BAB III
PERANCANGAN SISTEM

3.1. Pengantar

3.1.1. Ringkasan Isi Dokumen

Di dalam dokumen ini akan dibahas mengenai rencana pengembangan robot pembersih otomatis untuk pembangkit listrik tenaga surya. Akan dipaparkan beberapa hal mengenai latar belakang, tujuan dibuatnya produk, nilai komersial dari produk, kebutuhan masyarakat. Perencanaan dari pengembangan produk yang meliputi usaha pengembangan terkait penggunaan sumber daya yang diperlukan, estimasi biaya, *timeline* kerja, dan pihak-pihak yang akan membantu ataupun mendukung pengembangan produk. Dokumen ini berisi tentang penjelasan secara ringkas terkait isi dokumen. Dimana dokumen C-300 ini digunakan untuk menjelaskan usulan proyek *capstone* yang berupa perancangan sistem yang akan diusulkan mulai dari penjabaran sistem level, desain sistem, desain *hardware*, dan desain *software*.

3.1.2. Tujuan Penulisan dan Aplikasi / Kegunaan Dokumen

Dokumen ini dibuat dengan tujuan untuk dokumentasi gagasan dan ide dasar dalam proyek pembuatan robot pembersih otomatis untuk pembangkit listrik tenaga surya. Dokumen ini akan memaparkan definisi produk, menjelaskan fungsi produk, menjabarkan spesifikasi produk, memberikan gambaran mengenai latar belakang, gagasan, konsep, nilai jual, pengembangan produk, yang akan memberikan informasi kepada pihak-pihak yang terkait dalam pengembangan pembuatan robot pembersih otomatis untuk pembangkit listrik tenaga surya.

3.2. Spesifikasi

3.2.1. Definisi, Fungsi, dan Spesifikasi

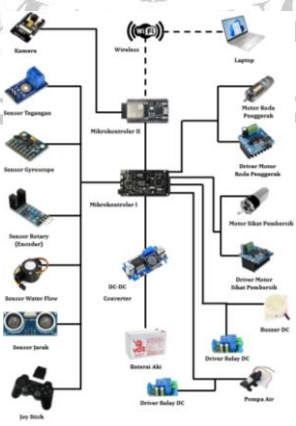
Robot pembersih otomatis untuk pembangkit listrik tenaga surya merupakan robot yang dibuat untuk membersihkan panel surya. Robot tersebut berfungsi untuk membersihkan panel surya dari debu dan kotoran dengan cara

melakukan penyemprotan dan penyikatan, hal tersebut bertujuan agar tetap menjaga kebersihan panel surya, sehingga penyerapan cahaya matahari untuk menghasilkan energi listrik yang dilakukan oleh panel surya tersebut tetap optimal. Manusia tidak perlu lagi bekerja pada ketinggian untuk sekedar membersihkan panel surya, melainkan manusia dapat menggunakan robot sebagai alternatif untuk melakukan suatu kegiatan, dalam hal ini adalah membersihkan (*maintenance*) panel surya di tempat yang tinggi. Robot ini dapat menggantikan pekerjaan yang dilakukan secara manual menjadi otomatis, sekaligus menambahkan efisiensi dalam proses bekerja.

Sudah terdapat beberapa pengembangan dari robot pembersih otomatis untuk pembangkit listrik tenaga surya, seperti robot yang menggunakan tongkat dengan sikat yang berputar secara otomatis, dan robot yang menggunakan rel sebagai media untuk melakukan pergerakannya. Perubahan spesifikasi yang terdapat pada robot ini adalah sistem pembersihan (*cleaning*) yang dilakukan oleh robot ini menggunakan metode penyemprotan dan penyikatan (*spray and brush*), dan robot dapat dikendalikan menggunakan *joy stick*, serta parameter berdasarkan kinerja robot tersebut dapat dilihat (*monitoring*) secara langsung melalui *base station* pada perangkat laptop.

3.2.2. Desain

Pada sub bab ini akan di jelaskan mengenai gambaran umum desain alat seperti gambaran interaksi alat dengan manusia (*user interface*), desain atau gambaran instalasi produk, dan perawatan produk.

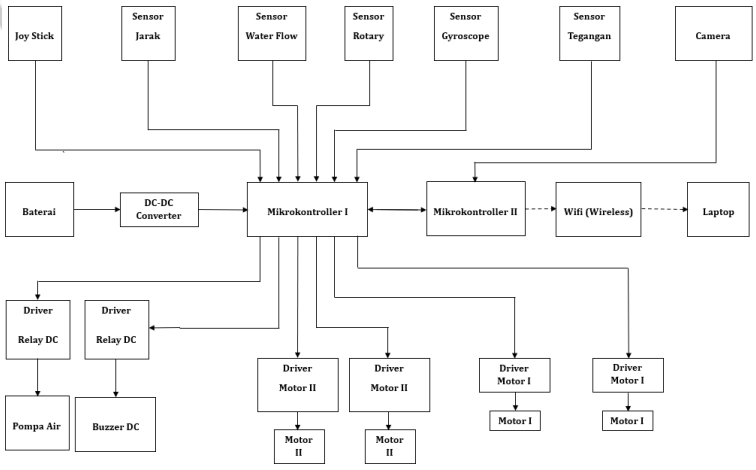


Gambar 3.1 Desain Sistem

Baterai aki digunakan sebagai suplai tegangan, tegangan tersebut diturunkan (*step down*) menggunakan DC-DC *converter*. Tegangan tersebut digunakan sebagai *input* pada mikrokontroler I dan II, dimana terdapat beberapa sensor seperi sensor tegangan (untuk mengukur tegangan), sensor *gyroscope* (untuk mengukur kemiringan derajat robot), sensor *rotary* (*encoder*) (untuk menghitung RPM pada motor), sensor *water flow* (untuk menghitung penggunaan air), sensor jarak (untuk mengukur jarak robot terhadap luas permukaan panel surya dan *ground*), dan *joy stick* (untuk sistem pengendalian robot). Terdapat *driver* motor dan motor untuk menggerakan roda robot, juga terdapat *driver* motor dan motor untuk menggerakan sikat pembersih robot, dimana kedua *driver* motor tersebut berfungsi untuk mengendalikan motor berdasarkan tegangan dan PWM. Terdapat *driver relay* DC yang berfungsi untuk mengendalikan (*on* / *off*) *buzzer* DC dan pompa air. Kemudian terdapat mikrokontroler II yang terhubung dengan kamera (untuk melihat permukaan panel surya), sekaligus mikrokontroler II tersebut berfungsi sebagai kendali untuk melakukan komunikasi kepada laptop dengan menggunakan media *WiFi* sebagai perantaranya.

3.2.2.1. Spesifikasi Fungsi dan Performansi

Pada sub bab ini menggambarkan diagram blok mengenai komponen-komponen alat, fungsi dan cara kerja alat beserta spesifikasi komponen setiap alat. Berikut penjabaran produk dengan diagram blok beserta spesifikasi masing-masing komponen :



Gambar 3.2 Diagram Blok

1. Baterai Aki

Baterai Aki digunakan sebagai sumber daya portabel untuk menyimpan energi listrik yang disalurkan ke DC-DC *converter*, dimana baterai aki berfungsi sebagai sumber utama penggerak robot. Dengan menggunakan baterai aki harapannya robot dapat beroperasi dengan lebih efisien.

Tabel 3.1 Spesifikasi Baterai Aki

Spesifikasi	
Tegangan	12V
Kapasitas	2 Ah
Terminal Size	T2
Initial Current	2,25 A

2. DC-DC *Converter*

DC-DC *Converter* digunakan sebagai pengatur sumber keluaran (*output*) dari baterai aki dan pengatur sumber masukan (*input*) ke mikrokontroler. Dengan menggunakan DC-DC *converter* harapannya suplai tegangan dari baterai aki untuk mikrokontroler dapat diturunkan (agar tidak *over voltage*).

Tabel 3.2 Spesifikasi DC-DC *Converter*

Spesifikasi	
V*in*	5 V – 30 V DC
V*out*	1,25 V – 30 V DC
I*max*	3 A – 5 A
Frekuensi	180 KHz

3. Mikrokontroler

Mikrokontroler digunakan sebagai *low level controller* yang berfungsi untuk menerima *input* dari sensor dan memberikan *output* kepada aktuator. Dengan menggunakan mikrokontroler harapannya robot dapat beroperasi dengan baik berdasarkan pertimbangan banyaknya pin atau banyaknya komponen yang digunakan.

Tabel 3.3 Spesifikasi Mikrokontroler I

Spesifikasi	
Kecepatan *Clock*	16 MHz
Memori Program (*Flash*)	256 KB (8 KB *bootloader*)
RAM	8 Kb
EEPROM	4 Kb
Tegangan Operasional	5 V
V*in*	7 V – 12 V
Pin Digital	54 Pin (15 pin digunakan sebagai PWM)
Pin Analog *Input*	16 Pin
Pin Serial (UART)	4 Pin
Pin I2C	1 Pin
Pin SPI	1 Pin

Tabel 3.4 Spesifikasi Mikrokontroler II

Spesifikasi	
V*in*	5 V
V*out*	3,3 V
Pin	V*in*, GND, GPIO, TX0, RX0, SCL, DCA, MISO, MOSI, SCK, TX2, RX2, GPIO0, GPIO2, EN, GPIO34 – GPIO39, PWM, A0, SD2, SD3

4. *WiFi* (*wireless*)

   WiFi digunakan sebagai media komunikasi antara mikrokontroler dengan *user interface* (laptop) tanpa menggunakan media kabel sebagai perantaranya. Dengan menggunakan *WiFi* harapannya robot dapat mengirimkan data secara *online* dengan waktu *real time* yang lebih cepat.

5. Laptop

   Laptop digunakan sebagai *user interface*, laptop tersebut berfungsi untuk melihat data parameter yang dihasilkan oleh sensor-sensor. Dengan menggunakan laptop harapannya dapat memudahkan (memperjelas) pada saat melihat parameter robot di *base station*.

6. *Joy Stick*

   Joy Stick digunakan sebagai sistem kendali dari setiap pergerakan yang dilakukan oleh robot, pengendali tersebut berbasis tanpa kabel (*wireless*). Dengan menggunakan *joy stick* harapannya dapat memudahkan pengendalian robot.

Tabel 3.5 Spesifikasi *Joy Stick*

Spesifikasi	
V*in*	3,3 V

7. Sensor Jarak

Sensor Jarak digunakan sebagai indikator pergerakan robot berdasarkan jarak dan luas panel surya, sehingga robot tetap berada pada area panel surya, dan dapat menghindari sisi tepi panel surya (*ground*). Dengan menggunakan sensor jarak harapannya robot dapat mendeteksi tepian (*ground*) berdasarkan luas permukaan panel surya.

Tabel 3.6 Spesifikasi Sensor Jarak

Spesifikasi	
Jarak Deteksi	20 – 150 cm
V*in*	4 – 5,5 V
I	33 – 50 mA

8. Sensor *Water Flow*

Sensor *Water Flow* digunakan sebagai pengukur aliran air melalui saluran pipa, pada robot ini sensor *water flow* digunakan untuk mengetahui konsumsi air yang digunakan untuk membersihkan panel surya. Dengan menggunakan sensor *water flow* harapannya dapat mengetahui penggunaan (debit air) yang digunakan untuk membersihkan panel surya.

Tabel 3.7 Spesifikasi Sensor *Water Flow*

Spesifikasi	
Temperatur	-25 ~ +80 °C
Tekanan	1,75 Mpa
Kelembaban	35 % - 90 % (*no frost*)
V*in*	5 – 18 V DC
I*max*	15 mA

9. Sensor *Rotary* (*Encoder*)

Sensor *Rotary* (*Encoder*) digunakan sebagai pengukur perubahan sudut dari suatu objek yang berputar, dan mengontrol putaran mesin (motor). Dengan menggunakan sensor *rotary* (*encoder*) harapannya dapat digunakan untuk mengetahui kecepatan (RPM) pada motor.

10. Sensor *Gyroscope*

Sensor *Gyroscope* digunakan sebagai pengukur sudut kemiringan dari aktivitas robot tersebut diatas panel surya. Dengan menggunakan sensor *gyroscope* harapannya dapat mengetahui kemiringan (derajat) robot berdasarkan kemiringan panel surya.

Tabel 3.8 Spesifikasi Sensor *Gyroscope*

Spesifikasi	
V*in*	3 V – 5 V DC

11. Sensor Tegangan

Sensor Tegangan digunakan sebagai pengukur tegangan listrik yang ada pada baterai aki. Dengan menggunakan sensor tegangan harapannya dapat mengetahui tegangan dari baterai aki.

Tabel 3.9 Spesifikasi Sensor Tegangan

Spesifikasi	
Mengukur Tegangan	0 – 25 V

12. Kamera

Kamera digunakan untuk melihat kondisi robot diatas panel surya, kamera berfungsi untuk melihat bagian permukaan panel surya yang kotor, maupun yang telah dibersihkan. Dengan menggunakan kamera harapannya dapat melihat permukaan panel surya dengan jelas melalui *base station*.

Tabel 3.10 Spesifikasi kamera

Spesifikasi	
V*in*	5 V
Clock Speed	240 MHz
SRAM	520 Kb
Eksternal PSRAM	4 Mb
Interfaces	UART, SPI, I2C, PWM, ADC, DAC

13. Pompa Air

Pompa Air digunakan sebagai pemompa yang akan memberikan tekanan pada air yang melewati selang, kemudian air tersebut disemprotkan melalui *nozzle*. Dengan menggunakan pompa air harapannya dapat memaksimalkan tekanan air untuk membersihkan panel surya.

