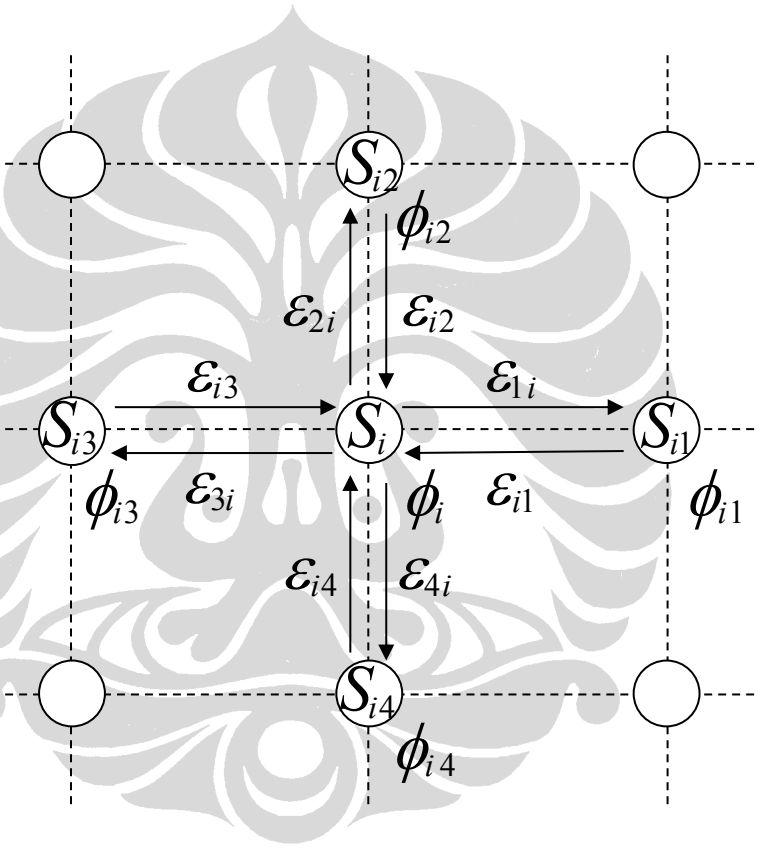
BAB 4

DESAIN DAN PENGENDALIAN SISTEM LAMPU LALU LINTAS

Bab ini akan memaparkan model matematik dari sistem lampu lalu lintas yang akan dibuat berikut dengan penjabaran dan penjelasannya.*Urban traffic sinyal network* dimodelkan dengan *bi-directed graph* seperti pada gambar berikut.



Gambar 4.1 Model *signal network*

Setiap persimpangan dilambangkan dengan node $S_i(1 \le i \le N)$, dan tetangga dari sinyal S_i adalah S_{ij} yang memiliki sinyalnya sendiri yaitu S_j.

ϕ_i menyatakan fase dari satu putaran sinyal S_i, dan $\phi_{ij}(j \in \{1,2,...,n_i\})$ merupakan fase untuk sinyal S_{ij}, di mana n_i adalah banyaknya sinyal yang

dipasangkan dengan S_i. ε_{ij} adalah aliran kendaraan yang telah dinormalisasi, yang didefinisikan sebagai:

$$\varepsilon_{ij} = \frac{1}{\rho_{im} Ti} \int_{t}^{t+T_i} \rho_{ij}(t)dt \qquad \textbf{(4.1)}$$

dimana $\rho_{im} = const.$ Merupakan kapasitas jalan antara signal S_i dan S_{ij}.

$$\rho_{im} = R_{ij} / L_k^{ij}(t) \qquad \textbf{(4.2)}$$

dimana R_{ij} = panjang jalan antara signal S_i dan S_{ij}.

$L_k^{ij}(t)$ = Panjang kendaraan pada ruas jalan signal S_i dan S_{ij}.

$\rho_{ij}(t)$ merupakan kepadatan dari jalan antara sinyal S_i dan S_{ij} pada interval waktu $[t, t+T_i]$.

$$\rho_{ij}(t) = \frac{\sum_{k=1}^{V_{ij}} L_k^{ij}(t)}{R_{ij}} \qquad \textbf{(4.3)}$$

dimana R_{ij} = panjang jalan antara signal S_i dan S_{ij}.

V_{ij} = volume kendaraan pada ruas jalan antara signal S_i dan S_{ij}.

$L_k^{ij}(t)$ = Panjang kendaraan pada ruas jalan signal S_i dan S_{ij}.

