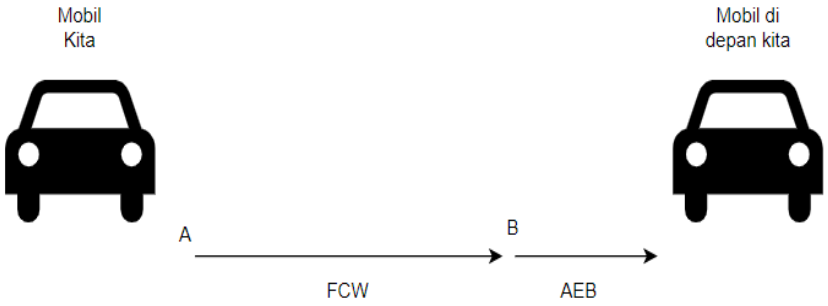
BAB III
ANALISIS DAN PERANCANGAN PENELITIAN

3.1 Perancangan Sistem AEB dan FCW

Dalam proses pembuatan *hardware*, mobil RC (*Remote Control*) digunakan sebagai *prototype* dari penelitian ini dengan alasan mempermudah mensimulasikan cara kerja sistem pada masalah yang akan diatasi dan menghemat biaya. Sistem penggerak yang digunakan adalah motor DC untuk mengontrol kecepatan. Untuk sistem rem yang diimplementasikan pada mobil RC, sensor *ultrasonic* HC-SR04 akan dipasang pada bagian depan mobil untuk mendeteksi ada tidaknya *obstacle* yang berada di depan mobil RC. Sensor *ultrasonic* akan mendeteksi berapa jarak dari mobil kita ke mobil di depan kita.

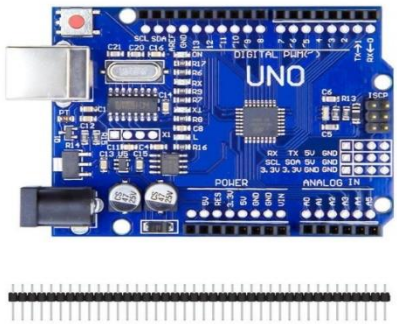


Gambar 3. 1 Gambaran Umum Cara Kerja FCW dan AEB

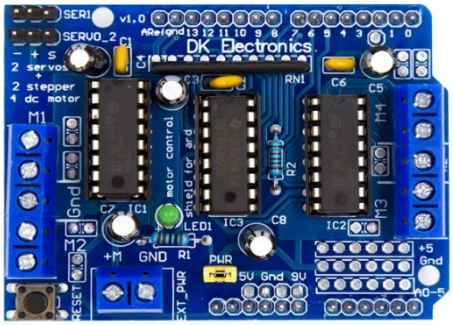
Gambar 3.1 menunjukkan ketika mobil RC melaju dengan kecepatan penuh (motor.setSpeed(255)) dan sensor mendeteksi adanya kemungkinan tabrakan dengan mobil di depan kita pada titik A dimana jaraknya adalah 35 cm dari *obstacle* di depan mobil untuk pengujian, maka sistem FCW akan memberikan peringatan pada pengemudi berupa suara dari *passive buzzer speaker* untuk memperingati pengemudi agar dapat melakukan pencegahan tabrakan. Jika sampai titik B dimana jaraknya adalah 25 cm dari *obstacle* di depan mobil pengemudi tidak melakukan apapun, maka AEB akan melakukan rem penuh secara otomatis untuk terhindar dari

tabrakan dengan *obstacle* di depan mobil. Teknik rem yang digunakan adalah memutarkan roda secara *counter clockwise* dengan kecepatan penuh. Proses rem menggunakan teknik tersebut memiliki kekurangan yaitu dapat memundurkan kendaraan jika lama waktu roda diputar *counter clockwise* tidak tepat. Teknik yang digunakan tidak aman untuk diujikan secara nyata namun karena tidak memiliki rem cakram, maka teknik tersebut digunakan sebagai mekansime rem pada *prototype* mobil RC ini.