Tabel 3.11 Spesifikasi Pompa Air

Spesifikasi	
Tekanan Maksimal	0,68 Mpa
P	60 – 65 Watt
V*in*	12 V – 15 V DC
I	5 A
Aliran Maksimal	4 Liter / *Min*

14. *Driver Relay* DC

Driver Relay DC digunakan sebagai sakelar pengontrol untuk menyalakan dan mematikan pompa air dan *buzzer* DC. Dengan menggunakan *driver relay* DC harapannya dapat mengendalikan *buzzer* DC dan pompa air.

Tabel 3.12 Spesifikasi *Driver Relay* DC

Spesifikasi	
V*in*	12 V – 15 V DC atau 9 V – 12 V AC

15. *Buzzer* DC

Buzzer DC digunakan sebagai bel atau indikator suara pada sistem, ketika sistem pada robot tersebut mengalami suatu kesalahan (*error*), maka *buzzer* tersebut akan berbunyi. Dengan menggunakan *buzzer* DC harapannya dapat menjadi indikator ketika sistem mengalami suatu kesalahan (*error*).

Tabel 3.13 Spesifikasi *Buzzer* DC

Spesifikasi	
V*in*	3 - 24 V DC
Diameter	3 cm

16. *Driver* Motor

Driver Motor I (roda penggerak) dan *driver* motor II (sikat pembersih) digunakan sebagai pengendali untuk menggerakkan motor, sehingga motor dapat berputar sebagai roda penggerak robot (2 *driver* motor I) dan sikat pembersih (2 *driver* motor II), *driver* motor tersebut berfungsi untuk mengontrol kecepatan, arah putaran dari motor yang digunakan pada robot tersebut. Dengan menggunakan *driver* motor harapannya dapat melakukan kontrol sekaligus menggerakan motor dengan baik.

Tabel 3.14 Spesifikasi *Driver* Motor I dan II

Spesifikasi	
V*max*	27 V DC
I*max*	43 A
Control Mode	PWM
Max Logic Input Voltage	5 V
Duty Cycle	0 – 100 %

17. Motor

Motor digunakan sebagai roda penggerak roda robot (motor I) dan penggerak sikat pembersih (motor II) robot tersebut. Dengan menggunakan motor harapannya dapat memudahkan dalam sistem roda penggerak robot dan sistem sikat pembersih robot.

Tabel 3.15 Spesifikasi Motor I

Spesifikasi	
Putaran	5000 RPM
Gear Ratio	26,9 : 1
Rotary Encoder	13 PPR
Constant Torque	16 kgfcm
Max Force Torque	25 kgfcm
Torque Stall	50 Kgfcm
V*in*	12 V – 24 V DC
Rated Current	3,5 A
P*out*	35 Watt

Tabel 3.16 Spesifikasi Motor II

Spesifikasi	
Putaran	1500 RPM
Rotary Encoder	12 PPR
Constant Torque	7,6 kgfcm
Max Force Torque	10 kgfcm
Torque Stall	26 kgfcm
Torque	20 Kgfcm
V*in*	24 V DC
Rated Current	2,5 A
P*out*	40 Watt

3.2.2.2. Spesifikasi Fisik dan Lingkungan

Robot pembersih otomatis untuk pembangkit listrik tenaga surya memiliki spesifikasi fisik dengan dimensi 60 cm x 100 cm dengan berat sekitar 20 Kg. Berdasarkan spesifikasi komponen-komponen yang digunakan, robot ini mampu bekerja pada suhu dingin sekitar -54 °C dengan batas suhu dingin maksimal sekitar -65 °C, sedangkan robot ini mampu bekerja pada suhu panas sekitar 30 °C dengan batas suhu panas maksimal sekitar 39 °C (suhu terpanas di Indonesia). Produk ini mampu bekerja dalam kondisi hujan, karena berdasarkan penempatan komponennya berada didalam *box*, sehingga robot ini kedap air. Produk ini mampu berpindah tempat untuk membersihkan panel surya yang kotor dengan cara disemprot air dan disikat.

3.2.3. Verifikasi

Pada sub bab ini menjelaskan proses dan tahapan pengujian, analisa toleransi, pengujian reliability.

Verifikasi yang dilakukan meliputi pengujian mikrokontroler, pengujian sensor-sensor, pengujian sistem kendali (*joy stick*), pengujian kamera, pengujian *driver* motor dan motor, pengujian *driver relay* DC, pengujian *buzzer* DC dan pompa air. Serta melakukan pengujian terhadap komunikasi untuk mengirim dan menerima data (parameter) kepada *base station* yang ada pada laptop. Verifikasi tersebut bertujuan untuk memastikan kinerja operasi dari masing-masing komponen yang digunakan untuk menyusun robot ini secara keseluruhan.

3.2.3.1. Prosedur Pengujian

Prosedur pengujian yang dilakukan adalah pengujian pembuatan produk robot pembersih otomatis untuk pembangkit listrik tenaga surya. Langkah-langkah yang dilakukan adalah sebagai berikut :

- Mikrokontroler I : Proses pengujian mikrokontroler dengan melakukan *uploading code* menggunakan *code editor* yang selanjutnya melakukan pengukuran pada pin *output* mikrokontroler.
- Baterai Aki dan DC-DC *Converter* : Proses pengujian baterai aki dan DC-DC *converter* untuk memastikan daya atau tegangan yang bekerja pada robot.
- Mikrokontroler II : Proses pengujian komunikasi untuk mengirim dan menerima data ke laptop yang berbasis *wireless* (*WiFi*).
- *Joy Stick* : Proses pengujian *joy stick* untuk memastikan kinerja pengendali terhadap robot secara *wireless*.
- Sensor : Proses pengujian sensor yang meliputi pengujian jarak robot terhadap *ground* panel surya, penggunaan air, *rotary* atau *encoder* dihubungkan ke pin *interrupt* mikrokontroler dan memantau kecepatan saat motor berputar, sudut kemiringan dari robot tersebut berdasarkan kemiringan panel surya, dan penggunaan daya atau tegangan yang dibutuhkan oleh robot.

- Kamera : Proses pengujian dengan melihat kamera, memastikan kamera dapat melihat bagian permukaan panel surya dalam kondisi kotor maupun bersih.
- Pompa air : Proses pengujian tekanan air yang dapat dikeluarkan oleh pompa tersebut melalui selang dan *nozzle*.
- *Driver Relay* DC : Proses pengujian sakelar pengaman dan pengujian pengontrolan tegangan.
- *Buzzer* DC : Proses pengujian bel indikator untuk memberitahu ketika terjadi suatu kesalahan (*error*) pada sistem robot.
- *Driver* Motor I dan II : Proses pengujian *driver* motor dengan menghubungkan pin PWM dan *supply*, lalu mengukur *output* tegangan menggunakan multimeter.
- Motor I dan II : Proses pengujian motor dengan menghubungkan motor ke *supply* tegangan dan melihat putaran motor.

3.2.3.2. Analisa Toleransi

Komponen utama yang paling menentukan dari keseluruhan sistem robot pembersih otomatis untuk PLTS ini adalah mikrokontroler dan *driver* motor. Karena mikrokontroler memiliki peran penting yang berfungsi sebagai pusat kontrol (pengendalian) dari keseluruhan komponen yang digunakan. Serta *driver* motor memiliki peran penting yang berfungsi sebagai pusat kontrol (pengendalian) dari motor roda penggerak robot dan motor sikat pembersih robot berdasarkan PWM dari mikrokontroler.

3.2.3.3. Pengujian Keandalan

Pengujian keandalan akan dilakukan ketika robot telah selesai dibuat. Dimana pengujian keandalan pada robot ini dilakukan dengan cara menguji kinerja robot pada saat robot bergerak (dikendalikan) diatas panel surya, melakukan penyemprotan (*spraying*) dan penyikatan (*brushing*), melihat parameter berdasarkan hasil dari sensor-sensor yang digunakan, menguji sinkronasi kamera terhadap *base station*, serta menguji koneksi saat mengirimkan dan menerima data

(menggunakan media *WiFi*) dimana data atau parameter yang dihasilkan tersebut akan ditampilkan pada *base station* melalui laptop berdasarkan *real time condition*.

3.2.4. Biaya dan Jadwal

Pada sub bab berikut menjabarkan terkait biaya komponen, dan perhitungan biaya produksi. Selain biaya juga dijabarkan jadwal pengerjaan dan tugas masing-masing anggota kelompok.

3.2.4.1. Biaya Komponen

Tabel 3.17 Biaya Komponen

Pengeluaran / Alat	Harga Satuan	Jumlah	Total
Camera ESP32	Rp. 100.000	1 Buah	Rp. 100.000
Sensor Tegangan	Rp. 10.000	1 Buah	Rp. 10.000
Sensor Gyroscope	Rp. 30.000	1 Buah	Rp. 30.000
Sensor Rotary	Rp. 15.000	1 Buah	Rp. 15.000
Sensor Water Flow	Rp. 60.000	1 Buah	Rp. 60.000
Sensor Ultrasonic	Rp. 12.000	4 Buah	Rp. 48.000
Joy Stick	Rp. 100.000	1 Buah	Rp. 100.000
ESP32	Rp. 70.000	1 Buah	Rp. 70.000
Arduino Mega 2560	Rp. 220.000	1 Buah	Rp. 220.000
DC-DC Converter 5 Volt	Rp. 80.000	1 Buah	Rp. 80.000
Baterai Aki 12 Volt	Rp. 400.000	2 Buah	Rp. 800.000
Motor DC PG42	Rp. 700.000	2 Buah	Rp. 1.400.000
Motor DC PG36	Rp. 500.000	2 Buah	Rp. 1.000.000
Buzzer DC	Rp. 10.000	1 Buah	Rp. 10.000
Driver Relay DC	Rp. 130.000	2 Buah	Rp. 260.000
Pompa Air	Rp. 100.000	1 Buah	Rp. 100.000
Selang Air Pneumatic 10mm	Rp. 10.000	5 Meter	Rp. 50.000
Steam	Rp. 10.000	10 Buah	Rp. 100.000
PCB Board	Rp. 100.000	2 Buah	Rp. 200.000
Roda (Belt)	Rp. 300.000	2 Buah	Rp. 600.000
Sikat Pembersih	Rp. 70.000	2 Buah	Rp. 140.000
Reflector	Rp. 100.000	1 Buah	Rp. 100.000
Driver Motor	Rp. 80.000	4 Buah	Rp. 320.000
Keperluan lainnya	Rp. 150.000	1 Buah	Rp. 150.000
Total Keseluruhan			**Rp. 5.963.000**

3.2.4.2. Perhitungan Biaya Produksi

Proses yang dilakukan dalam pembuatan robot ini membutuhkan modal sekitar Rp. 5.963.000, dimana modal produksi robot tersebut didasarkan pada keseluruhan komponen yang digunakan untuk menyusun robot ini. Jika dilihat dari sisi keuntungan dan penambahan jumlah produksi, robot ini membutuhkan modal sekitar Rp. 8.000.000, dimana harga tersebut didasarkan pada aspek yang terkait dengan keuntungan serta penambahan jumlah produksi robot untuk kedepannya.

Berdasarkan modal tersebut, robot ini dapat diproduksi dalam jumlah yang cukup banyak. Mengingat banyaknya industri yang membutuhkan robot tersebut. Terkait dengan biaya karyawan / jasa, proses pada pembuatan robot ini melibatkan 4 orang *engineer* dengan masing-masing gaji Rp. 3.500.000 / bulan, dimana proses pengerjaan dari robot ini membutuhkan waktu sekitar 9 bulan, sehingga total biaya dari karyawan / jasa tersebut sekitar Rp. 126.000.000.

3.2.4.3. Jadwal Pengerjaan

Tabel 3.18 Jadwal Pengerjaan

Nama Kegiatan	Bulan Pelaksanaan								Responden
	November	Desember	Januari	Februari	Maret	April	Mei	Juni	
Penentuan Konsep									Semua Anggota
Penentuan Komponen									Semua Anggota
Pengecekan Sensor									Akhmad Bhakti Prastyo
Pengecekan Mikrokontroller									Akhmad Bhakti Prastyo
Pengecekan Camera									Ahmad Faisal Hafid
Pengecekan Komunikasi Mikrokontroller									Ahmad Faisal Hafid
Pembuatan Desain Skematik Driver Motor dan Layout PCB									Alvin Fa'iz Fathurrahman
Pembuatan Desain Skematik Driver Motor dan Layout PCB									Mohammad Taufiqurrohman
Pembuatan Sistem Base Station									Ahmad Faisal Hafid
Pengecekan Kendali Produk									Akhmad Bhakti Prastyo
Pengecekan Motor									Alvin Fa'iz Fathurrahman
Pengecekan Motor									Mohammad Taufiqurrohman
Pengecekan Setiap Komponen									Semua Anggota
Melakukan Perancangan Produk Secara Keseluruhan									Semua Anggota
Melakukan Pengujian Produk									Semua Anggota

3.3. Perancangan Sistem

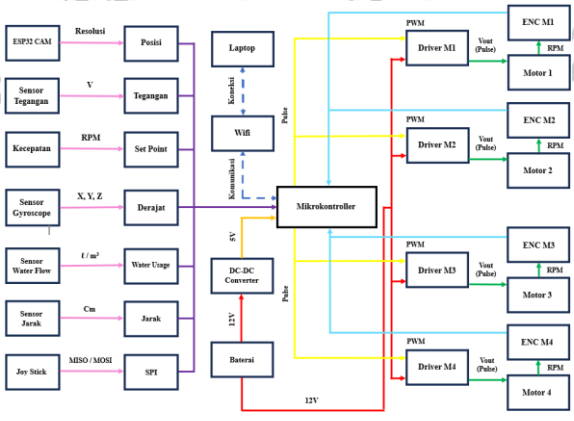
3.3.1. Penjabaran Sistem Level

Penjabaran sistem level dilakukan dengan menggunakan penjabaran sistem yang diusulkan menggunakan data *flow* diagram (DFD). Pada sub-bab ini berisi

gambar DFD dari usulan yang berisi sistem diagram, sistem level 0, dan sistem level 1.