4.1 Deskripsi sinyal network dengan menggunakan nonlinear coupled oscillator

Pada bagian ini, akan dijabarkan *traffic signal network* dengan menggunakan sebuah sistem *nonlinear oscillators* dengan *coupling* terdekat. Secara spesifik, akan diterapkan deskripsi model fase dari *coupled oscillator* yang diperkenalkan oleh Kuramoto, yaitu sebuah model yang digeneralisasi dan dinyatakan dengan:

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{n_i}\sum_{j=1}^{n_i}\Gamma_i(\phi_i, \phi_{ij}) \qquad \textbf{(4.4)}$$

dimana:

$$\Gamma_i(\phi_i + 2\pi, \phi_{ij}) = \Gamma_i(\phi_i, \phi_{ij} + 2\pi) = \Gamma_i(\phi_i, \phi_{ij}) \qquad \textbf{(4.5)}$$

Dapat dikatakan pula bahwa Γ adalah fungsi yang periodik dengan periode 2π, sedangkan ω_i adalah frekuensi alami dari sinyal S_i yang ditentukan dengan $\omega_i = \frac{2\pi}{T_i}$, dengan K merupakan konstanta positif.

Untuk pengulangan dan kemudahan, digunakan fungsi sinus untuk memodelkan oscillator yang dinyatakan dengan fungsi matematik sebagai berikut:

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{n_i}\sum_{j=1}^{n_i}\varepsilon_{ij}\sin(\phi_{ij} - \phi_i) \qquad \textbf{(4.6)}$$

Formula Euler menyatakan bahwa,

$$\sin x = \frac{e^{ix} - e^{-ix}}{2i}$$

Jika $x = (\phi_{ij} - \phi_i)$, maka dengan menggunakan formula Euler, persamaan (4.6) dapat disusun ulang menjadi persamaan berikut,

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{n_i}\sum_{j=1}^{n_i}\varepsilon_{ij}\frac{e^{i(\phi_{ij}-\phi_i)} - e^{-i(\phi_{ij}-\phi_i)}}{2i} \quad \textbf{(4.7)}$$

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\sum_{j=1}^{n_i}\left(\varepsilon_{ij}\frac{e^{i(\phi_{ij}-\phi_i)}}{n_i} - \varepsilon_{ij}\frac{e^{-i(\phi_{ij}-\phi_i)}}{n_i}\right) \quad \textbf{(4.8)}$$

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\left(\sum_{j=1}^{n_i}\varepsilon_{ij}\frac{e^{i(\phi_{ij}-\phi_i)}}{n_i} - \sum_{j=1}^{n_i}\varepsilon_{ij}\frac{e^{-i(\phi_{ij}-\phi_i)}}{n_i}\right) \quad \textbf{(4.9)}$$

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\left(\sum_{j=1}^{n_i}\varepsilon_{ij}\frac{e^{i\phi_{ij}-i\phi_i}}{n_i} - \sum_{j=1}^{n_i}\varepsilon_{ij}\frac{e^{-i\phi_{ij}+i\phi_i}}{n_i}\right) \quad \textbf{(4.10)}$$

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\left(e^{-i\phi_i}\sum_{j=1}^{n_i}\frac{\varepsilon_{ij}}{n_i}e^{i\phi_{ij}} - e^{i\phi_i}\sum_{j=1}^{n_i}\frac{\varepsilon_{ij}}{n_i}e^{-i\phi_{ij}}\right) \quad \textbf{(4.11)}$$

Definisikan A_i dan B_i sebagai berikut:

$$A_i = \sum_{j=1}^{n_i}\frac{\varepsilon_{ij}}{n_i}e^{i\phi_{ij}} \quad \textbf{(4.12)}$$

$$B_i = \sum_{j=1}^{n_i}\frac{\varepsilon_{ij}}{n_i}e^{-i\phi_{ij}} \quad \textbf{(4.13)}$$

Maka persamaan **(4.11)** dapat dinyatakan dengan:

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\left(e^{-i\phi_i}A_i - e^{i\phi_i}B_i\right) \qquad \textbf{(4.14)}$$

Formula Euler menyatakan bahwa,

$$e^{ix} = \cos x + i\sin x$$

$$e^{-ix} = \cos x - i\sin x$$

Jika $x = \phi_i$, maka dengan menggunakan formula Euler, persamaan (4.14) dapat disusun ulang menjadi persamaan berikut,

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\left((\cos\phi_i - i\sin\phi_i)A_i - (\cos\phi_i + i\sin\phi_i)B_i\right) \qquad \textbf{(4.15)}$$

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\left(A_i\cos\phi_i - A_i i\sin\phi_i - B_i\cos\phi_i - B_i i\sin\phi_i\right) \qquad \textbf{(4.16)}$$

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\left((A_i\cos\phi_i - B_i\cos\phi_i) - i(A_i\sin\phi_i + B_i\sin\phi_i)\right) \qquad \textbf{(4.17)}$$

