audio DAC with integrated class D speaker driver
- Eight LEDs:
  - LD1 (red/green) for USB communication
  - LD2 (red) for 3.3 V power on
  - Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue)
  - 2 USB OTG LEDs LD7 (green) VBUS and LD8 (red) over-current
- Two push buttons (user and reset)
- USB OTG FS with micro-AB connector
- Extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing
- Comprehensive free software including a variety of examples, part of STM32CubeF4 package or STSW-STM32068 for legacy standard libraries usage

Figure 5. STM32F40x block diagram



1. The timers connected to APB2 are clocked from TIMxCLK up to 168 MHz, while the timers connected to APB1 are clocked

Table 3. Timer feature comparison

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max interface clock (MHz)	Max timer clock (MHz)
Advanced-control	TIM1, TIM8	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	168
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	168
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	168
	TIM12	16-bit	Up	Any integer between 1 and 65536	No	2	No	42	84
	TIM13, TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No	42	84
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No	42	84

Advanced-control timers (TIM1, TIM8)

The advanced-control timers (TIM1, TIM8) can be seen as three-phase PWM generators multiplexed on 6 channels. They have complementary PWM outputs with programmable inserted dead times. They can also be considered as complete general-purpose timers. Their 4 independent channels can be used for:

- Input capture
- Output compare
- PWM generation (edge- or center-aligned modes)
- One-pulse mode output

If configured as standard 16-bit timers, they have the same features as the general-purpose TIMx timers. If configured as 16-bit PWM generators, they have full modulation capability (0-100%).

The advanced-control timer can work together with the TIMx timers via the Timer Link feature for synchronization or event chaining.

TIM1 and TIM8 support independent DMA request generation.

General-purpose timers (TIMx)

There are ten synchronizable general-purpose timers embedded in the STM32F40x devices (see *Table 3* for differences).

- **TIM2, TIM3, TIM4, TIM5**

  The STM32F40x include 4 full-featured general-purpose timers: TIM2, TIM5, TIM3, and TIM4.The TIM2 and TIM5 timers are based on a 32-bit auto-reload up/downcounter and a 16-bit prescaler. The TIM3 and TIM4 timers are based on a 16-bit auto-reload up/downcounter and a 16-bit prescaler. They all feature 4 independent channels for input capture/output compare, PWM or one-pulse mode output. This gives up to 16 input capture/output compare/PWMs on the largest packages.

  The TIM2, TIM3, TIM4, TIM5 general-purpose timers can work together, or with the other general-purpose timers and the advanced-control timers TIM1 and TIM8 via the Timer Link feature for synchronization or event chaining.

  Any of these general-purpose timers can be used to generate PWM outputs.

  TIM2, TIM3, TIM4, TIM5 all have independent DMA request generation. They are capable of handling quadrature (incremental) encoder signals and the digital outputs from 1 to 4 hall-effect sensors.

- **TIM9, TIM10, TIM11, TIM12, TIM13, and TIM14**

  These timers are based on a 16-bit auto-reload upcounter and a 16-bit prescaler. TIM10, TIM11, TIM13, and TIM14 feature one independent channel, whereas TIM9 and TIM12 have two independent channels for input capture/output compare, PWM or one-pulse mode output. They can be synchronized with the TIM2, TIM3, TIM4, TIM5 full-featured general-purpose timers. They can also be used as simple time bases.

Basic timers TIM6 and TIM7

These timers are mainly used for DAC trigger and waveform generation. They can also be used as a generic 16-bit time base.

TIM6 and TIM7 support independent DMA request generation.

Independent watchdog

The independent watchdog is based on a 12-bit downcounter and 8-bit prescaler. It is clocked from an independent 32 kHz internal RC and as it operates independently from the main clock, it can operate in Stop and Standby modes. It can be used either as a watchdog to reset the device when a problem occurs, or as a free-running timer for application timeout management. It is hardware- or software-configurable through the option bytes.

Window watchdog

The window watchdog is based on a 7-bit downcounter that can be set as free-running. It can be used as a watchdog to reset the device when a problem occurs. It is clocked from the main clock. It has an early warning interrupt capability and the counter can be frozen in debug mode.

Inter-integrated circuit interface (I²C)

Up to three I²C bus interfaces can operate in multimaster and slave modes. They can support the Standard- and Fast-modes. They support the 7/10-bit addressing mode and the 7-bit dual addressing mode (as slave). A hardware CRC generation/verification is embedded.