3.3.1.1. Sistem Level 0

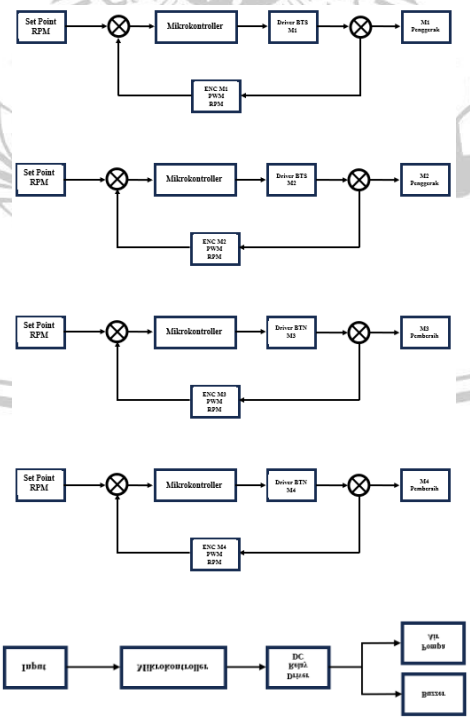
Pada sistem level 0, kamera akan mengatur parameter resolusi sebagai *vision* (berdasarkan posisi) saat melihat permukaan panel surya. Sensor tegangan akan mengatur parameter voltase yang akan digunakan untuk *driver* motor dan mikrokontroler, dimana kecepatannya didasarkan dengan mengatur parameter (*set point*) untuk RPM. Sensor *gyroscope* akan mengatur parameter dari X, Y, dan Z (derajat kemiringan). Sensor *water flow* akan mengatur parameter penggunaan air (liter dan debit). Sensor jarak akan mengatur parameter cm terhadap luas permukaan panel surya. *Joy stick* akan mengatur parameter SPI (MISO, MOSI) untuk kendali robot. Laptop dan *WiFi* digunakan sebagai koneksi (perantara) dengan mikrokontroler. Mikrokontroler akan mengontrol motor PG42 (roda penggerak) dan motor PG36 (sikat pembersih) sesuai dengan PWM, mikrokontroler akan mengirim PWM ke *driver* motor untuk mengatur besar tegangan *output* ke *driver* motor. Tegangan *output driver* motor tersebut akan menjadi *input* tegangan untuk motor PG42 dan PG36, dimana kecepatan (RPM) dari motor PG42 dan PG36 akan dibaca oleh *encoder*. *Pulse* yang dihasilkan oleh *encoder* akan dikirim ke mikrokontroler, berdasarkan *pulse* yang dikirim, akan didapatkan RPM dari motor PG42 dan PG36, jika nilai RPM kurang dari nilai *set point* yang diatur, maka *duty cycle* PWM akan ditambah, dan jika RPM lebih dari nilai *set point* yang diatur, maka *duty cycle* PWM akan dikurang.



Gambar 3.3 Sistem Level 0

3.3.1.2. Sistem Level 1

Pada sistem level 1, mikrokontroler akan mengontrol motor PG42 (roda penggerak) dan motor PG36 (sikat pembersih) sesuai dengan PWM dan *set point* RPM yang telah diatur, mikrokontroler akan mengirim PWM ke *driver* motor untuk mengatur besar tegangan *output* ke *driver* motor. Tegangan *output driver* motor tersebut akan menjadi *input* tegangan untuk motor PG42 dan motor PG36, dimana kecepatan (RPM) dari motor PG42 dan motor PG36 akan dibaca oleh *encoder*. *Pulse* yang dihasilkan oleh *encoder* akan dikirim ke mikrokontroler dan akan dilakukan perbandingan berdasarkan nilai *error* yang ada. Berdasarkan *pulse* yang dikirim oleh *encoder*, akan didapatkan nilai RPM dari motor PG42 dan motor PG36, jika nilai RPM kurang dari nilai *set point* yang diatur (*error*), maka akan dilakukan perbandingan, dan kemudian *duty cycle* PWM akan ditambah, sedangkan jika RPM lebih dari nilai *set point* yang diatur (*error*), maka akan dilakukan perbandingan, dan kemudian *duty cycle* PWM akan dikurang. Berdasarkan sistem air, *input* yang diberikan kepada mikrokontroler akan mengendalikan *driver relay* DC (*on* / *off*) dimana aktif dan non aktifnya *driver relay* DC ini akan mempengaruhi aktif dan *non* aktifnya *buzzer* DC dan pompa air tersebut.



Gambar 3.4 Sistem level 1

3.3.2. Pendahuluan Metode

Kegiatan dikerjakan oleh 4 orang mahasiswa dengan rentang waktu 9 Bulan, dan tempat pengerjaan berada di bengkel LSO Robotika UMM.

Metode yang akan digunakan adalah PID (*Proportional Integral Derivative*) untuk mengontrol (*tuning*) dari motor PG42 dan motor PG36.

3.3.2.1. PID (*Proportional Integral Derivative*)

PID adalah kontroller yang digunakan untuk menentukan tingkat presisi dari suatu pengontrolan *loop* tertutup (*closed loop*) berdasarkan *set point* dan nilai *error*.

- Kontrol *Proportional* (Kp)

Kontrol *proportional* berfungsi untuk memberikan respon sistem yang sebanding (sama nilainya) berdasarkan perbedaan antara nilai *set point* dan hasil (aktual). Kontrol ini juga berfungsi untuk mengurangi *over shoot* (*error*) pada saat awal motor berjalan (*starting*) dan untuk memberikan respon yang cepat berdasarkan perubahan sistem pada motor tersebut.

Rumus :

$$u(t) = Kp * e(t)$$

Keterangan :

u (t) = sinyal kontrol pada waktu t.

Kp = *gain* proporsional (konstanta proporsional).

e (t) = *error* pada waktu t.

Dapat dihitung sebagai :

$$e(t) = r(t) - y(t)$$

Keterangan :

r (t) = *set point* atau nilai awal (referensi).

y (t) = nilai keluaran (*output* proses) atau nilai yang diukur.

Error e (t) merupakan perbedaan nilai antara *set point* atau nilai awal (referensi) dengan nilai yang diukur (hal ini akan menunjukkan seberapa jauh perbandingan sistem dari nilai yang kita inginkan atau kita atur).

Gain proporsional (Kp) merupakan faktor pengali yang dapat menentukan reaksi pengendali terhadap nilai *error*. Nilai Kp yang tinggi akan menyebabkan reaksi yang lebih kuat terhadap nilai *error*, hal tersebut dapat menyebabkan sistem menjadi tidak stabil (jika *tuning* nilai Kp terlalu tinggi).

Contoh perhitungan :

Semisal,

r (t) = 10 (nilai *set point* atau referensi).

y (t) = 8 (nilai keluaran atau *output* yang kita ukur).

Kp = 2 (*gain* atau penguatan proporsional).

Maka,

Untuk menghitung nilai *error* adalah :

$$e(t) = r(t) - y(t)$$

$$= 10 - 8 = 2$$

Untuk menghitung nilai sinyal kontrol adalah :

$$u(t) = Kp * e(t)$$

$$= 2 * 2 = 4$$

- Kontrol Integral (Ki)

Kontrol integral berfungsi untuk mengurangi kesalahan motor berdasarkan respon waktu. Kontrol ini juga berfungsi untuk memastikan sistem motor mencapai nilai *set point* dan tetap stabil.

Rumus :

$$u(t) = Ki \int_0^t e(T)\, dT$$

Keterangan :

u (t) = sinyal kontrol pada waktu t.

Ki = *gain* integral (konstanta integral).

e (t) = *error* pada waktu t.

$\int_0^t e(T)\, dT$ = integral dari *error* waktu ke 0 sampai waktu ke t.

Error e (t) merupakan perbedaan antara nilai *set point* (referensi) dengan nilai nilai yang diukur pada waktu t. *Error* ini selalu terintegrasi sepanjang waktu (sistem berjalan) untuk memberikan sinyal kontrol.

Gain integral (Ki) merupakan faktor pengali yang dapat menentukan reaksi pengendali terhadap integral dari nilai *error*. Nilai Ki yang tinggi akan menyebabkan reaksi yang lebih cepat terhadap nilai *error*, hal tersebut dapat menyebabkan sistem menjadi tidak stabil (jika *tuning* nilai Ki terlalu tinggi).

Contoh perhitungan :

Semisal,

r (t) = 10 (nilai *set point* atau referensi).

y (t) = 8 (nilai keluaran atau *output* yang kita ukur).

Ki = 1 (*gain* atau penguatan integral).

Jika,

Error pada waktu t adalah e (t) = r (t) – y (t) = 2, dan kita mengintegrasi (integral) *error* dari waktu ke 0 sampai waktu ke t, maka kita dapat menghitung sinyal kontrol integral seperti berikut.

Untuk menghitung nilai error adalah :

e (t) = r (t) – y (t)

= 10 – 8 = 2

Untuk menghitung nilai integral dari error adalah :

Kita misalkan nilai integral dari error waktu ke 0 sampai waktu ke t adalah $\int_0^t e(T)\ dT$

Untuk menghitung nilai sinyal kontrol adalah :

u (t) = Ki $\int_0^t e(T)\ dT$

Jika,

$\int_0^t e(T)\ dT$ = 4

Maka,

u (t) = 1 * 4 = 4

- Kontrol *Derivative* (Kd)

Kontrol *derivative* berfungsi untuk merespon kesalahan motor dan mencegah *overshoot*, serta meningkatkan stabilitas sistem. Kontrol ini juga berfungsi untuk mengurangi osilasi dan meningkatkan respon dinamis pada motor.

Rumus :

$$u\ (t) = Kd\ \frac{de\ (t)}{dt}$$

Keterangan :

u (t) = sinyal kontrol pada waktu t.

Kd = *gain* derivatif (konstanta derivatif).

e (t) = *error* pada waktu t.

$\frac{de\ (t)}{dt}$ = turunan pertama dari *error* terhadap waktu.

Error e (t) merupakan perbedaan nilai antara *set point* atau nilai awal (referensi) dengan nilai yang diukur pada waktu t (kontroler derivatif akan mempertimbangkan perubahan *error*).

Gain derivatif (Kd) merupakan faktor pengali yang dapat menentukan reaksi pengendali terhadap perubahan nilai *error*. Nilai Kd yang tinggi akan menyebabkan reaksi yang lebih cepat terhadap perubahan nilai *error*, hal tersebut dapat menyebabkan sistem menjadi sensitif terhadap *noise* (jika *tuning* nilai Kd terlalu tinggi).

Contoh perhitungan :

Semisal,

r (t) = 10 (nilai *set point* atau referensi).

y (t) = 8 (nilai keluaran atau *output* yang kita ukur).

Kd = 0,5 (*gain* atau penguatan derivatif).

Jika,

Perubahan *error* pada waktu t adalah $\frac{de\ (t)}{dt}$ = 4 per detik.

Maka,

Untuk menghitung nilai *error* adalah :

e (t) = r (t) – y (t)

= 10 – 8 = 2

Untuk menghitung perubahan *error* adalah :

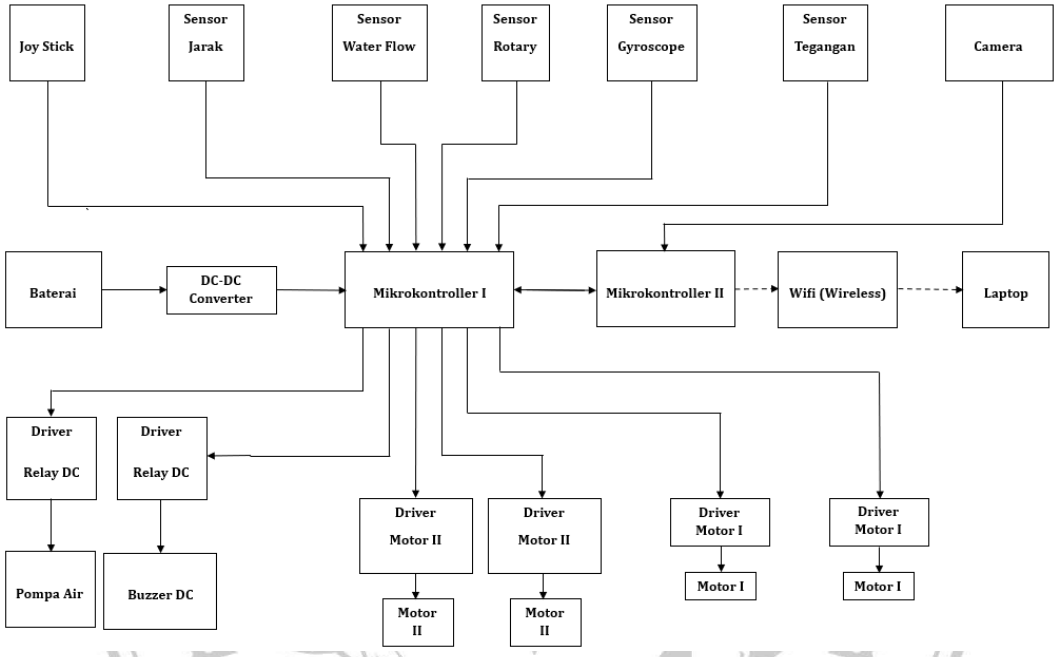
$\frac{de\ (t)}{dt}$ = 4

Untuk menghitung sinyal kontrol adalah :