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\left((A_i - B_i)\cos\phi_i - (A_i + B_i)i\sin\phi_i\right) \qquad \textbf{(4.18)}$$

Dimana,

$$A_i - B_i = \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} e^{i\phi_{ij}} - \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} e^{-i\phi_{ij}}$$

$$A_i - B_i = \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} 2i \left(\frac{e^{i\phi_{ij}} - e^{-i\phi_{ij}}}{2i} \right)$$

$$A_i - B_i = 2 \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} i \sin \phi_{ij} \qquad \textbf{(4.19)}$$

$$A_i + B_i = \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} e^{i\phi_{ij}} + \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} e^{-i\phi_{ij}}$$

$$A_i + B_i = \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} 2 \left(\frac{e^{i\phi_{ij}} + e^{-i\phi_{ij}}}{2} \right)$$

$$A_i + B_i = 2 \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} \cos \phi_{ij} \qquad \textbf{(4.20)}$$

Jika,

$$a_i = \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} \cos \phi_{ij} \qquad \textbf{(4.21)}$$

$$b_i = \sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} \sin \phi_{ij} \qquad \textbf{(4.22)}$$

Sehingga persamaan (4.19) dan (4.20) dapat dituliskan sebagai,

$$A_i - B_i = 2ib_i \quad \textbf{(4.23)}$$

$$A_i + B_i = 2a_i \quad \textbf{(4.24)}$$

Setelah persamaan a_i dan b_i didapatkan, persamaan **(4.18)** dapat disusun ulang sehingga menjadi:

$$\dot{\phi}_i(t) = \omega_i + \frac{K}{2i}\left(2ib_i \cos\phi_i - 2ia_i \sin\phi_i\right) \quad \textbf{(4.25)}$$

$$\dot{\phi}_i(t) = \omega_i + K\left(b_i \cos\phi_i - a_i \sin\phi_i\right) \quad \textbf{(4.26)}$$

Selanjutnya, persamaan **(4.26)** dapat disampaikan sebagai *model entrainment*, yaitu:

$$\dot{\phi}_i(t) = \omega_i + \sigma_i K \sin(\bar{\phi}_i - \phi_i) \quad \textbf{(4.27)}$$

Dengan menyamakan persamaan **(4.26)** dan persamaan **(4.27),** maka diperoleh hubungan seperti berikut:

$$\sigma_i \sin(\bar{\phi}_i - \phi_i) = b_i \cos\phi_i - a_i \sin\phi_i \quad \textbf{(4.28)}$$

$$\sigma_i (\sin\bar{\phi}_i \cos\phi_i - \sin\phi_i \cos\bar{\phi}_i) = b_i \cos\phi_i - a_i \sin\phi_i \quad \textbf{(4.29)}$$

$$\sigma_i \sin\bar{\phi}_i \cos\phi_i - \sigma_i \sin\phi_i \cos\bar{\phi}_i = b_i \cos\phi_i - a_i \sin\phi_i \quad \textbf{(4.30)}$$

Dengan menyamakan koefisien cos dan sin maka didapatkan:

$$b_i = \sigma_i \sin \bar{\phi}_i \tag{4.31}$$

$$a_i = \sigma_i \cos \bar{\phi}_i \tag{4.32}$$

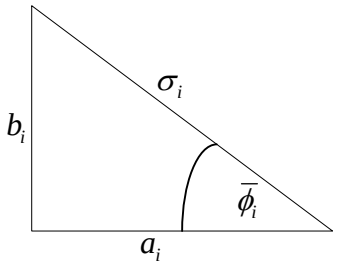
Sehingga hubungan antara σ_i, a_i dan b_i adalah:

$$\tan \bar{\phi}_i = \frac{b_i}{a_i} \tag{4.33}$$

$$\bar{\phi}_i = \arctan\left(\frac{b_i}{a_i}\right) \tag{4.34}$$

$$\sin \bar{\phi}_i = \frac{b_i}{\sigma_i}$$

$$\cos \bar{\phi}_i = \frac{a_i}{\sigma_i}$$



$$\sigma_i = \sqrt{a_i^2 + b_i^2} \tag{4.35}$$

4.2 Split setting

Bagian ini akan mengatur *split setting* dari lampu lalu lintas. Pertama-tama definisikan waktu putaran dari sinyal S_i sebagai berikut:

$$T_i(\tau_i) = T_{i1}(\tau_i) + T_{i2}(\tau_i) + 2T_{cl} \tag{4.36}$$

Termasuk dalam persamaan diatas, T_{cl} merupakan konstanta *clearence*, dimana $T_{i1}(\tau_i)$ dan $T_{i2}(\tau_i)$ secara berurutan adalah *split time* dari sinyal S_i pada arah vertikal dan horizontal.