They can be served by DMA and they support SMBus 2.0/PMBus.

Universal synchronous/asynchronous receiver transmitters (USART)

The STM32F405xx and STM32F407xx embed four universal synchronous/asynchronous receiver transmitters (USART1, USART2, USART3 and USART6) and two universal asynchronous receiver transmitters (UART4 and UART5).

These six interfaces provide asynchronous communication, IrDA SIR ENDEC support, multiprocessor communication mode, single-wire half-duplex communication mode and have LIN Master/Slave capability. The USART1 and USART6 interfaces are able to communicate at speeds of up to 10.5 Mbit/s. The other available interfaces communicate at up to 5.25 bit/s.

USART1, USART2, USART3 and USART6 also provide hardware management of the CTS and RTS signals, Smart Card mode (ISO 7816 compliant) and SPI-like communication capability. All interfaces can be served by the DMA controller.

Table 6. STM32 pin description versus board functions

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
BOOT0	VPP	94	-	-	-	-	-	-	-	-	-	-	-	-	-	21
NRST	-	14	-	-	-	RESET	-	NRST	-	-	-	-	-	5	6	-
PA0-WKUP	USART2_CTS/ USART4_TX/ ETH_MII_CRS/ TIM2_CH1_ETR/ TIM5_CH1/ TIM8_ETR/ ADC123_IN0/ WKUP	23	-	-	-	USER	-	-	-	-	-	-	-	-	12	-
PA1	USART2_RTS/ USART4_RX/ ETH_RMII_REF_CLK/ ETH_MII_RX_CLK/ TIM5_CH2/ TIMM2_CH2/ ADC123_IN1	24	-	-	-	-	-	-	-	-	-	-	-	-	11	-

Table 6. STM32 pin description versus board functions (continued)

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PA2	USART2_TX/ TIM5_CH3/ TIM9_CH1/ TIM2_CH3/ ETH_MDIO/ ADC123_IN2	25	-	-	-	-	-	-	-	-	-	-	-	-	14	-
PA3	USART2_RX/ TIM5_CH4/ TIM9_CH2/ TIM2_CH4/ OTG_HS_ULPI_D0/ ETH_MII_COL/ ADC123_IN3	26	-	-	-	-	-	-	-	-	-	-	-	-	13	-
PA4	SPI1_NSS/ SPI3_NSS/ USART2_CK/ DCMI_HSYNC/ OTG_HS_SOF/ I2S3_WS/ ADC12_IN4/ DAC1_OUT	29	LRCK/AIN1x	-	-	-	-	-	-	-	-	-	-	-	16	-
PA5	SPI1_SCK/ OTG_HS_ULPI_CK/ TIM2_CH1_ETR/ TIM8_CHIN/ ADC12_IN5/ DAC2_OUT	30	-	-	SCL/SPC	-	-	-	-	-	-	-	-	-	15	-
PA6	SPI1_MISO/ TIM8_BKIN/ TIM13_CH1/ DCMI_PIXCLK/ TIM3_CH1/ TIM1_BKIN/ ADC12_IN6	31	-	-	SDO	-	-	-	-	-	-	-	-	-	18	-
PA7	SPI1_MOSI/ TIM8_CH1N/ TIM14_CH1TIM3_CH2/ ETH_MII_RX_DV/ TIM1_CH1N/ RMII_CRS_DV/ ADC12_IN7	32	-	-	SDA/SDI/SDO	-	-	-	-	-	-	-	-	-	17	-

Table 6. STM32 pin description versus board functions (continued)

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PA8	MCO1/ USART1_CK/ TIM1_CH1/ I2C3_SCL/ OTG_FS_SOF	67	-	-	-	-	-	-	-	-	-	-	-	-	-	43
PA9	USART1_TX/ TIM1_CH2/ I2C3_SMBA/ DCMI_D0/ OTG_FS_VBUS	68	-	-	-	-	GREEN	-	VBUS	-	-	-	1	-	-	44
PA10	USART1_RX/ TIM1_CH3/ OTG_FS_ID/ DCMI_D1	69	-	-	-	-	-	-	ID	-	-	-	4	-	-	41
PA11	USART1_CTS/ CAN1_RX/ TIM1_CH4/ OTG_FS_DM	70	-	-	-	-	-	-	DM	-	-	-	2	-	-	-
PA12	USART1_RTS/ CAN1_TX/ TIM1_ETR/ OTG_FS_DP	71	-	-	-	-	-	-	DP	-	-	-	3	-	-	-
PA13	JTMS-SWDIO	72	-	-	-	-	-	SWDIO	-	-	-	-	-	4	-	42
PA14	JTCK-SWCLK	76	-	-	-	-	-	SWCLK	-	-	-	-	-	2	-	39
PA15	JTDI/ SPI3_NSS/ I2S3_WS/ TIM2_CH1_ETR/ SPI1_NSS	77	-	-	-	-	-	-	-	-	-	-	-	-	-	40
PB0	TIM3_CH3/ TIM8_CH2N/ OTG_HS_ULPI_D1/ ETH_MII_RXD2/ TIM1_CH2N/ ADC12_IN8	35	-	-	-	-	-	-	-	-	-	-	-	-	22	-