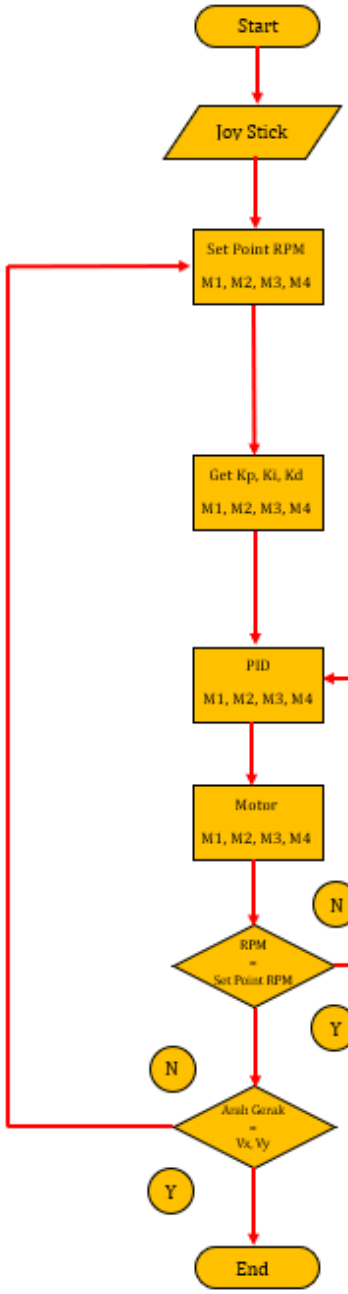
$$u\,(t) = Kd\,\frac{de\,(t)}{dt} \quad = 0{,}5 * 4 = 2$$

3.3.3. Desain Sistem

Menggambarkan proses kerja secara keseluruhan. Di dalam sub-bab ini juga terdapat diagram alir dari proses sistem yang diusulkan. Proses diagram alir harus dilengkapi dengan keterangan fungsi dari masing-masing elemen. Gambar 3.5 merupakan contoh dari proses kerja suatu usulan secara keseluruhan dan Gambar 3.6 adalah diagram alir proses sistem.



Gambar 3.5 Desain Sistem Keseluruhan



Gambar 3.6 Diagram Alir Proses Sistem

Keterangan:

1. *Joy Stick* : Berfungsi untuk mengatur kendali (pergerakan) robot.

2. Sensor Jarak : Berfungsi untuk mendeteksi jarak robot terhadap *ground* panel surya.

3. Sensor *Water Flow* : Berfungsi untuk mendeteksi penggunaan debit (liter) air.

4. Sensor *Rotary Encoder* : Berfungsi untuk membaca kecepatan putar motor (RPM) dan mengirimkan *pulse* ke mikrokontroler.

5. Sensor *Gyroscope* : Berfungsi untuk mendeteksi kemiringan robot.

6. Sensor Tegangan : Berfungsi untuk mendeteksi tegangan pada baterai aki robot.

7. Kamera : Berfungsi untuk melihat kondisi (*vision*) panel surya.

8. Baterai Aki : Berfungsi untuk suplai tegangan pada *driver* motor dan mikrokontroler.

9. DC-DC *Converter* : Berfungsi untuk menurunkan tegangan dari baterai aki sebelum masuk ke mikrokontroler.

10. Mikrokontroler : Berfungsi untuk pusat pengendalian sistem.

11. *WiFi* : Berfungsi untuk media komunikasi antara mikrokontroler dan Laptop.

12. Laptop : Berfungsi untuk melihat paramater yang dihasilkan oleh sensor.

13. *Driver Relay* DC : Berfungsi untuk mengontrol tegangan dan *on* / *off* dari pompa air dan *buzzer* DC.

14. Pompa Air : Berfungsi untuk melakukan pemompaan air untuk membersihkan panel surya.

15. *Buzzer DC* : *Buzzer* DC digunakan sebagai bel atau indikator suara pada sistem, ketika sistem pada robot tersebut mengalami suatu kesalahan (*error*), maka *buzzer* tersebut akan berbunyi.

16. *Driver* Motor : Berfungsi untuk mengubah PWM menjadi tegangan *output* untuk mengatur kecepatan putaran motor.

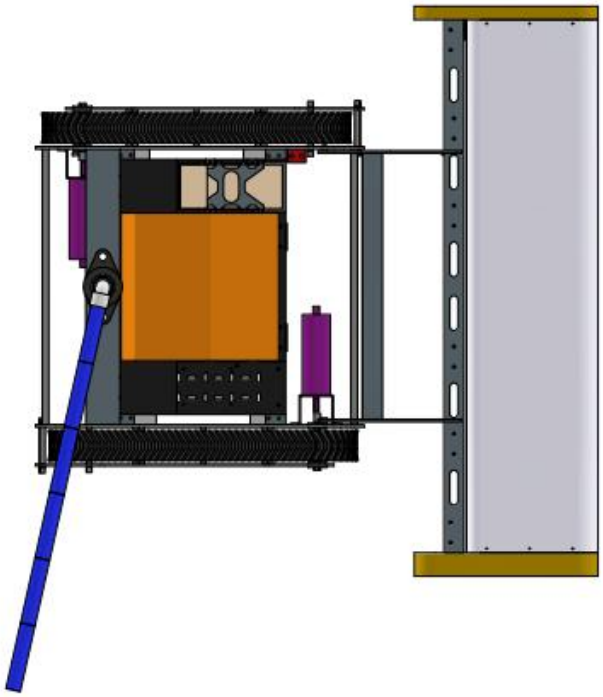
17. Motor : Berfungsi untuk menggerakan robot.

18. Kp, Ki, Kd : Berfungsi untuk melakukan pengontrolan pada motor.

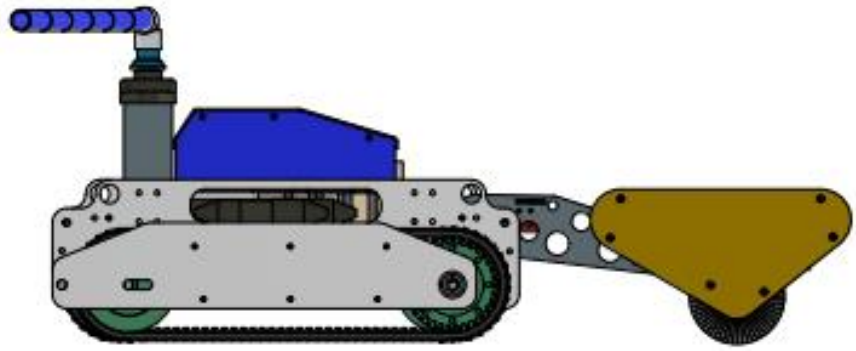
19. Vx, Vy : Berfungsi untuk mengatur arah gerak robot.

3.3.4. Desain *Hardware*

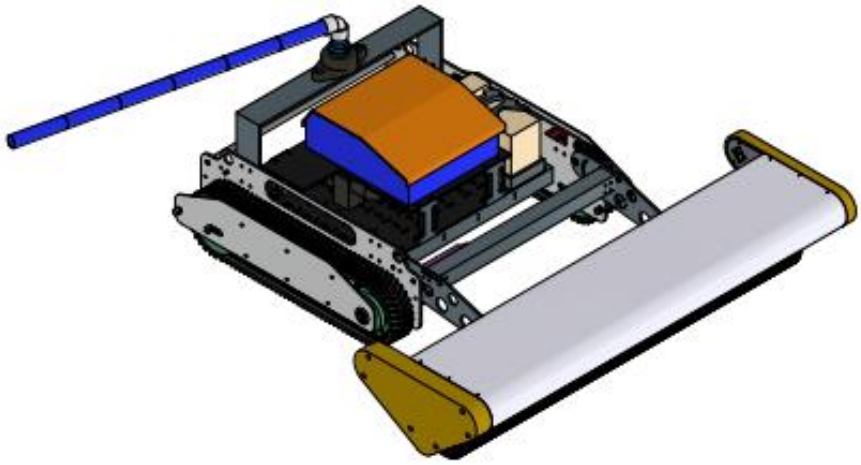
Berikut merupakan penjelasan mengenai perangkat beserta komponen yang akan digunakan dalam proyek Pengembangan Robot Pembersih Otomatis untuk Pembangkit Listrik Tenaga Surya, serta penjelasan berupa spesifikasi dari masing-masing komponen dan perangkat.



Gambar 3.7 Desain Robot Tampak Atas



Gambar 3.8 Desain Robot Tampak Samping



Gambar 3.9 Desain Robot

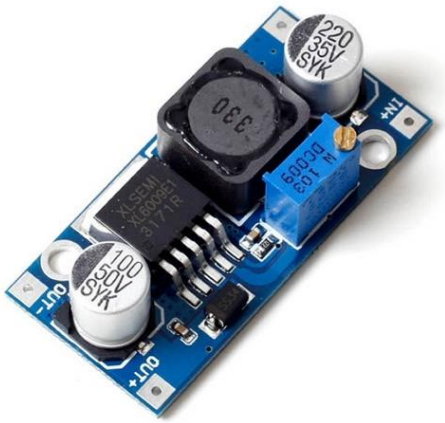
Terdapat beberapa komponen yang akan digunakan untuk merancang robot pembersih otomatis untuk pembangkit listrik tenaga surya, komponen-komponen tersebut meliputi :

3.3.4.1. Baterai Aki 12V

Gambar 3.10 Baterai Aki 12V

Baterai Aki adalah tipe baterai dengan timbal-asam yang berfungsi untuk menyediakan listrik. Pada baterai aki terdapat suatu proses reaksi kimia antara timbal dan asam sulfat untuk menghasilkan listrik. Kelebihan dari baterai aki adalah harga yang terjangkau, mampu menyediakan arus besar untuk *starting*, mudah ditemukan, daya tahan lama, memiliki stabilitas tegangan.

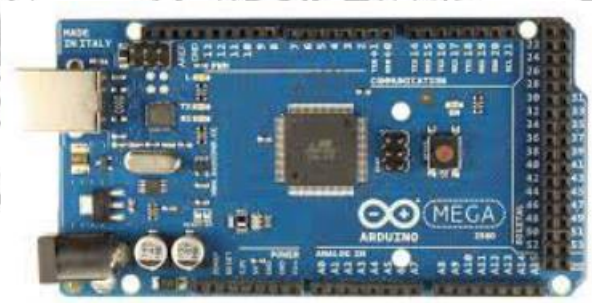
3.3.4.2. DC-DC *Converter* XL6009 *Step Down* 12V – 5V



Gambar 3.11 DC-DC *Converter* XL6009 *Step Down* 12V – 5V

DC-DC *Converter* digunakan sebagai pengatur sumber keluaran daya listrik dari baterai aki yang telah diturunkan dari 12V menjadi 5V, dan kemudian disalurkan ke mikrokontroler.

3.3.4.3. Mikrokontroler Arduino Mega 2560



Gambar 3.12 Mikrokontroler Arduino Mega 2560

Mikrokontroler digunakan sebagai *low level controller* yang digunakan untuk menerima *input* dari sensor dan memberikan *output* kepada aktuator.

3.3.4.4. Mikrokontroler ESP32CAM

Gambar 3.13 Mikrokontroler ESP32CAM

Mikrokontroler digunakan sebagai *low level controller* yang digunakan untuk komunikasi dengan laptop yang menggunakan media perantara *WiFi*.

3.3.4.5. Laptop

Gambar 3.14 Laptop

Laptop digunakan sebagai *user interface*, dari laptop tersebut kita dapat melihat data parameter yang dihasilkan oleh sensor-sensor.

3.3.4.6. *Joy Stick Play Station* 2 *Wireless*

Gambar 3.15 *Joy Stick Play Station* 2 *Wireless*

Joy stick digunakan sebagai kendali dari setiap pergerakan yang dilakukan oleh robot tersebut, pengendali yang digunakan berbasis *wireless*.

3.3.4.7. Sensor Jarak *Ultrasonic* HCSR04

Gambar 3.16 Sensor Jarak *Ultrasonic* HCSR04

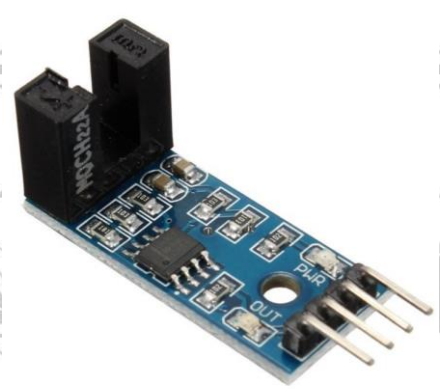
Sensor jarak digunakan sebagai indikasi pergerakan robot terhadap jarak dan luasan panel surya, sehingga robot tetap berada pada area panel surya, dan menghindari sisi tepi (*ground*).

3.3.4.8. Sensor *Water Flow* 3/4 0-60L/Menit

Gambar 3.17 Sensor *Water Flow* 3/4 0-60L/Menit

Sensor *water flow* digunakan sebagai pengukur aliran air melalui saluran pipa, pada robot ini sensor *water flow* digunakan untuk mengetahui konsumsi air yang digunakan untuk membersihkan panel surya.