Split time T_{ik}, di*update* pada setiap permulaan putaran dan proposional dengan jumlah dari arus kendaraan yang masuk, baik dari arah horizontal maupun vertikal.

Untuk melihat total *flow* yang masuk ke persimpangan, digunakan rumus berikut:

$$r_{i1} = \varepsilon_{i,1} + \varepsilon_{i,3}, r_{i2} = \varepsilon_{i,2} + \varepsilon_{i,4} \tag{4.37}$$

Dari persamaan diatas, penulis dapat mencari *desired split time* untuk arus kendaraan yang masuk dari arah horizontal dan vertikal dengan persamaan di bawah ini:

$$T_{ik}^{*} = \frac{r_{ik}(\tau_i)}{r_{i1}(\tau_i) + r_{i2}(\tau_i)}(T_i(\tau_i) - 2T_{cl}), (k = 1,2) \quad \textbf{(4.38)}$$

Setelah itu, *updating function* dapat didefinisikan dengan persamaan berikut:

$$T_{ik}(\tau_i + 1) = T_{ik}(\tau_i) + \gamma(T_{ik}^{*} - T_{ik}(\tau_i)), (k = 1,2) \quad \textbf{(4.39)}$$

Di mana γ merupakan konstanta perubahan. Yang akan menentukan seberapa cepat perubahan dari *split time* menuju *desired split time*.

4.3 Self-organizing dari penyesuaian offset

Bagian ini akan membahas strategi untuk menyesuaikan pola *offset* dari lampu lalu lintas dengan menggunakan *mutual entrainment*[1].

Misalkan ψ_i menyatakan fase relatif antara $\bar{\phi}_i$ dan ϕ_i yang didefinisikan sebagai:

$$\psi_i = \bar{\phi}_i - \phi_i \quad \textbf{(4.40)}$$

Maka dinamik dari fase relatifnya dapat diekspresikan dengan:

$$\dot{\psi}_i = \Omega_i - \omega_i - \sigma_i K \sin(\psi_i) \quad \textbf{(4.41)}$$

dimana Ω_i adalah frekuensi alami dari fase rata-rata $\overline{\phi}_i$.

Dengan memeriksa *fixed point* dari dinamik tersebut dengan memperhitungkan sifat dari fungsi *sinus* yang memiliki nilai antara -1 dan 1, maka *fixed point* pada persamaan **(4.41)** hanya terjadi jika:

$$0 < \frac{\Omega_i - \omega_i}{K\sigma_i} \leq 1 \quad \textbf{(4.42)}$$

Atau,

$$\left| \frac{\Omega_i - \omega_i}{\sigma_i K} \right| \leq 1 \quad \textbf{(4.43)}$$

phase locking akan terjadi ketika,

$$\dot{\psi}_i = 0 \quad \textbf{(4.44)}$$

$$\Omega_i - \omega_i - K\sigma_i \sin\psi_i = 0$$

$$K\sigma_i \sin\psi_i = \Omega_i - \omega_i$$

$$\sin\psi_i = \frac{\Omega_i - \omega_i}{K\sigma_i}$$

$$\psi_i = \arcsin\left(\frac{\Omega_i - \omega_i}{\sigma_i K}\right) \quad \textbf{(4.45)}$$

Saat *oscillator* S_i di *phase lock* dengan *oscillator* disampingnya sehingga nilai σ_i akan menjadi konstanta $\tilde{\sigma}_i$. Nilai dari $\tilde{\sigma}_i$ akan menjadi,

$$\tilde{\sigma}_i = \sqrt{\tilde{a}_i^2 + \tilde{b}_i^2}$$

$$\tilde{\sigma}_i = \sqrt{\left(\sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} \cos\Delta\phi_{ij}\right)^2 + \left(\sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} \sin\Delta\phi_{ij}\right)^2}$$

$$\tilde{\sigma}_i = \sqrt{\frac{1}{n_i^2}\left(\sum_{j=1}^{n_i} \varepsilon_{ij} \cos\Delta\phi_{ij}\right)^2 + \frac{1}{n_i^2}\left(\sum_{j=1}^{n_i} \varepsilon_{ij} \sin\Delta\phi_{ij}\right)^2}$$

$$\tilde{\sigma}_i = \sqrt{\frac{1}{n_i^2}\left(\left(\sum_{j=1}^{n_i} \varepsilon_{ij} \cos\Delta\phi_{ij}\right)^2 + \left(\sum_{j=1}^{n_i} \varepsilon_{ij} \sin\Delta\phi_{ij}\right)^2\right)}$$

$$\tilde{\sigma}_i = \frac{1}{n_i}\sqrt{\left(\sum_{j=1}^{n_i} \varepsilon_{ij} \cos\Delta\phi_{ij}\right)^2 + \left(\sum_{j=1}^{n_i} \varepsilon_{ij} \sin\Delta\phi_{ij}\right)^2} \quad \textbf{(4.46)}$$