Berikut adalah komponen-komponen secara keseluruhan yang dibutuhkan untuk membuat *prototype* agar dapat mensimulasikan sistem AEB dan FCW :



Gambar 3. 2 Arduino Uno

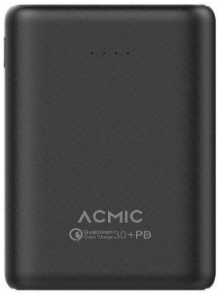


Gambar 3. 3 Motor Driver Shield L293D

Gambar 3. 4 Sensor Ultrasonic

Gambar 3. 5 Bluetooth Module HC-06

Gambar 3. 6 Kabel USB Header Clone



Gambar 3. 7 Powerbank

Gambar 3. 8 Passive Buzzer Speaker

Arduino UNO berperan sebagai penyambung setiap komponen yang dibutuhkan pada sistem rem ini agar setiap komponen dapat menjalankan fungsinya masing-masing dan bisa saling berkomunikasi agar dapat menghasilkan output yang diinginkan. Tentunya, Arduino ini juga dikoding melalui software Arduino IDE agar dapat menjalankan fungsionalitas sistem secara keseluruhan.

Motor Driver Shield dipilih karena fungsinya untuk mengontrol motor DC, dan dapat digabungkan dengan Arduino UNO sehingga dapat meminimalisir penempatan komponen di dalam mobil RC dikarenakan ruangan yang minim di dalam mobil RC.

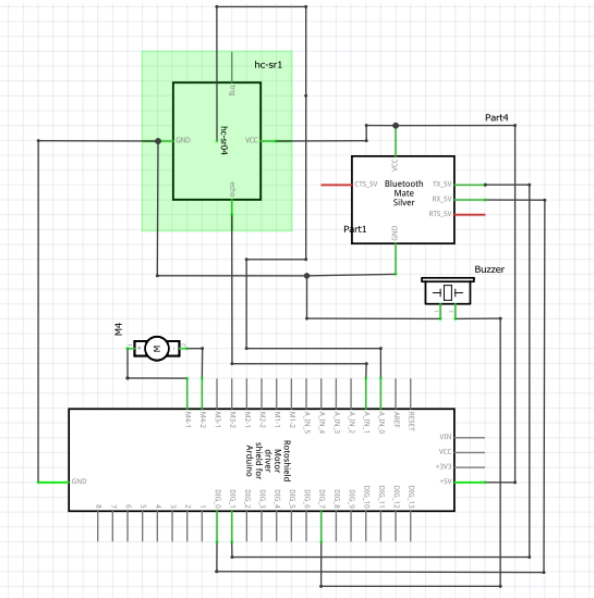
Sensor *ultrasonic* digunakan untuk mendeteksi *obstacle* yang ada di depan mobil RC. Sensor akan mendeteksi *obstacle* pada 2 jarak yang berbeda seperti pada Gambar 3.1. Jarak pertama adalah di bawah 40 cm dimana ketika sensor telah mendeteksi jarak tersebut, *alert* akan dibunyikan. Pada jarak kedua yaitu di bawah 20 cm, rem otomatis akan dilakukan oleh sistem.

Bluetooth Module HC-06 digunakan untuk mengontrol mobil RC menggunakan *Bluetooth* melewati aplikasi *"Arduino Bluetooth RC Car"* yang dapat diunduh di *Google Play Store* sehingga dapat dikontrol melalui *smartphone Android* pribadi.

Powerbank menjadi sumber tenaga untuk Arduino UNO dan *Motor Driver Shield*. Untuk menghubungkan *Powerbank* ke Arduino UNO digunakan Kabel USB *Header Clone*.

Passive buzzer speaker digunakan untuk menjalankan sistem FCW dimana *buzzer* akan membunyikan suara pada saat sensor ultrasonik mendeteksi *obstacle* pada jarak di bawah 40 cm.