3.3.4.9. Sensor *Rotary* (*Encoder*)



Gambar 3.18 Sensor *Rotary* (*Encoder*)

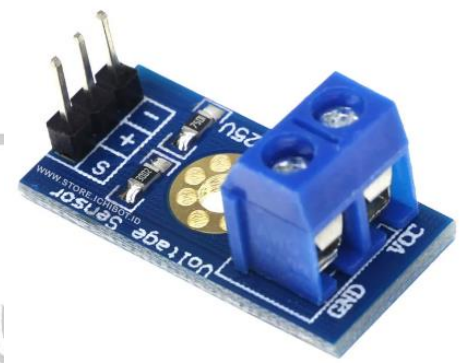
Sensor *rotary* (*encoder*) digunakan sebagai pengukur perubahan sudut dari suatu objek yang berputar, dan mengontrol putaran mesin (motor).

3.3.4.10. Sensor *Gyroscope* MPU 6050

Gambar 3.19 Sensor *Gyroscope* MPU 6050

Sensor *gyroscope* digunakan sebagai pengukur sudut kemiringan dari aktivitas robot tersebut diatas panel surya.

3.3.4.11. Sensor Tegangan DC 0-25V



Gambar 3.20 Sensor Tegangan DC 0-25V

Sensor Tegangan digunakan sebagai pengukur tegangan listrik yang ada pada baterai aki.

3.3.4.12. Kamera ESP32

Gambar 3.21 Kamera ESP32

Kamera digunakan untuk melihat kondisi robot diatas panel surya, Kamera berfungsi untuk melihat bagian permukaan panel surya yang kotor, maupun yang telah dibersihkan.

3.3.4.13. Pompa Air DC 12V

Gambar 3.22 Pompa Air DC 12V

Pompa air digunakan sebagai pemompa yang akan memberikan tekanan pada air yang melewati selang, kemudian air tersebut disemprotkan melalui *nozzle*.

3.3.4.14. *Driver Relay* DC 5V

Gambar 3.23 *Driver Relay* DC 5V

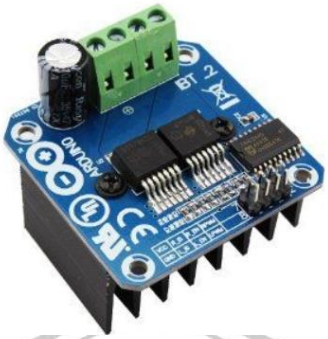
Driver relay DC digunakan sebagai sakelar pengontrol untuk menyalakan dan mematikan pompa air dan *buzzer* DC.

3.3.4.15. *Buzzer* DC 12V

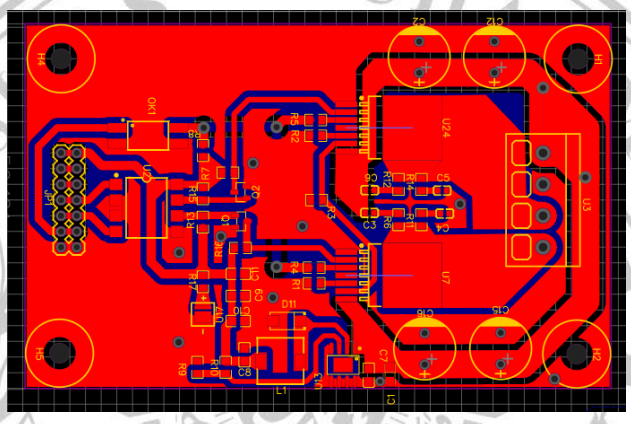
Gambar 3.24 *Buzzer* DC 12V

Buzzer DC digunakan sebagai bel atau indikator suara pada sistem, ketika sistem pada robot tersebut mengalami suatu kesalahan (*error*), maka *buzzer* tersebut akan berbunyi.

3.3.4.16. *Driver* Motor BTS 7960



Gambar 3.25 *Driver* Motor BTS 7960

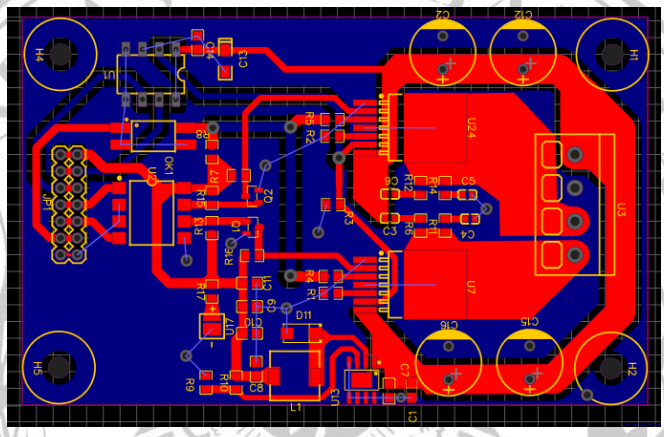


Gambar 3.26 Skematik *Driver* Motor BTS 7960

Driver motor BTS 7960 digunakan sebagai *driver* motor untuk motor yang akan menggerakan sikat pembersih robot.

3.3.4.17. *Driver* Motor BTN 7970

Gambar 3.27 *Driver* Motor BTN 7970



Gambar 3.28 Skematik *Driver* Motor BTN 7970

Driver motor BTN 7970 digunakan sebagai *driver* motor untuk motor yang akan menggerakan roda penggerak robot.

3.3.4.18. Motor DC PG42

Gambar 3.29 Motor DC PG 42

Motor DC PG42 adalah motor yang akan digunakan sebagai penggerak dari roda robot.

3.3.4.19. Motor DC PG36

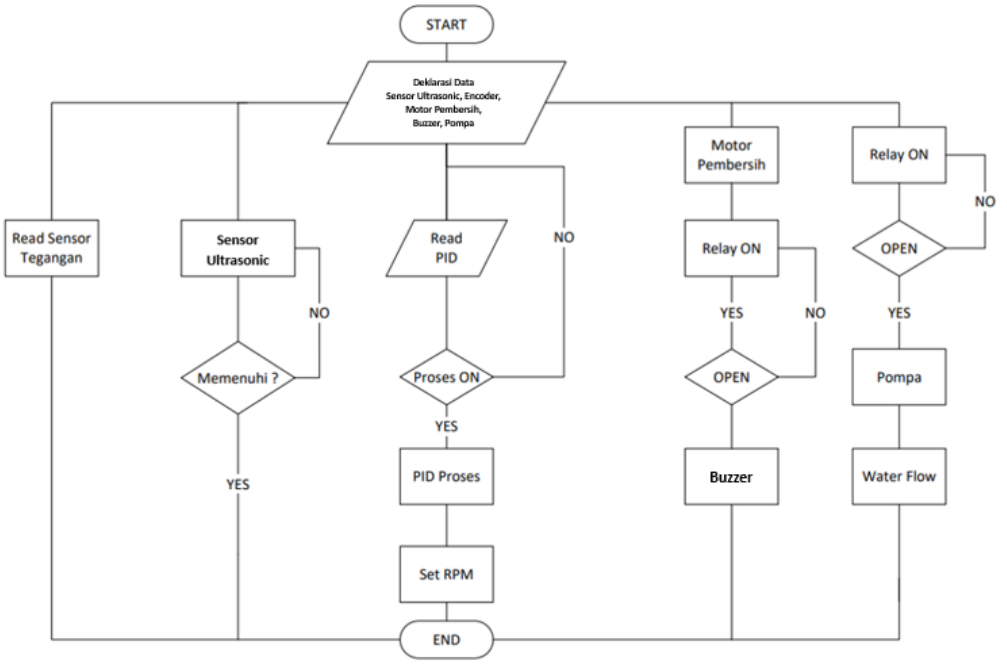
Gambar 3.30 Motor DC PG36

Motor DC PG36 adalah motor yang akan digunakan sebagai penggerak dari sikat pembersih robot.

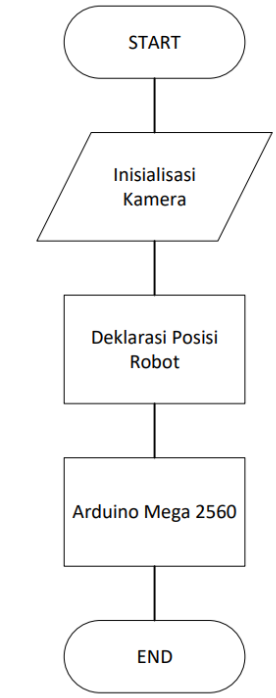
3.3.5. Desain *Software*

Untuk merancang perintah yang di unggah atau *upload* ke mikrokontroler, akan digunakan aplikasi *Arduino*. Aplikasi tersebut menggunakan bahasa C++ sebagai kode programnya. *Source code* yang dibuat mencakup beberapa bagian, yaitu :