$\Delta\phi_{ij}$ adalah selisih fase terhadap ϕ_i yang didefinisikan sebagai berikut,

$$\Delta\phi_{ij} = \phi_{ij} - \phi_i \quad \textbf{(4.47)}$$

Misalkan t_{ij}^* merupakan waktu yang dibutuhkan untuk sebuah kendaraan mencapai persimpangan S_i dari S_{ij}, maka *offset* yang diinginkan antara sinyal S_{ij} dan S_i ditentukan oleh:

$$\Delta\phi_{ij}^* = 2\pi \frac{t_{ij}^*}{T_i} (mod\, 2\pi) \quad \textbf{(4.48)}$$

dimana untuk *secondary direction* ditentukan oleh,

$$\Delta\phi_{ij}^* = 2\pi \frac{t_{ij}^* + (T_{i-1,1} - T_{i,1})}{T_i} (\mathrm{mod}\, 2\pi) \quad \textbf{(4.49)}$$

Waktu yang diharapkan, yaitu t_{ij}^* adalah:

$$t_{ij}^* = \frac{l_{ij}}{v_{ij}^{max}} \quad \textbf{(4.50)}$$

dimana l_{ij} merupakan jarak antar sinyal yang bersebelahan dan v_{ij}^{max} adalah batas kecepatan tertinggi dari jalan antara interval (S_{ij}, S_i).

Frekuensi alami yang diinginkan dari *oscillator* S_i adalah:

$$\omega_i^* = \Omega_i - \tilde{\sigma}_i^* K \sin \Delta\bar{\phi}_i^* \quad \textbf{(4.51)}$$

Untuk mencapai fase relatif yang diinginkan, digunakan persamaan seperti ini:

$$\Delta\bar{\phi}_i^* = \arctan\left(\frac{\sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} \sin \Delta\phi_{ij}^*}{\sum_{j=1}^{n_i} \frac{\varepsilon_{ij}}{n_i} \cos \Delta\phi_{ij}^*}\right)$$

$$\Delta\bar{\phi}_i^* = \arctan\left(\frac{\sum_{j=1}^{n_i} \varepsilon_{ij} \sin \Delta\phi_{ij}^*}{\sum_{j=1}^{n_i} \varepsilon_{ij} \cos \Delta\phi_{ij}^*}\right) \quad \textbf{(4.52)}$$

4.4 Aturan perubahan frekuensi alami

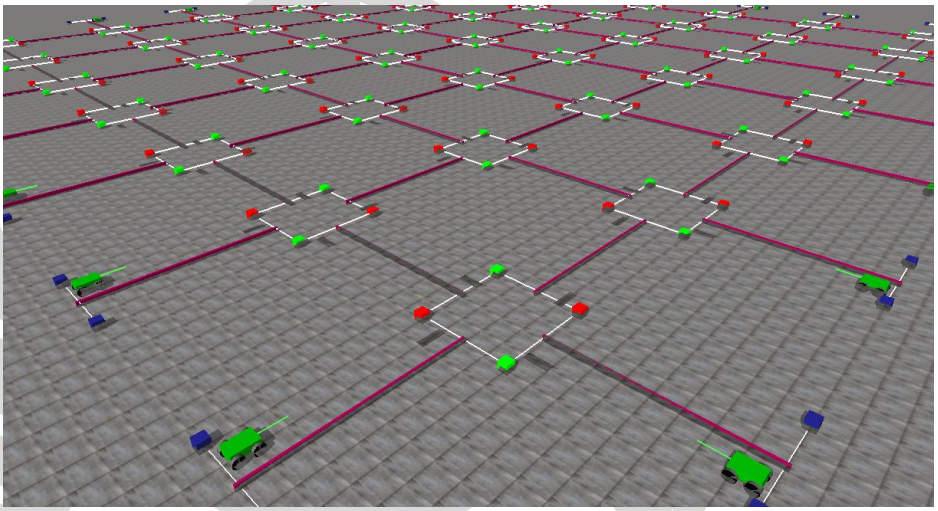
Berikut ini adalah persamaan untuk mengatur perubahan dari frekuensi alamiah:

$$\omega_i(\tau_i + 1) = \omega_i(\tau_i) + \alpha(\omega_i^* - \omega_i(\tau_i)) + \beta(\tilde{\omega}_i^* - \omega_i(\tau_i)) \quad \textbf{(4.53)}$$

Dimana τ_i adalah *dimensionless-time,* yaitu waktu yang telah diberi skala yang berhubungan dengan periode dari sinyal dan $\alpha, \beta \in (0,1)$ merupakan konstanta bernilai positif. Pada aturan perubahan ini, sebuah frekuensi dasar $\tilde{\omega}_i^*$ digunakan untuk setiap sinyal yang berguna untuk mencegah pergeseran dari frekuensi natural di setiap sinyal.