Table 6. STM32 pin description versus board functions (continued)

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PB1	TIM3_CH4/ TIM8_CH3N/ OTG_HS_ULPI_D2/ ETH_MII_RXD3/ OTG_HS_INTN/ TIM1_CH3N/ ADC12_IN9	36	-	-	-	-	-	-	-	-	-	-	-	-	21	-
PB2	-	37	-	-	-	-	-	-	-	-	-	-	-	-	24	-
PB3	JTDO/ TRACESWO/ SPI3_SCK/ I2S3_CK/ TIM2_CH2/ SPI1_SCK	89	-	-	-	-	-	SWO	-	-	-	-	-	6		28
PB4	NJTRST/ SPI3_MISO/ TIM3_CH1/ SPI1_MISO/ I2S3ext_SD	90	-	-	-	-	-	-	-	-	-	-	-	-	-	25
PB5	I2C1_SMBA/ CAN2_RX/ OTG_HS_ULPI_D7/ ETH_PPS_OUT/ TIM3_CH2/ SPI1_MOSI/ SPI3_MOSI/ DCMI_D10/ I2S3_SD	91	-	-	-	-	-	-	-	-	-	-	-	-	-	26
PB6	I2C1_SCL/ TIM4_CH1/ CAN2_TX/ OTG_FS_INTN/ DCMI_D5/ USART1_TX	92	SCL	-	-	-	-	-	-	-	-	-	-	-	-	23
PB7	I2C1_SDA/ FSMC_NL/ DCMI_VSYNC/ USART1_RX/ TIM4_CH2	93	-	-	-	-	-	-	-	-	-	-	-	-		24

Table 6. STM32 pin description versus board functions (continued)

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PB8	TIM4_CH3/ SDIO_D4/ TIM10_CH1/ DCMI_D6/ OTG_FS_SCL/ ETH_MII_TXD3/ I2C1_SCL/ CAN1_RX	95	-	-	-	-	-	-	-	-	-	-	-	-	-	19
PB9	SPI2_NSS/ I2S2_WS/ TIM4_CH4/ TIM11_CH1/ OTG_FS_SDA/ SDIO_D5/ DCMI_D7/ I2C1_SDA/ CAN1_TX	96	SDA	-	-	-	-	-	-	-	-	-	-	-	-	20
PB10	SPI2_SCK/ I2S2_CK/ I2C2_SCL/ USART3_TX/ OTG_HS_ULPI_D3/ ETH_MII_RX_ER/ OTG_HS_SCL/ TIM2_CH3	47	-	CLK	-	-	-	-	-	-	-	-	-	-	34	-
PB11	I2C2_SDA/ USART3_RX/ OTG_HS_ULPI_D4/ ETH_RMII_TX_EN/ ETH_MII_TX_EN/ OTG_HS_SDA/ TIM2_CH4	48	-	-	-	-	-	-	-	-	-	-	-	-	35	-
PB12	SPI2_NSS/ I2S2_WS/ I2C2_SMBA/ USART3_CK/ TIM1_BKIN/ CAN2_RX/ OTG_HS_ULPI_D5/ ETH_RMII_TXD0/ ETH_MII_TXD0/ OTG_HS_ID	51	-	-	-	-	-	-	-	-	-	-	-	-	36	-

Table 6. STM32 pin description versus board functions (continued)