1. Pembacaan RPM
2. Pembacaan PWM
3. Kontrol PID



Gambar 3.31 *Flowchart Software* Sistem



Gambar 3.32 *Flowchart* ESP32CAM

3.3.5.1. *Coding Arduino Mega* (Keseluruhan)

```
#include <PS2X_lib.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
#include "Motor.h"
/*--- Mendefinisikan Pin sensor UltraSonic -----*/
#define Sensor1 54
#define Sensor2 55
#define Sensor3 56
#define Sensor4 57
/*--- Mendefinisikan Pin sensor UltraSonic -----*/

/*--- Mendefinisikan PWM Motor -----*/
#define PWM_M1A 10
#define PWM_M1B 7

#define PWM_M2A 11
#define PWM_M2B 6

#define PWM_M3A 12
#define PWM_M3B 5

#define PWM_M4A 13
#define PWM_M4B 4
/*--- Mendefinisikan PWM Motor -----*/

/*--- Mendefinisikan Relay -----*/
#define Relay1 9
#define Relay2 8
/*--- Mendefinisikan Relay -----*/

/*--- Mendefinisikan Flow -----*/
#define Flow 2
```

```
/*--- Mendefinisikan Flow -----*/

PS2X ps2x;    //membuat object dari class library stick ps2
MPU6050 mpu;  //membuat object dari class library gyroscope

//variabel yang menyimpan pin sensor dan pwm
uint8_t pin_sensor[4] = { Sensor1, Sensor2, Sensor3, Sensor4 };  //menyimpan pin Sensor
kedalam array dengan jumlah index 4

int analogY;  //variabel yang menyimpan nilai Control Stick PS2
int analogX;  //variabel yang menyimpan nilai Control Stick PS2


// The hall-effect flow sensor outputs approximately 4.5 pulses per second per
// litre/minute of flow.
float calibrationFactor = 4.5;  //kalibrasi waterflow

volatile float pulseCount;  //menghitung putaran sensor waterflow

float flowRate;        //menyimpan nilai besarnya kecepatan air
long flowMilliLitres;   //menyimpan seberapa banyak air yang mengalir
long totalMilliLitres;  //total air yang sudah mengalir

unsigned long oldTime;  //menyimpan waktu terakhir


/*---------------------------------- Konfigurasi Sensor Gyro ----------------------------------
---*/
bool dmpReady = false;   // set true if DMP init was successful
uint8_t mpuIntStatus;    // holds actual interrupt status byte from MPU
uint8_t devStatus;       // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;     // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;      // count of all bytes currently in FIFO
uint8_t fifoBuffer[64];  // FIFO storage buffer
unsigned char count;
int Distance[4];
unsigned long Duration[4];
// orientation/motion vars
```

```
Quaternion q;         // [w, x, y, z]         quaternion container
VectorInt16 aa;       // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;   // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;  // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;  // [x, y, z]            gravity vector
float euler[3];       // [psi, theta, phi]    Euler angle container
float ypr[3];         // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector
uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, '\r', '\n' };
volatile bool mpuInterrupt = false;

void dmpDataReady() {
  mpuInterrupt = true;
}
/*---------------------------------- Konfigurasi Sensor Gyro ----------------------------------
---*/

float yaw = 0;    //variabel yang menyimpan Nilai Gyroskop Sumbu Z
float pitch = 0;  //variabel yang menyimpan Nilai Gyroskop Sumbu Y
float roll = 0;   //variabel yang menyimpan Nilai Gyroskop Sumbu X

float yaw_ = 0;    //variabel yang menyimpan Nilai Gyroskop Sumbu Z Pada Awal Program
Di jalankan/Nilai Awal
float pitch_ = 0;  //variabel yang menyimpan Nilai Gyroskop Sumbu Y Pada Awal Program
Di jalankan/Nilai Awal
float roll_ = 0;   //variabel yang menyimpan Nilai Gyroskop Sumbu X Pada Awal Program Di
jalankan/Nilai AWal
float volt = 0;    // Variabel yang menyimpan Nilai Voltase Batrey
bool pompa = false;
uint8_t control = 0;
bool sikat = false;
bool basestation = false;
uint8_t speed = 0;
//Variabel Yang Menyimpan Data sensor dan lainnya, kemudian akan dikirim ke BaseStation
unsigned char transmitBuffer[13] = { 0 };
unsigned char receiveBuffer[7] = { 0 };
//Variabel Yang Menyimpan Data sensor dan lainnya, kemudian akan dikirim ke BaseStation
void setup() {
```

```
  Serial2.begin(115200);  //Konfigurasi Komunikasi Serial RX TX Ke ESP32 dengan baudrate
1jt

  /*---- Inisialisasi GyroScope -------*/
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  Wire.begin();
  Wire.setClock(400000);  // 400kHz I2C clock
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif
  mpu.initialize();
  devStatus = mpu.dmpInitialize();
  mpu.setXGyroOffset(70);      //Nilai Offset Didapatkan Dari hasil Kalibrasi
  mpu.setYGyroOffset(11);      //Nilai Offset Didapatkan Dari hasil Kalibrasi
  mpu.setZGyroOffset(-37);     //Nilai Offset Didapatkan Dari hasil Kalibrasi
  mpu.setXAccelOffset(-3329);  //Nilai Offset Didapatkan Dari hasil Kalibrasi
  mpu.setYAccelOffset(556);    //Nilai Offset Didapatkan Dari hasil Kalibrasi
  mpu.setZAccelOffset(1523);   //Nilai Offset Didapatkan Dari hasil Kalibrasi
  if (devStatus == 0) {
    mpu.CalibrateAccel(6);
    mpu.CalibrateGyro(6);
    mpu.PrintActiveOffsets();
    mpu.setDMPEnabled(true);
    mpuIntStatus = mpu.getIntStatus();
    dmpReady = true;
    packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
  }
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) {  // Get the Latest packet
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    yaw_ = ypr[0] * 180 / M_PI;    //sudut kemiringan arah hadap
    pitch_ = ypr[1] * 180 / M_PI;  //sudut kemiringan depan belakang
    roll_ = ypr[2] * 180 / M_PI;   //sudut kemiringan kiri kanan
  }
  /*-------------- Inisialisasi GyroScope -----------------*/
```

```
  delay(100);
  /*-------------- Inisialisasi Pin Sensor Dan PWM ----------------*/
  for (short i = 0; i < 4; i++) {  //looping for
    pinMode(pin_sensor[i], OUTPUT);
  }
  delay(100);
  InitMotor();
  /*-------------- Inisialisasi Pin Sensor Dan PWM ----------------*/
  Serial.begin(115200);
  delay(100);
  /*-------------- Inisialisasi Stick PS2----------------*/
  ps2x.config_gamepad(52, 51, 53, 50, false, false);  //GamePad(clock, MOSI, CS, MISO,
Pressures?, Rumble?) //KOMUNIKASI SPI
  delay(100);
  /*-------------- Inisialisasi Stick PS2----------------*/
  pinMode(Relay1, OUTPUT);
  pinMode(Relay2, OUTPUT);
  digitalWrite(Relay1, HIGH);
  digitalWrite(Relay2, HIGH);

  pinMode(Flow, INPUT_PULLUP);

  pulseCount = 0;        //menyimpan nilai awal dari putaran sensor waterflow
  flowRate = 0;          //menyimpan nilai awal dari besarnya kecepatan air
  flowMilliLitres = 0;   //menyimpan nilai awal dari seberapa banyak air yang mengalir
  totalMilliLitres = 0;  //menyimpan nilai awal dari total air yang sudah mengalir
  oldTime = 0;           //

  // The Hall-effect sensor is connected to pin 2 which uses interrupt 0.
  // Configured to trigger on a FALLING state change (transition from HIGH
  // state to LOW state)
  attachInterrupt(digitalPinToInterrupt(Flow), pulseCounter, RISING);
}
bool Pompa_State = false;
void loop() {
  SendToEsp32();
  ReceiveFromEsp32();
```

```
  if (basestation) {
    //Serial.println("Control From BaseStation");
    ControlBaseStation();
    digitalWrite(Relay1, !pompa);
    digitalWrite(Relay2, !pompa);

  } else {
    //GetJoyStick();
    //Serial.println("Control From Remote");
    // digitalWrite(Relay1, false);
    // digitalWrite(Relay2, false);
  }

  getSensor();
  GetGyro();
  GetWaterFlow();
}
void serialEvent2() {
  //Serial2.readBytes(receiveBuffer, 7);
  //ReceiveFromEsp32();
}
unsigned long lastTime = 0;
void ControlBaseStation(void) {
  int x = 0, y = 0;
  unsigned long currentTime = millis();
  float deltaTime = currentTime - lastTime;
  if (control == 1 || control == 2) {
    if (control == 1) {
      //Serial.println("Maju");
      //drivePID(0, 10, speed);
      y = 10;
    } else if (control == 2) {
      //Serial.println("Mundur");
      y = -10;
      //drivePID(0, -10, speed);
    }
  } else if (control == 3 || control == 4) {
```

```
    if (control == 3) {
      x = -10;
      //drivePID(-10, 0, speed);
    } else if (control == 4) {
      x = 10;
      //drivePID(10, 0, speed);
    }
  } else {
    x = 0;
    y = 0;
    //drivePID(0, 0, 0);
    //digitalWrite(Relay2, true);
  }
  if (deltaTime > 10) {
    drivePID(x, y, speed, deltaTime);
    lastTime = currentTime;
  }
}
bool Solenoid_State = false;
void GetJoyStick(void) {
  ps2x.read_gamepad();  //Membaca Data Dari Stick Ps2
  unsigned long currentTime = millis();
  float deltaTime = currentTime - lastTime;
  analogY = ps2x.Analog(PSS_RY) - 128;          //mengambil nilai dari analog Kanan dengan
Sumbu Y
  analogX = ps2x.Analog(PSS_LX) - 128;          //mengambil nilai dari analog Kiri dengan
Sumbu X
  analogY = map(-analogY, -128, 128, -180, 180);  //Normalilasasi Nilai Joystick
  analogX = map(-analogX, -128, 128, -180, 180);  //Normalilasasi Nilai Joystick
  // //Konfigurasi Pergerakan Roda
  if (deltaTime > 10) {
    if (analogY > 100 || analogY < -100) {
      drivePID(0, analogY, 150,deltaTime);
    } else if (analogX > 100 || analogX < -100,deltaTime) {
      drivePID(-analogX, 0, 150,deltaTime);
    } else {
      drivePID(0, 0, 0,deltaTime);
```

```
    }
    lastTime = currentTime;
  }
  if (ps2x.ButtonPressed(PSB_CROSS) && !Solenoid_State) {
    //Solenoid_State = true;
  } else if (ps2x.ButtonPressed(PSB_CROSS) && Solenoid_State) {
    //Solenoid_State = false;
  }

  if (ps2x.ButtonPressed(PSB_CIRCLE) && !Pompa_State) {
    //Pompa_State = true;
  } else if (ps2x.ButtonPressed(PSB_CIRCLE) && Pompa_State) {
    //Pompa_State = false;
  }
}
void getSensor(void) {
  for (short i = 0; i < 4; i++) {
    pinMode(pin_sensor[i], OUTPUT);
    digitalWrite(pin_sensor[i], LOW);
    delayMicroseconds(2);
    digitalWrite(pin_sensor[i], HIGH);
    delayMicroseconds(5);
    digitalWrite(pin_sensor[i], LOW);
    delayMicroseconds(2);
    pinMode(pin_sensor[i], INPUT);
    Duration[i] = pulseIn(pin_sensor[i], HIGH);
    Distance[i] = Duration[i] / 29 / 2;
    // delay(50);
  }
}
void GetGyro(void) {
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) {
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    yaw = ypr[0] * 180 / M_PI;
    pitch = ypr[1] * 180 / M_PI;
```

```
    roll = ypr[2] * 180 / M_PI;
    yaw = yaw - yaw_;
    pitch = pitch - pitch_;
    roll = roll - roll_;
    if (yaw < 0) {
      yaw += 360;
    }
    if (pitch < 0) {
      pitch += 360;
    }
    if (roll < 0) {
      roll += 360;
    }
  }
}

const float R1Divide = 40000.0;
const float R2Divide = 7500.0;
int adc = 0;
void SendToEsp32(void) {
  adc = analogRead(A4);
  volt = (adc * 4.7) / 1024.0;  //1024 = 2^10 BIT ADC ARDUINO MEGA
  volt = volt * (R1Divide + R2Divide) / R2Divide;
  // /Serial.println(volt);
  // pitch *= 10.0;
  // roll *= 10.0;
  volt *= 10.0;
  //totalMilliLitres *= 10.0;
  transmitBuffer[0] = 255;
  transmitBuffer[1] = (int(yaw * 10.0) & 0xFF00) >> 8;
  transmitBuffer[2] = (int(yaw * 10.0) & 0x00FF);
  transmitBuffer[3] = (int(pitch * 10.0) & 0xFF00) >> 8;
  transmitBuffer[4] = (int(pitch * 10.0) & 0x00FF);
  transmitBuffer[5] = (int(roll * 10.0));
  transmitBuffer[6] = (int(totalMilliLitres / 1000));
  transmitBuffer[7] = int(volt);
```

```
  transmitBuffer[8] = int(Distance[0]);
  transmitBuffer[9] = int(Distance[1]);
  transmitBuffer[10] = int(Distance[2]);
  transmitBuffer[11] = int(Distance[3]);
  transmitBuffer[12] = ~(transmitBuffer[0] + transmitBuffer[1] + transmitBuffer[2] +
transmitBuffer[3] + transmitBuffer[4] + transmitBuffer[5] + transmitBuffer[6] +
transmitBuffer[7] + transmitBuffer[8] + transmitBuffer[9] + transmitBuffer[10] +
transmitBuffer[11]);  //+ transmitBuffer[12] + transmitBuffer[13] + transmitBuffer[14] +
transmitBuffer[15] + transmitBuffer[16]);
  Serial2.write(transmitBuffer, 13);
}
void ReceiveFromEsp32(void) {
  if (Serial2.available()) {
    Serial2.readBytes(receiveBuffer, 7);
    if (receiveBuffer[0] == 255) {
      uint8_t checksumesp32 = ~(receiveBuffer[0] + receiveBuffer[1] + receiveBuffer[2] +
receiveBuffer[3] + receiveBuffer[4] + receiveBuffer[5]);
      if (checksumesp32 == receiveBuffer[6]) {
        control = receiveBuffer[1];
        pompa = receiveBuffer[2];
        sikat = receiveBuffer[3];
        basestation = receiveBuffer[4];
        speed = receiveBuffer[5];
        //Serial.println("Data benar");
      }
    }
  }
}

void GetWaterFlow(void) {
  if ((millis() - oldTime) > 1000)  // Only process counters once per second
  {
    if (true) {
      // Disable the interrupt while calculating flow rate and sending the value to
      // the host
      //detachInterrupt(digitalPinToInterrupt(Flow));
```

```
// Because this loop may not complete in exactly 1 second intervals we calculate
// the number of milliseconds that have passed since the last execution and use
// that to scale the output. We also apply the calibrationFactor to scale the output
// based on the number of pulses per second per units of measure (litres/minute in
// this case) coming from the sensor.
flowRate = ((1000.0 / (millis() - oldTime)) * pulseCount) / calibrationFactor;
pulseCount = 0;
// Note the time this processing pass was executed. Note that because we've
// disabled interrupts the millis() function won't actually be incrementing right
// at this point, but it will still return the value it was set to just before
// interrupts went away.


// Divide the flow rate in litres/minute by 60 to determine how many litres have
// passed through the sensor in this 1 second interval, then multiply by 1000 to
// convert to millilitres.
flowMilliLitres = (flowRate / 60) * 1000;

// Add the millilitres passed in this second to the cumulative total
totalMilliLitres += flowMilliLitres;

unsigned int frac;

// Print the flow rate for this second in litres / minute
Serial.print("Flow rate: ");
Serial.print(int(flowMilliLitres));  // Print the integer part of the variable
Serial.print("L/min");
Serial.print("\t");  // Print tab space

// Print the cumulative total of litres flowed since starting
Serial.print("Output Liquid Quantity: ");
Serial.print(totalMilliLitres);
Serial.println("mL");
Serial.print("\t");  // Print tab space
Serial.print(totalMilliLitres / 1000);
Serial.print("L");
```

```
      // Reset the pulse counter so we can start incrementing again


      // Enable the interrupt again now that we've finished sending output
      //attachInterrupt(digitalPinToInterrupt(Flow), pulseCounter, FALLING);
    }
    oldTime = millis();
  }
}
/*
Insterrupt Service Routine
 */
void pulseCounter() {
  // Increment the pulse counter
  pulseCount++;
}
```

```
#include <Arduino.h>

#define IR1 A0
#define IR2 A1
#define IR3 A2
#define IR4 A3
#define Voltage A4

float ADC_Sensor[5];
float Voltage_ADC[5];

float R1 = 30000.0;
float R2 = 7500.0;

const uint8_t PIN_ADC[5] = { IR1, IR2, IR3, IR4, Voltage };

void GetADC(void) {
  ADC_Sensor[4] = analogRead(PIN_ADC[4]);
  Voltage_ADC[4] = (ADC_Sensor[4] * 5.0) / 1024.0;  //1024 = 2^10 BIT ADC ARDUINO
MEGA
```

```
  Voltage_ADC[4] = Voltage_ADC[4] * (R1 + R2) / R2;
}
```

```
#ifndef Motor_h
#define Motor_h
void SerialPrintEncoder(void);
void InitMotor(void);
void drivePID(int x, int y, int speed,float deltaT);
#endif
```

```
#include "HardwareSerial.h"
#include "Arduino.h"
#include <util/atomic.h>
#define ENCM1A 19
#define ENCM2A 18

#define ENCM1B 15
#define ENCM2B 14

#define PWM_M1A 10
#define PWM_M1B 7

#define PWM_M2A 11
#define PWM_M2B 6

#define PWM_M3A 12
#define PWM_M3B 5

#define PWM_M4A 13
#define PWM_M4B 4

#define NUM_ENC_CH 2
#define PWM_CH 8
void WriteMotorSpeed(short motor, uint8_t speed);

void readEncoderM1();
```

```
void readEncoderM2();
volatile long tickEncoder[4];
float _SP[4];
uint8_t pin_PWM[8] = { PWM_M1A, PWM_M1B, PWM_M2A, PWM_M2B, PWM_M3A,
PWM_M3B, PWM_M4A, PWM_M4B };  //menyimpan pin PWM kedalam array dengan
jumlah index 8

void InitMotor(void) {
  for (short i = 0; i < NUM_ENC_CH; i++) {
    pinMode(ENCM1A, INPUT_PULLUP);
    pinMode(ENCM2A, INPUT_PULLUP);
    pinMode(ENCM1B, INPUT_PULLUP);
    pinMode(ENCM2B, INPUT_PULLUP);
    tickEncoder[i] = 0;
  }
  delay(100);
  for (short i = 0; i < 8; i++) {
    pinMode(pin_PWM[i], OUTPUT);
    delay(100);
    analogWrite(pin_PWM[i], 0);
    delay(100);
  }
  attachInterrupt(digitalPinToInterrupt(ENCM1A), readEncoderM1, RISING);
  attachInterrupt(digitalPinToInterrupt(ENCM2A), readEncoderM2, RISING);
  // WriteMotorSpeed(5, 255);//4-5
  // WriteMotorSpeed(6, 255);//6-7
}

void readEncoderM1() {
  bool chB = digitalRead(ENCM1B);
  if (chB) {
    tickEncoder[0]++;
  } else {
    tickEncoder[0]--;
  }
}
```

```
void readEncoderM2() {
  bool chB = digitalRead(ENCM2B);
  if (chB) {
    tickEncoder[1]++;
  } else {
    tickEncoder[1]--;
  }
}

void SerialPrintEncoder(void) {
  Serial.print("Encoder 1 : ");
  Serial.println(tickEncoder[0]);
  Serial.print("Encoder 2 : ");
  Serial.println(tickEncoder[1]);
  Serial.print("Encoder 3 : ");
  Serial.println(tickEncoder[2]);
  Serial.print("Encoder 4 : ");
  Serial.println(tickEncoder[3]);
}

void WriteMotorSpeed(short motor, uint8_t speed) {
  analogWrite(pin_PWM[motor], speed);
}
// set KP, KI, KD value
const float KP = 0.5;
const float KI = 0.5;
const float KD = 2;

// set encoder PPR
float encoder_PPR = 7;
// set encoder PPR
float gearRatio = 139;
// set RPM Maximum
float maxRpm = 186;
const int TimeSampling = 10;  //ms

unsigned long lastTimeM1 = 0;
```

```
/*====================== PID MOTOR Kanan
============================*/
void pidComputeM1(float dt) {
  // put your main code here, to run repeatedly:
  unsigned long currentTime = millis();
  float lasterror;
  float deltaTime = currentTime - lastTimeM1;
  float sumerror;
  // if (deltaTime >= TimeSampling) {
  float freqSignal = (float)tickEncoder[0] / (dt / 1.0e3);  //(deltaTime/1.0e3) --> convert ms to
s
  tickEncoder[0] = 0;
  float rpm = freqSignal * 60.0 / (gearRatio * encoder_PPR);  //224.4 = p*N; p=step-up gear
ratio; N=pulses per revolution
  float error = (_SP[0] - rpm);
  sumerror += error;
  int P = (KP * error);
  int I = (KI * sumerror);
  int D = KD * (error - lasterror) / dt;
  int PID = P + I + D;
  lasterror = error;
  // Serial.print("RPM 1 :");
  // Serial.println(rpm);
  PID = constrain(PID, -255, 255);
  // Serial.print("PID 1 :");
  // Serial.println(PID);
  if (PID > 0) {
    WriteMotorSpeed(0, abs(int(PID)));
    WriteMotorSpeed(1, abs(int(0)));
  } else {
    WriteMotorSpeed(1, abs(int(PID)));
    WriteMotorSpeed(0, abs(int(0)));
  }

  lastTimeM1 = currentTime;
  //}
}
```

```
/*====================== PID MOTOR Kanan
============================*/
unsigned long lastTimeM2 = 0;

/*====================== PID MOTOR Kiri ============================*/
void pidComputeM2(float dt) {
  // put your main code here, to run repeatedly:
  unsigned long currentTime2 = millis();
  float lasterror2;
  float deltaTime2 = currentTime2 - lastTimeM2;
  float sumerror2;
  //if (deltaTime2 >= TimeSampling) {
  float freqSignal2 = (float)tickEncoder[1] / (dt / 1.0e3);  //(deltaTime/1.0e3) --> convert ms to
s
  tickEncoder[1] = 0;
  float rpm2 = freqSignal2 * 60.0 / (gearRatio * encoder_PPR);  //224.4 = p*N; p=step-up
gear ratio; N=pulses per revolution
  float error2 = (_SP[1] - rpm2);
  sumerror2 += error2;
  int P = (KP * error2);
  int I = (KI * sumerror2);
  int D = KD * (error2 - lasterror2) / dt;
  int PID2 = P + I + D;
  lasterror2 = error2;
  PID2 = constrain(PID2, -255, 255);
  if (PID2 > 0) {
    WriteMotorSpeed(3, abs(int(PID2)));
    WriteMotorSpeed(2, abs(int(0)));
  } else {
    WriteMotorSpeed(2, abs(int(PID2)));
    WriteMotorSpeed(3, abs(int(0)));
  }
  lastTimeM2 = currentTime2;
  // }
}
/*====================== PID MOTOR Kiri ============================*/
```

```
/*====================== Roda Controller ==========================*/
void drivePID(int x, int y, int speed, float deltaT) {
 if (y > 0) {
  WriteMotorSpeed(4, 255);  //Sikat Depan Mutar Ke Arah Depan
  WriteMotorSpeed(6, 255);  //Sikat Belakang Mutar Ke Arah Depan
  WriteMotorSpeed(5, 0);
  WriteMotorSpeed(7, 0);
  _SP[0] = speed;   //Maju Setpoint PID Motor Kanan
  _SP[1] = -speed;  //Maju Setpoint PID Motor Kiri
 } else if (y < 0) {
  _SP[0] = -speed;          //Maju Setpoint PID Motor Kiri
  _SP[1] = speed;           //Maju Setpoint PID Motor Kiri
  WriteMotorSpeed(5, 255);  //Sikat Depan Mutar Ke Arah Belakang
  WriteMotorSpeed(7, 255);  //Sikat Belakang Mutar Ke Arah Belakang
  WriteMotorSpeed(4, 0);
  WriteMotorSpeed(6, 0);
 } else if (x < 0) {
  _SP[0] = -speed;  //setting Setpoint PID
  _SP[1] = -speed;  //setting Setpoint PID
 } else if (x > 0) {
  _SP[0] = speed;  //setting Setpoint PID
  _SP[1] = speed;  //setting Setpoint PID
 } else {
  _SP[0] = 0;             //setting Setpoint PID
  _SP[1] = 0;             //setting Setpoint PID
  WriteMotorSpeed(4, 0);  //Stop Motor Sikat
  WriteMotorSpeed(6, 0);  //Stop Motor Sikat
  WriteMotorSpeed(5, 0);  //Stop Motor Sikat
  WriteMotorSpeed(7, 0);  //Stop Motor Sikat
 }
 //_SP[0] = 100;
 //_SP[1] = 100;
 pidComputeM1(deltaT);  //Komputasi PID Motor Kiri
 pidComputeM2(deltaT);  //Komputasi PID Motor Kiri
}
/*====================== Roda Controller ==========================*/
```

```
#ifndef Motor_h
#define Motor_h
void SerialPrintEncoder(void);
void InitMotor(void);
void drivePID(int x, int y, int speed,float deltaT);
#endif
```