BAB 5
RANCANGAN SIMULASI

Simulasi digunakan untuk memodelkan algoritma yang telah dijabarkan pada bab sebelumnya untuk melihat apakah algoritma yang digunakan berjalan dengan baik. Simulasi dikembangkan dengan menggunakan *Open Dynamics Engine* (ODE).



Gambar 5.1 *Screenshot* tampilan simulasi

Open Dynamics Engine (ODE) adalah *platform* berbasis OpenGL yang ditulis dengan bahasa pemrograman C++ untuk mensimulasikan obyek 3 dimensi. ODE memiliki fitur-fitur seperti *joint*, *collision detection*, *error reduction parameter*, dan lain-lain. Dunia dalam ODE merepresentasikan dunia nyata dan dapat memiliki beberapa *space* sekaligus. Berikut ini adalah kelas-kelas yang dibuat dalam pengembangan simulasi:

- ***Class Car***

  Class ini memiliki *constructor* yang berfungsi untuk menciptakan objek mobil di dunia ODE dan juga memiliki fungsi *draw* untuk menggambar

objek mobil tersebut. *Class* ini menyimpan posisi awal mobil diciptakan dan juga posisi saat dia sudah berjalan.

- ***Class Sign***

  Class ini memiliki *constructor* yang berfungsi untuk menciptakan objek sign di dunia ODE dan juga memiliki fungsi *draw* untuk menggambar objek sign tersebut. *Class* ini menyimpan posisi koordinat sign. Sign adalah penanda bagi mobil bahwa ia berada dekat dengan lokasi persimpangan.

- ***Class Border***

  Class ini memiliki *constructor* yang berfungsi untuk menciptakan objek *border* di dunia ODE dan juga memiliki fungsi *draw* untuk menggambar objek border tersebut. *Class* ini menyimpan posisi koordinat *border*. *Border* adalah pembatas jalan antara jalur kiri dan kanan dalam suatu ruas jalan.

- ***Class Carcntr***

  Class ini memiliki *constructor* yang berfungsi untuk menciptakan objek carcntr di dunia ODE dan juga memiliki fungsi *draw* untuk menggambar objek carcntr tersebut. *Class* ini menyimpan posisi koordinat objek carcntr. *Carcntr* adalah sensor yang berada di ujung-ujung jalan yang berfungsi untuk menghitung jumlah mobil yang keluar dan masuk ruas jalan. *Carcntr* juga berguna sebagai lampu lalu lintas yang dapat berubah warna menjadi merah, hijau, atau kuning pada sebuah perimpangan.

- ***Class Crossroad***

  Class ini memiliki *constructor* yang berfungsi untuk menggabungkan dan menciptakan objek-objek yang berada dalam satu persimpangan, objek-objek tersebut antara lain empat buah objek sign dan empat buah objek carcntr. *Class* ini menyimpan koordinat titik pusat persimpangan dan variabel-variabel yang dibutuhkan dalam desain dan pengendalian lampu lalu lintas.

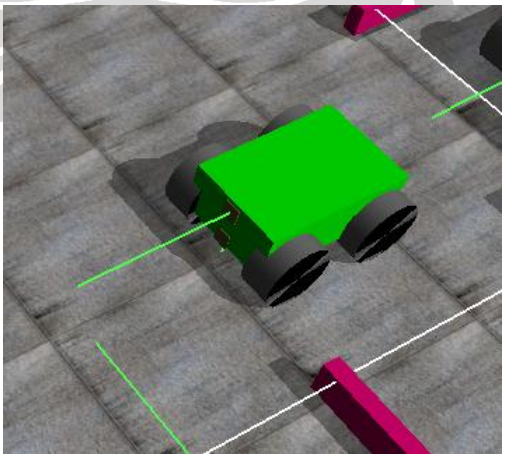
- ***Class Initcc***

  Class ini memiliki *constructor* yang berfungsi untuk menciptakan objek initcc di dunia ODE dan juga memiliki fungsi *draw* untuk menggambar objek *initcc* tersebut. Class ini menyimpan posisi koordinat objek *initcc* dan waktu masuk dan waktu keluar mobil dalam sistem. *Initcc* adalah sensor pada awal dan akhir ruas jalan yang berfungsi sebagai penghitung jumlah mobil yang masuk dan penghitung waktu masuk dan waktu keluar mobil dalam sistem lalu lintas.