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PB13	SPI2_SCK/ I2S2_CK/ USART3_CTS/ TIM1_CH1N/ CAN2_TX/ OTG_HS_ULPI_D6/ ETH_RMII_TXD1/ ETH_MII_TXD1/ OTG_HS_VBUS	52	-	-	-	-	-	-	-	-	-	-	-	-	37	-
PB14	SPI2_MISO/ TIM1_CH2N/ TIM12_CH1/ OTG_HS_DMUSART3 _RTS/ TIM8_CH2N/ I2S2ext_SD	53	-	-	-	-	-	-	-	-	-	-	-	-	38	-
PB15	SPI2_MOSI/ I2S2_SD/ TIM1_CH3N/ TIM8_CH3N/ TIM12_CH2/ OTG_HS_DP	54	-	-	-	-	-	-	-	-	-	-	-	-	39	-
PC0	OTG_HS_ULPI_STP/ ADC123_IN10	15	-	-	-	-	-	-	PowerOn	-	-	-	-	-	8	-
PC1	ETH_MDC/ ADC123_IN11	16	-	-	-	-	-	-	-	-	-	-	-	-	7	-
PC2	SPI2_MISO/ OTG_HS_ULPI_DIR/ TH_MII_TXD2/ I2S2ext_SD/ ADC123_IN12	17	-	-	-	-	-	-	-	-	-	-	-	-	10	-
PC3	SPI2_MOSI/ I2S2_SD/ OTG_HS_ULPI_NXT/ ETH_MII_TX_CLK/ ADC123_IN13	18	-	DOUT/AIN4x	-	-	-	-	-	-	-	-	-	-	9	-
PC4	ETH_RMII_RX_D0/ ETH_MII_RX_D0/ ADC12_IN14	33	-	-	-	-	-	-	-	-	-	-	-	-	20	-

Table 6. STM32 pin description versus board functions (continued)

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PC5	ETH_RMII_RX_D1/ ETH_MII_RX_D1/ ADC12_IN15	34	-	-	-	-	-	-	-	-	-	-	-	-	19	-
PC6	I2S2_MCK/ TIM8_CH1/ SDIO_D6/ USART6_TX/ DCMI_D0/ TIM3_CH1	63	-	-	-	-	-	-	-	-	-	-	-	-	-	47
PC7	I2S3_MCK/ TIM8_CH2/ SDIO_D7/ USART6_RX/ DCMI_D1/ TIM3_CH2	64	MCLK	-	-	-	-	-	-	-	-	-	-	-	-	48
PC8	TIM8_CH3/ SDIO_D0/ TIM3_CH3/ USART6_CK/ DCMI_D2	65	-	-	-	-	-	-	-	-	-	-	-	-	-	45
PC9	I2S_CKIN/ MCO2/ TIM8_CH4/ SDIO_D1/ I2C3_SDA/ DCMI_D3/ TIM3_CH4	66	-	-	-	-	-	-	-	-	-	-	-	-	-	46
PC10	SPI3_SCK/ I2S3_CK/ UART4_TX/ SDIO_D2/ DCMI_D8/ USART3_TX	78	SCLK	-	-	-	-	-	-	-	-	-	-	-	-	37
PC11	UART4_RX/ SPI3_MISO/ SDIO_D3/ DCMI_D4/ USART3_RX/ I2S3ext_SD	79	-	-	-	-	-	-	-	-	-	-	-	-	-	38

Table 6. STM32 pin description versus board functions (continued)

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PC12	UART5_TX/ SDIO_CK/ DCMI_D9/ SPI3_MOSI/ I2S3_SD/ USART3_CK	80	SDIN	-	-	-	-	-	-	-	-	-	-	-	-	35
PC13	RTC_AF1	7	-	-	-	-	-	-	-	-	-	-	-	-	-	12
PC14	OSC32_IN	8	-	-	-	-	-	-	-	OSC32_IN	-	-	-	-	-	9
PC15	OSC32_OUT	9	-	-	-	-	-	-	-	OSC32_OUT	-	-	-	-	-	10
PD0	FSMC_D2/ CAN1_RX	81	-	-	-	-	-	-	-	-		-	-	-	-	36
PD1	FSMC_D3/ CAN1_TX	82	-	-	-	-	-	-	-	-	-	-	-	-	-	33
PD2	TIM3_ETR/ UART5_RXSDIO_CMD / DCMI_D11	83	-	-	-	-	-	-	-	-	-	-	-	-	-	34
PD3	FSMC_CLK/ USART2_CTS	84	-	-	-	-	-	-	-	-	-	-	-	-	-	31
PD4	FSMC_NOE/ USART2_RTS	85	RESET	-	-	-	-	-	-	-	-	-	-	-	-	32
PD5	FSMC_NWE/ USART2_TX	86	-	-	-	-	RED	-	Overcurrent	-	-	-	-	-	-	29
PD6	FSMC_NWAIT/ USART2_RX	87	-	-	-	-	-	-	-	-	-	-	-	-	-	30
PD7	USART2_CK/ FSMC_NE1/ FSMC_NCE2	88	-	-	-	-	-	-	-	-	-	-	-	-	-	27