3.3.5.2. *Coding* Kamera *Web Server*

```
#include "esp_camera.h"
#include <WiFi.h>

//
// WARNING!!! Make sure that you have either selected ESP32 Wrover Module,
//            or another board which has PSRAM enabled
//

// Select camera model
//#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
#define CAMERA_MODEL_AI_THINKER

#include "camera_pins.h"

const char* ssid = "DOME"
const char* password = "123456789"

void startCameraServer();

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
```

```
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  //init with high specs to pre-allocate larger buffers
  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif

  // camera init
  esp_err_t err = esp_camera_init(&config);
```

```
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();
  //initial sensors are flipped vertically and colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);//flip it back
    s->set_brightness(s, 1);//up the blightness just a bit
    s->set_saturation(s, -2);//lower the saturation
  }
  //drop down frame size for higher initial frame rate
  s->set_framesize(s, FRAMESIZE_QVGA);

#if defined(CAMERA_MODEL_M5STACK_WIDE)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}

void loop() {
```

```
  // put your main code here, to run repeatedly:
  delay(10000);
}
```

3.3.5.3. *Coding* ESP32

```
#include <WiFi.h>
#include <WiFiUdp.h>

// SSID dan Password WiFi
const char *ssid = "DOME";
const char *pwd = "123456789";
const char *udpAddress = "172.16.51.183";  //IP Basestation
const int udpPort = 9999;

WiFiUDP udp;
//Variabel Yang Menyimpan Data sensor dan lainnya, kemudian akan dikirim ke BaseStation
uint8_t Receive[13];
uint8_t ReceiveBaseStation[7];
void setup() {
  Serial.begin(115200);
  Serial2.begin(115200);

  WiFi.begin(ssid, pwd);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Ip: ");
  Serial.print(WiFi.localIP());
  // Memulai menjadi Server
}
void loop() {

  Serial2.read(Receive, 13);
  if (Receive[0] == 255) {
```

```
  uint8_t checksum = ~(Receive[0] + Receive[1] + Receive[2] + Receive[3] + Receive[4] +
Receive[5] + Receive[6] + Receive[7] + Receive[8] + Receive[9] + Receive[10] +
Receive[11]);
  if (checksum == Receive[12]) {
    // Mengirim data yang diterima dari sensor ke BaseStation melalui UDP
    udp.beginPacket(IPAddress(255, 255, 255, 255), udpPort);
    udp.write(Receive, 13);
    udp.endPacket();
  }
 }

 // Memeriksa apakah ada paket yang tersedia untuk dibaca dari BaseStation

  int packetSize = udp.parsePacket();
  if (packetSize) {
    Serial.println(packetSize);
    // Jika ada paket yang tersedia, membaca dan menyimpannya di dalam
ReceiveBaseStation
    udp.read(ReceiveBaseStation, packetSize);

    // Melakukan verifikasi data dari BaseStation (misalnya, dengan checksum)
    uint8_t checksum2 = ~(ReceiveBaseStation[0] + ReceiveBaseStation[1] +
ReceiveBaseStation[2] + ReceiveBaseStation[3] + ReceiveBaseStation[4] +
ReceiveBaseStation[5]);
    Serial.println(ReceiveBaseStation[0]);
    Serial.println(checksum2);

    if (ReceiveBaseStation[0] == 255) {
      if (checksum2 == ReceiveBaseStation[6]) {
        Serial.println("Data Benar dari BaseStation");

      }
    }
    // Mengirim kembali data yang diterima dari BaseStation ke Serial2 (atau ke perangkat
lain)
  }
 Serial2.write(ReceiveBaseStation, 7);
```