5.1 Pemodelan Simulasi

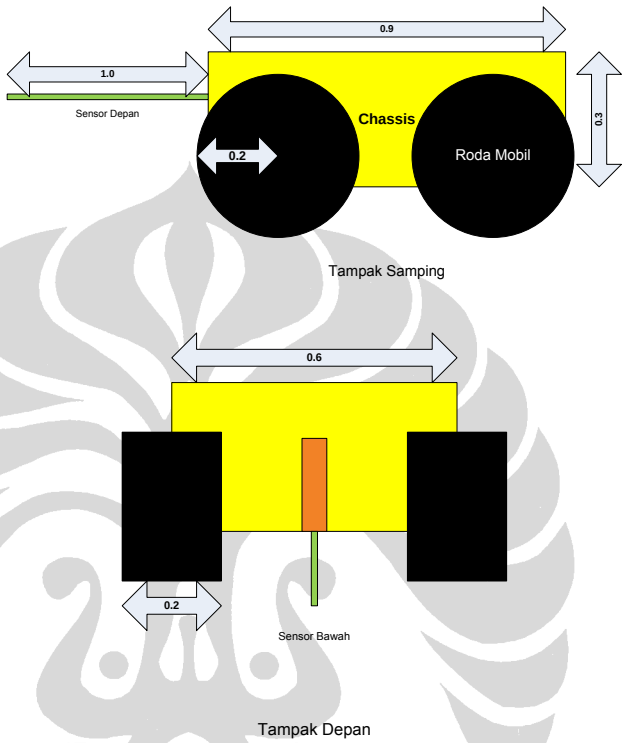
5.1.1 Mobil

Mobil dalam simulasi digambarkan sebagai sebuah kendaraan roda empat yang terdiri dari kerangka mobil yang berbentuk balok, empat buah roda yang berbentuk silinder serta dua buah sensor jarak yang terletak di bagian depan dan bawah mobil. Mobil merupakan suatu objek yang penting dalam simulasi karena nantinya keefektifan dari algoritma yang dipakai akan terlihat dari lama perjalanan mobil dalam simulasi.



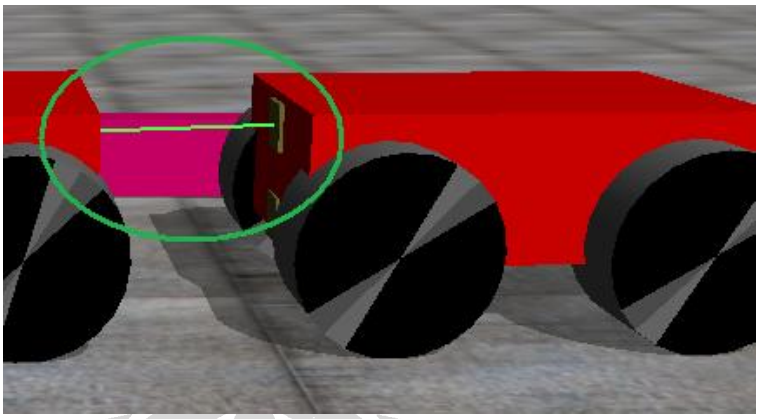
Gambar 5.12 *Screenshot* mobil pada simulasi

Berikut ini adalah tampak samping dan tampak depan dari mobil yang akan disimulasikan berikut dengan detil ukuran setiap komponen pembentuk mobil itu sendiri:



Gambar 5.13 Tampak samping dan tampak depan mobil

Dalam simulasi mobil dibuat untuk dapat mengenali lingkungan sekitarnya melalui dua sensor yang ia miliki. Sensor yang berada di bawah mobil berguna untuk mendeteksi lokasi persimpangan sedangkan sensor yang berada di depan mobil berguna untuk menghindari tabrakan antar sesama mobil. *Pseudocode* dari metode penghindaran tabrakan antar mobil dapat dilihat pada gambar 5.5. Metode penghindaran tabrakan antar mobil dan mengecek warna lampu persimpangan berada di dalam *class* nearCallback. *Class* nearCallback merupakan sebuah kelas khusus dalam *Open Dynamic Engine* yang berfungsi untuk menentukan relasi antara dua objek. *Class* nearCallback akan dipanggil apabila ada dua objek yang saling bertumbukan atau bersentuhan.