Untuk gambar 3.9, menunjukkan gambaran skematik *hardware* dimana terdapat komponen *bluetooth module* (HC-06), sensor *ultrasonic* (HC-SR04), *passive buzzer speaker*, dan 1 motor DC yang terhubung dengan *motor driver shield*



Gambar 3. 9 Hardware Schematic

3.2 Perancangan Driver Drowsiness Detection System

Dalam proses pembuatan deteksi *drowsiness*, penulis membuat sebuah program dengan bahasa pemrograman *python* 3 yang bisa mendeteksi wajah dan

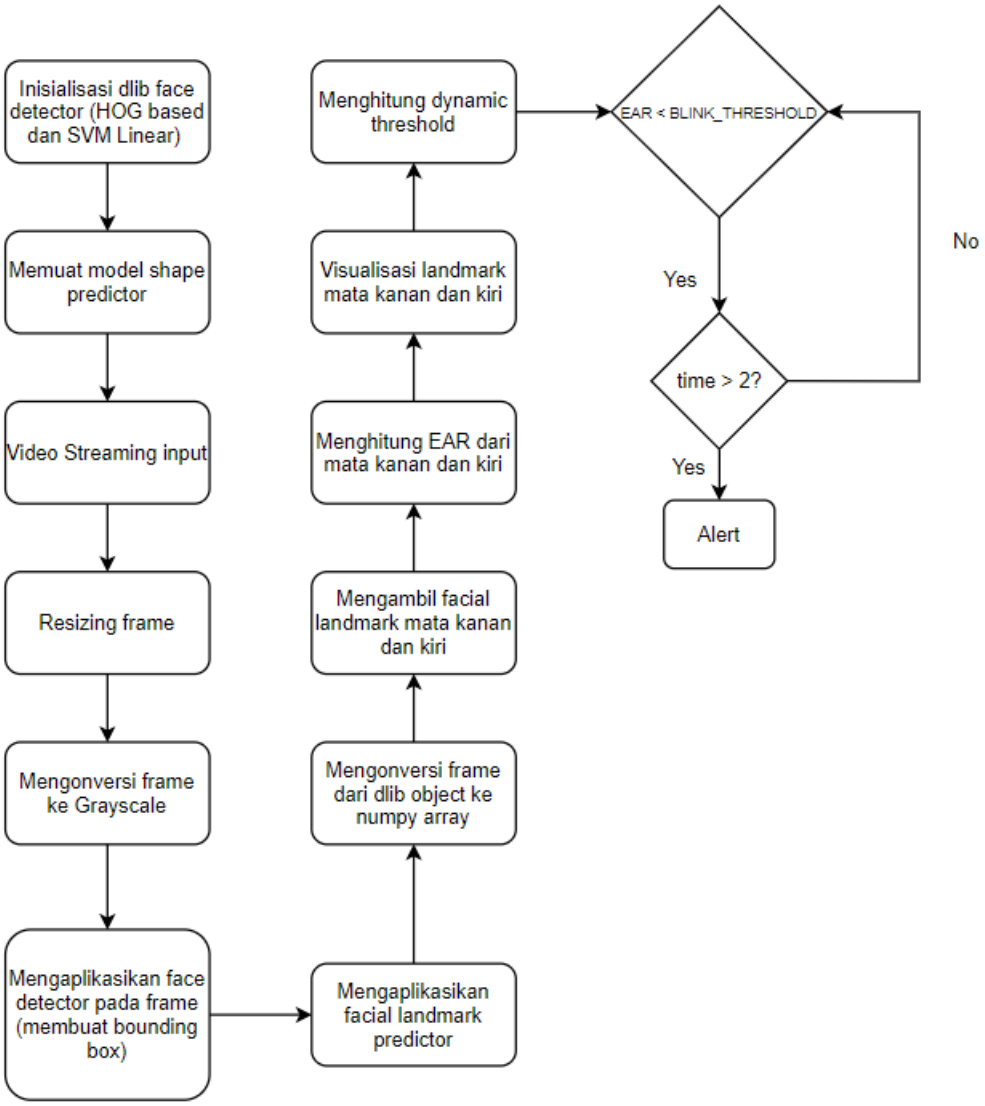
landmark dari wajah. *Library* yang dibutuhkan untuk dapat melakukan deteksi tersebut ada VideoStream dari imutils.video untuk dapat mengakses kamera secara *real-time*, face_utils dari imutils untuk mengonversi hasil output dari bentuk wajah yang didapatkan dari *facial landmark predictor* menjadi *numpy array* dan memudahkan kita untuk menghitung *bounding box* dari deteksi wajah tertentu dari model dlib berbasis HOG dan SVM *Linear*, argparse untuk membuat *command line arguments* agar dapat secara spesifik menjalankan file yang mana dan model yang mana, imutils untuk mengubah ukuran dari gambar sesuai kebutuhan agar dapat mengurangi *noise* dan mempercepat *pipeline* berjalan, time untuk memberikan delay agar kamera saat dibuka dapat menyesuaikan kualitas frame yang rendah dan menyesuaikan kondisi cahaya, serial untuk koneksi dengan Arduino UNO untuk membunyikan *buzzer* saat kondisi *drowsy* dideteksi, dlib untuk *face detector* dan *facial landmark predictor*, cv2 untuk *open*cv *bindings*, dan distance dari scipy.spatial untuk perhitungan EAR.