Table 6. STM32 pin description versus board functions (continued)

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PD8	FSMC_D13/ USART3_TX	55	-	-	-	-	-	-	-	-	-	-	-	-	40	-
PD9	FSMC_D14/ USART3_RX	56	-	-	-	-	-	-	-	-	-	-	-	-	41	-
PD10	FSMC_D15/ USART3_CK	57	-	-	-	-	-	-	-	-	-	-	-	-	42	-
PD11	FSMC_A16/ USART3_CTS	58	-	-	-	-	-	-	-	-	-	-	-	-	43	-
PD12	FSMC_A17/ TIM4_CH1/ USART3_RTS	59	-	-	-	-	GREEN	-	-	-	-	-	-	-	44	-
PD13	FSMC_A18/ TIM4_CH2	60	-	-	-	-	ORANGE	-	-	-	-	-	-	-	45	-
PD14	FSMC_D0/ TIM4_CH3	61	-	-	-	-	RED	-	-	-	-	-	-	-	46	-
PD15	FSMC_D1/ TIM4_CH4	62	-	-	-	-	BLUE	-	-	-	-	-	-	-	47	-
PE0	TIM4_ETR/ FSMC_NBL0/ DCMI_D2	97	-	-	INT1	-	-	-	-	-	-	-	-	-	-	17
PE1	FSMC_NBL1/ DCMI_D3	98	-	-	INT2	-	-	-	-	-	-	-	-	-	-	18
PE2	TRACECLK/ FSMC_A23/ ETH_MII_TXD3	1	-	-	-	-	-	-	-	-	-	-	-	-	-	15
PE3	TRACED0/ FSMC_A19	2	-	-	CS_I2C/SPI	-	-	-	-	-	-	-	-	-	-	16
PE4	TRACED1/ FSMC_A20/ DCMI_D4	3	-	-	-	-	-	-	-	-	-	-	-	-	-	13

Table 6. STM32 pin description versus board functions (continued)

STM32 pin			Board function													
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL or LIS3DSH	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PE5	TRACED2/ FSMC_A21/ TIM9_CH1/ DCMI_D6	4	-	-	-	-	-	-	-	-	-	-	-	-	-	14
PE6	TRACED3/ FSMC_A22/ TIM9_CH2/ DCMI_D7	5	-	-	-	-	-	-	-	-	-	-	-	-	-	11
PE7	FSMC_D4/ TIM1_ETR	38	-	-	-	-	-	-	-	-	-	-	-	-	25	-
PE8	FSMC_D5/ TIM1_CH1N	39	-	-	-	-	-	-	-	-	-	-	-	-	26	-
PE9	FSMC_D6/ TIM1_CH1	40	-	-	-	-	-	-	-	-	-	-	-	-	27	-
PE10	FSMC_D7/ TIM1_CH2N	41	-	-	-	-	-	-	-	-	-	-	-	-	28	-
PE11	FSMC_D8/ TIM1_CH2	42	-	-	-	-	-	-	-	-	-	-	-	-	29	-
PE12	FSMC_D9/ TIM1_CH3N	43	-	-	-	-	-	-	-	-	-	-	-	-	30	-
PE13	FSMC_D10/ TIM1_CH3	44	-	-	-	-	-	-	-	-	-	-	-	-	31	-
PE14	FSMC_D11/ TIM1_CH4	45	-	-	-	-	-	-	-	-	-	-	-	-	32	-
PE15	FSMC_D12/ TIM1_BKIN	46	-	-	-	-	-	-	-	-	-	-	-	-	33	-
PH0	OSC_IN	12	-	-	-	-	-	-	-	OSC_IN	-	-	-	-	-	7
PH1	OSC_OUT	13	-	-	-	-	-	-	-	OSC_OUT	-	-	-	-	-	8
-	-	-	-	-	-	-	-	-	-	-	-	5V	-	-	-	3
-	-	-	-	-	-	-	-	-	-	-	-	5V	-	-	-	4
-	-	-	-	-	-	-	-	-	-	-	-	3V	-	-	-	5