}

3.3.5.4. *Coding Base Station*

```
from hashlib import new
from PyQt5 import uic
from PyQt5.QtMultimedia import QCameraInfo
from PyQt5.QtWidgets import QApplication, QMainWindow, QDialog
from PyQt5.QtCore import QThread, pyqtSignal, pyqtSlot, QTimer, QDateTime, Qt
from PyQt5.QtGui import QImage, QPixmap, QPainter, QPen
import cv2,time,sys,sysinfo
import numpy as np
import random as rnd
#import keyboard
import server


class ThreadClass(QThread):
    ImageUpdate = pyqtSignal(np.ndarray)
    FPS = pyqtSignal(int)
    global camIndex
    global ip
    def run(self):
        URL = "http://"+ip+":81/stream"
        print(URL)
        if camIndex == 0:
            Capture = cv2.VideoCapture(URL)
        if camIndex == 1:
            Capture = cv2.VideoCapture(URL)

        Capture.set(cv2.CAP_PROP_FRAME_HEIGHT,480)
        Capture.set(cv2.CAP_PROP_FRAME_WIDTH,640)
        self.ThreadActive = True
        prev_frame_time = 10
        new_frame_time = 1
        fps = 0
        while self.ThreadActive:
            ret,frame_cap = Capture.read()
            flip_frame = cv2.flip(src=frame_cap,flipCode=1)
```

```
            new_frame_time = time.time()
            #fps = int(1/(new_frame_time-prev_frame_time))
            if ret:
                self.ImageUpdate.emit(flip_frame)
                self.FPS.emit(fps)
            prev_frame_time = new_frame_time

    def stop(self):
        self.ThreadActive = False
        self.quit()

class boardInfoClass(QThread):
    cpu = pyqtSignal(float)
    ram = pyqtSignal(tuple)
    temp = pyqtSignal(float)

    def run(self):
        self.ThreadActive = True
        while self.ThreadActive:
            cpu = sysinfo.getCPU()
            ram = sysinfo.getRAM()
            self.cpu.emit(cpu)
            self.ram.emit(ram)

    def stop(self):
        self.ThreadActive = False
        self.quit()

class randomColorClass(QThread):
    color = pyqtSignal(tuple)
    def run(self):
        self.ThreadActive = True
        while self.ThreadActive:
            color = ([rnd.randint(0,256),rnd.randint(0,256),rnd.randint(0,256)],
                     [rnd.randint(0,256),rnd.randint(0,256),rnd.randint(0,256)],
                     [rnd.randint(0,256),rnd.randint(0,256),rnd.randint(0,256)]
                     )
```

```
            self.color.emit(color)
            time.sleep(2)

    def stop(self):
        self.ThreadActive = False
        self.quit()

class Window_IOMonitor(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = uic.loadUi("IO_monitor.ui",self)

class Window_ErrorAlarm(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = uic.loadUi("Error.ui", self)

#   QLabel display
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = uic.loadUi("Opencv_PiDash.ui",self)

        self.online_cam = QCameraInfo.availableCameras()
        #self.camlist.addItems([c.description() for c in self.online_cam])
        self.camlist.addItems(['ESP32 CAM'])
        self.btn_start.clicked.connect(self.StartWebCam)
        self.btn_stop.clicked.connect(self.StopWebcam)
        self.IP.text()

        self.resource_usage = boardInfoClass()
        self.resource_usage.start()
        self.resource_usage.cpu.connect(self.getCPU_usage)
        self.resource_usage.ram.connect(self.getRAM_usage)

        self.randomColor_usage = randomColorClass()
        self.randomColor_usage.start()
```

```
        self.randomColor_usage.color.connect(self.get_randomColors)

        #Create Instance class
        self.Win_showIO = Window_IOMonitor()
        self.Win_showError = Window_ErrorAlarm()
        server.InitUDP()

        #QTimer Zone
        self.ready_lamp = QTimer(self, interval=100)
        self.ready_lamp.timeout.connect(self.Ready_lamp)
        self.ready_lamp.timeout.connect(self.Low_lamp)
        self.ready_lamp.start()

        self.loop = QTimer()
        self.loop.timeout.connect(self.LoopMain)
        self.loop.start()

        self.motor_on = QTimer(self, interval=1000)
        self.motor_on.timeout.connect(self.Running_lamp)


        self.lcd_timer = QTimer()
        self.lcd_timer.timeout.connect(self.clock)
        self.lcd_timer.start()

        self.Senddata = QTimer()
        self.Senddata.timeout.connect(self.SendataToESP32)
        self.Senddata.start()

        self.BatteryFull = True
        self.Status_lamp = [True,True,True]

        #button
        self.btn_motor.setEnabled(False)
        self.btn_motor.setText('Motor ON')
        self.btn_close.setEnabled(False)
        self.btn_motor.setCheckable(True)
```

```
        self.btn_motor.clicked.connect(self.Motor_on)

        self.maju.setCheckable(True)
        self.mundur.setCheckable(True)
        self.kiri.setCheckable(True)
        self.kanan.setCheckable(True)

        self.Sikat.setCheckable(True)
        self.Pompa.setCheckable(True)
        self.Control.setCheckable(True)

        self.maju.setEnabled(True)
        self.mundur.setEnabled(True)
        self.kiri.setEnabled(True)
        self.kanan.setEnabled(True)

        self.Sikat.setEnabled(True)
        self.Pompa.setEnabled(True)
        self.Control.setEnabled(True)


        self.btn_close.clicked.connect(self.Close_software)
        self.btn_iomonitor.clicked.connect(self.open_IOmonitor)

        self.Win_showError.btn_e_close.clicked.connect(self.Close_Error)
        self.btn_stop.setEnabled(False)

        self.batteryVolt = server.voltage

    def get_randomColors(self,color):
        self.RanColor1 = color[0]
        self.RanColor2 = color[1]
        self.RanColor3 = color[2]

    def getCPU_usage(self,cpu):
        self.Qlabel_cpu.setText(str(cpu) + " %")
        if cpu > 15: self.Qlabel_cpu.setStyleSheet("color: rgb(23, 63, 95);")
```

```
        if cpu > 25: self.Qlabel_cpu.setStyleSheet("color: rgb(32, 99, 155);")
        if cpu > 45: self.Qlabel_cpu.setStyleSheet("color: rgb(60, 174, 163);")
        if cpu > 65: self.Qlabel_cpu.setStyleSheet("color: rgb(246, 213, 92);")
        if cpu > 85: self.Qlabel_cpu.setStyleSheet("color: rgb(237, 85, 59);")

    def getRAM_usage(self,ram):
        self.Qlabel_ram.setText(str(ram[2]) + " %")
        if ram[2] > 15: self.Qlabel_ram.setStyleSheet("color: rgb(23, 63, 95);")
        if ram[2] > 25: self.Qlabel_ram.setStyleSheet("color: rgb(32, 99, 155);")
        if ram[2] > 45: self.Qlabel_ram.setStyleSheet("color: rgb(60, 174, 163);")
        if ram[2] > 65: self.Qlabel_ram.setStyleSheet("color: rgb(246, 213, 92);")
        if ram[2] > 85: self.Qlabel_ram.setStyleSheet("color: rgb(237, 85, 59);")

    def get_FPS(self,fps):
        self.Qlabel_fps.setText(str(fps))
        if fps > 5: self.Qlabel_fps.setStyleSheet("color: rgb(237, 85, 59);")
        if fps > 15: self.Qlabel_fps.setStyleSheet("color: rgb(60, 174, 155);")
        if fps > 25: self.Qlabel_fps.setStyleSheet("color: rgb(85, 170, 255);")
        if fps > 35: self.Qlabel_fps.setStyleSheet("color: rgb(23, 63, 95);")

    def clock(self):
        self.DateTime = QDateTime.currentDateTime()
        self.lcd_clock.display(self.DateTime.toString('hh:mm:ss'))

    #Open I/O Window monitor
    def open_IOmonitor(self):
        self.textEdit.append(f"{self.DateTime.toString('d MMMM yy hh:mm:ss')}: Open I/O
Monitor")
        self.Win_showIO.show()

    #Close Error Notification window
    def Close_Error(self):
        self.Win_error.close()

    @pyqtSlot(np.ndarray)
    def opencv_emit(self, Image):
        #QPixmap format
```

```
        original = self.cvt_cv_qt(Image)

        self.disp_main.setPixmap(original)
        self.disp_main.setScaledContents(True)



    def cvt_cv_qt(self, Image):
        rgb_img = cv2.cvtColor(src=Image,code=cv2.COLOR_BGR2RGB)
        h,w,ch = rgb_img.shape
        bytes_per_line = ch * w
        cvt2QtFormat = QImage(rgb_img.data, w, h, bytes_per_line, QImage.Format_RGB888)
        pixmap = QPixmap.fromImage(cvt2QtFormat)
        return pixmap
#-----------------------------------------------------------------------------------------------

    def StartWebCam(self):
        try:
            self.textEdit.append(f"{self.DateTime.toString('d MMMM yy hh:mm:ss')}: Start
Webcam ({self.camlist.currentText()})")
            self.btn_stop.setEnabled(True)
            self.btn_start.setEnabled(False)
            self.btn_close.setEnabled(False)
            self.btn_motor.setEnabled(True)
            self.Status_lamp[0]=True
            global camIndex
            global ip
            camIndex = self.camlist.currentIndex()
            ip = self.IP.text()
            print(ip)

            # Opencv QThread
            self.Worker1_Opencv = ThreadClass()
            self.Worker1_Opencv.ImageUpdate.connect(self.opencv_emit)
            self.Worker1_Opencv.FPS.connect(self.get_FPS)
            self.Worker1_Opencv.start()
```

```
        except Exception as error :
            pass

    def StopWebcam(self):
        self.textEdit.append(f"{self.DateTime.toString('d MMMM yy hh:mm:ss')}: Stop
Webcam ({self.camlist.currentText()})")
        self.btn_start.setEnabled(True)
        self.btn_stop.setEnabled(False)
        self.btn_close.setEnabled(True)
        self.btn_motor.setEnabled(False)
        self.Status_lamp[0]=False
        # Set Icon back to stop state
        self.Worker1_Opencv.stop()
        self.motor_on.stop()

    def Close_software(self):
        self.Worker1_Opencv.stop()
        self.resource_usage.stop()
        sys.exit(app.exec_())

    def Motor_on(self):
        if self.btn_motor.isChecked():
            self.textEdit.append(f"{self.DateTime.toString('d MMMM yy hh:mm:ss')}: Motor
ON")
            self.btn_motor.setText('Motor OFF')
            self.motor_on.start()
        else:
            self.textEdit.append(f"{self.DateTime.toString('d MMMM yy hh:mm:ss')}: Motor
OFF")
            self.motor_on.stop()
            self.btn_motor.setText('Motor ON')
            self.Qlabel_yellowlight.setStyleSheet("background-color: rgb(242, 214, 117); border-
radius:30px")


    # Motor, Tower Lamp Status
    def Low_lamp(self):#low voltage function
```

```
        global voltage
        self.batteryVolt = server.voltage
        if self.batteryVolt > 22:
         self.Battery.setStyleSheet("background-color: rgb(0, 255, 0);\nborder-radius:30px")
        elif self.batteryVolt > 15 and self.SPEED.value() < 22:
         self.Battery.setStyleSheet("background-color: rgb(255, 255, 0);\nborder-radius:30px")
        elif self.batteryVolt < 15:
         self.Battery.setStyleSheet("background-color: rgb(255, 0, 0);\nborder-radius:30px")


    def Ready_lamp(self):
        if self.Status_lamp[0]:
            self.Qlabel_greenlight.setStyleSheet("background-color: rgb(85, 255, 0); border-
radius:30px")
        else :
            self.Qlabel_greenlight.setStyleSheet("background-color: rgb(184, 230, 191); border-
radius:30px")


    def Running_lamp(self):
        if self.Status_lamp[1]: self.Qlabel_yellowlight.setStyleSheet("background-color:
rgb(255, 195, 0); border-radius:30px")
        else : self.Qlabel_yellowlight.setStyleSheet("background-color: rgb(242, 214, 117);
border-radius:30px")
        self.Status_lamp[1] = not self.Status_lamp[1]

    def Alarm_lamp(self):
        if self.Status_lamp[2]: self.Qlabel_yellowlight.setStyleSheet("background-color:
rgb(255, 0, 0); border-radius:30px")
        else : self.Qlabel_yellowlight.setStyleSheet("background-color:  rgb(255, 171, 175);
border-radius:30px")
        self.Status_lamp[2] = not self.Status_lamp[2]

    # override the key press event
    def keyPressEvent(self, event):
        if event.key() == Qt.Key.Key_W and  not self.maju.isChecked():
            self.maju.setChecked(True)
```

```
        elif event.key() == Qt.Key.Key_S and  not self.mundur.isChecked():
            self.mundur.setChecked(True)
        elif event.key() == Qt.Key.Key_A and  not self.kiri.isChecked():
            self.kiri.setChecked(True)
        elif event.key() == Qt.Key.Key_D and  not self.kanan.isChecked():
            self.kanan.setChecked(True)
        else:
            self.maju.setChecked(False)
            self.mundur.setChecked(False)
            self.kiri.setChecked(False)
            self.kanan.setChecked(False)


    def LoopMain(self):
        global port
        global yaw
        global pitch
        global roll
        global voltage
        global sensor1
        global sensor2
        global sensor3
        global sensor4

        if not self.btn_motor.isChecked():
            self.SPEED.setValue(0)

        server.ReceiveData()
        if self.maju.isChecked():
            print('maju')
            server.TransmitData[1]=1#Maju
        elif self.mundur.isChecked():
            print('mundur')
            server.TransmitData[1]=2#Mundur
        elif self.kiri.isChecked():
            print('kiri')
            server.TransmitData[1]=3#Kiri
```

```
elif self.kanan.isChecked():
    print('kanan')
    server.TransmitData[1]=4#Kanan
else:
    server.TransmitData[1]=0#Stop

if self.Pompa.isChecked():
    print('Pompa')
    server.TransmitData[2]=1#Pompa
else:
    server.TransmitData[2]=0#Pompa


if self.Sikat.isChecked():
    print('Sikat')
    server.TransmitData[3]=1#Sikat ON
else:
    server.TransmitData[3]=0#Sikat OFF


if self.Control.isChecked():
    print('Control')
    server.TransmitData[4]=1#Control
else:
     server.TransmitData[4]=0#Control


self.Speed.display(self.SPEED.value())
server.TransmitData[5]=int(self.SPEED.value())#Speed

self.YAW.display(server.yaw)
self.PITCH.display(server.pitch)
self.ROLL.display(server.roll)
self.Voltage.display(server.voltage)
self.IR1.display(server.sensor1)
self.IR2.display(server.sensor2)
self.IR3.display(server.sensor3)
self.IR4.display(server.sensor4)
```

```
        self.Debit.display(server.Flow)
        # + server.TransmitData[5] + server.TransmitData[6] + server.TransmitData[7] +
server.TransmitData[8]

    def SendataToESP32(self):
        server.TransmitData[0]=255
        server.TransmitData[6] = ~(server.TransmitData[0] + server.TransmitData[1] +
server.TransmitData[2] + server.TransmitData[3] + server.TransmitData[4]
+server.TransmitData[5])
        server.SendToESP32(server.TransmitData)
        #print(server.TransmitData[5])
        #server.TransmitData[1] = 0
        #server.TransmitData[2] = 0
        #server.TransmitData[3] = 0
        #server.TransmitData[4] = 0
        #print('send')

if __name__ == "__main__":

# IO Board
    IO_INPUT = [7,11,13,15]
    IO_OUTPUT = [12,16,18,22,32,36,38,40]
    #IO.setwarnings(False)
    #IO.setmode(IO.BOARD)

    for input_pins in IO_INPUT:
        pass#IO.setup(input_pins, IO.IN)

    for output_pins in IO_OUTPUT:
        pass#IO.setup(output_pins, IO.OUT, initial=IO.HIGH)

    app = QApplication([])
    window = MainWindow()
    #IO.add_event_detect(IO_INPUT[0],IO.FALLING,window.StartWebCam,500)
    #IO.add_event_detect(IO_INPUT[1],IO.FALLING,window.StopWebcam,500)
    window.show()
    app.exec_()
```

```
#IO.cleanup()
```