Cara kerja sistem *driver drowsiness detection* dimulai dari melakukan inisialisasi dlib *face detector* yang berdasarkan HOG dan SVM Linear. HOG bekerja dengan cara menguraikan gambar menjadi sel-sel kotak kecil lalu HOG pada setiap sel tersebut akan dihitung dan di normalisasi hasil perhitungannya menggunakan *block-wise pattern*. SVM bekerja dengan cara membuat garis yang memisahkan gambar mana yang merupakan wajah dan gambar mana yang bukan wajah. Alasan menggunakan HOG dan SVM karena [18] SVM yang dilatih pada fitur HOG merupakan standar *de facto* pada proses melakukan persepsi visual pada gambar. Setelah melakukan inisialisasi dlib, model *shape predictor* dimuat menggunakan *file* shape_predictor_68_face_landmarks.dat yang merupakan sebuah *binary file* dimana hanya dlib yang mengerti cara memuat model tersebut dan menjalankan *facial landmark predictor*. Setelah itu *video camera* dari laptop dibuka dimana terlebih dahulu diberi delay satu sampai dua detik agar *video camera* dapat melakukan *auto-adjustment* seperti menyesuaikan cahaya. *Frame* yang diperoleh dari *video* akan diubah ukurannya dengan tujuan mengurangi *noise* dan mempercepat proses *pipeline*, lalu akan di konversi ke *grayscale* karena sebagian besar dari pendeteksi objek yang berdasarkan HOG dan SVM Linear dilatih pada gambar *grayscale* sehingga *frame* yang didapat perlu di *pre-processed* juga ketika

ingin membuat prediksi wajah. *Frame* yang telah di *grayscale* akan diproses oleh *face detector* yang berdasarkan HOG dan SVM Linear dimana akan menghasilkan *bounding box* pada wajah yang dideteksi. Untuk setiap *bounding box* pada *frame*, akan di *apply facial landmark predictor* dalam area *bounding box* dan akan menghasilkan 68 *facial landmark points* yang muncul dari model *predictor* kita. Hasil tersebut karena masih merupakan objek dlib, akan dikonversi ke *numpy array* lalu koordinat/titik poin *facial landmark* dari mata kanan dan kiri akan di ekstrak dan hasil esktrak akan digunakan untuk menghitung EAR untuk kedua mata. Visualisasi landmark dari mata kanan dan kiri akan dilakukan menggunakan cv2 lalu angka EAR jg akan dimunculkan pada *frame* di *video*. *Dynamic* threshold akan dihitung dengan mengambil 10 angka EAR mata terbuka dan dihitung rata-rata dengan rumus $total_{EARopen} = (\sum_{1}^{10} EARopen)/10$, lalu mengambil 10 angka EAR mata tertutup dan dihitung rata-rata dengan rumus $total_{EARclosed} = (\sum_{1}^{10} EARclosed)/10$, dan terakhir akan dihitung *threshold* dengan rumus $dynamic_{threshold} = (total_{EARopen} + total_{EARclosed})/2$.

Deteksi akan dimulai dengan kondisi jika angka EAR lebih kecil daripada BLINK_THRESHOLD dan jika setelah 2 detik lebih masih terdeteksi lebih kecil, maka bunyi *alert* akan dibunyikan untuk memperingati pengemudi untuk fokus

kembali menyetir karena pengemudi terdeteksi dalam kondisi mengantuk. Gambaran secara *flowchart* dapat dilihat pada Gambar 3.10.



Gambar 3. 10 Flowchart Driver Drowsiness Detection