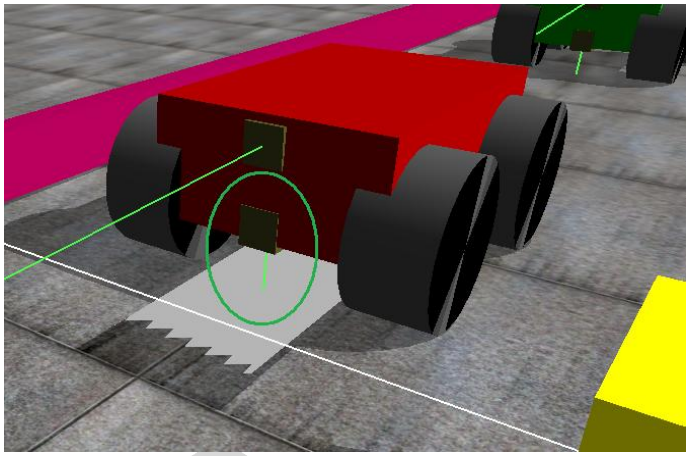


Gambar 5.4 *Screenshot* sensor pendeteksi mobil

```
if car's front sensor collide
        if distance 0.9 until 1.0
                Car run;
        else if distance less than 0.9
```

Gambar 5.5 *Pseudocode* metode penghindaran tabrakan

Sensor pada bawah mobil berfungsi untuk menangkap objek *sign* yang berada pada setiap persimpangan, dimana objek *sign* merupakan penanda bahwa mobil tersebut telah berada dekat dengan persimpangan. Apabila sensor mobil mendeteksi objek *sign*, ia harus mengecek warna lampu lalu lintas pada persimpangan tersebut untuk menentukan perilaku mobil terhadap persimpangan, apakah tetap berjalan ataukah berhenti. *Pseudocode* dari metode mobil dalam mengecek warna lampu di persimpangan dapat dilihat pada gambar 5.7.

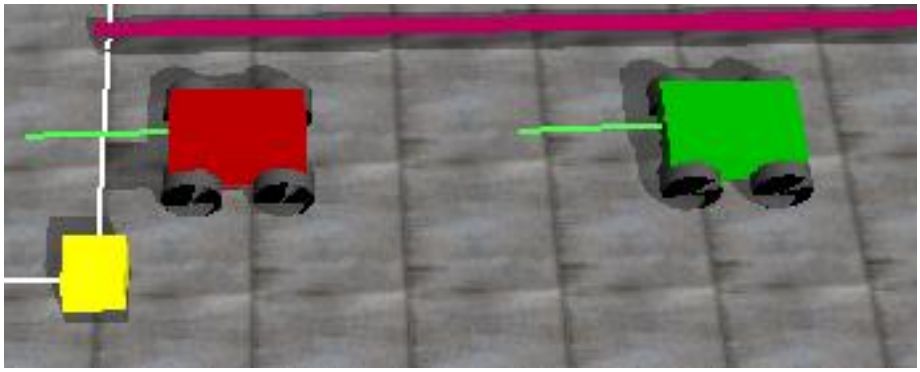


Gambar 5.6 *Screenshot* sensor mobil bagian bawah

```
if car's under sensor collide with sign
        if traffic light color=green
                Car run;
        else if traffic light color=yellow
                Car slow down; //until stop later
```

Gambar 5.7 *Pseudocode* metode pengecekan warna lampu lalu lintas

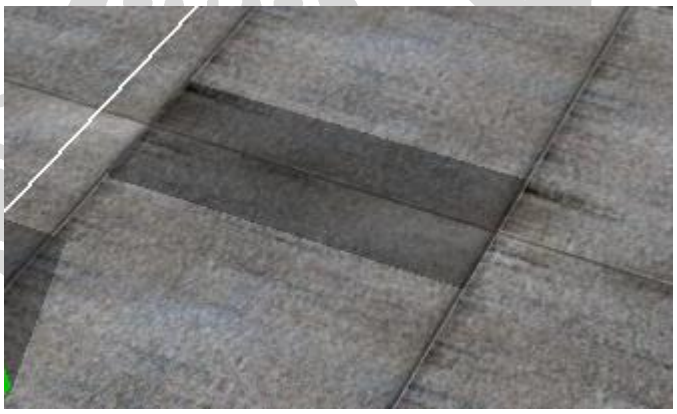
Untuk mempermudah pengamatan kecepatan mobil dalam simulasi, setiap mobil dapat berubah warna sesuai dengan kecepatan dia saat itu. Warna mobil merah mengrepresentasikan kecepatan mobil pada saat 0 m/s dan warna mobil hijau mengrepresentasikan kecepatan mobil pada saat kecepatan maksimal yaitu 10 m/s.



Gambar 5.8 Perbedaan warna mobil saat berhenti dan berjalan

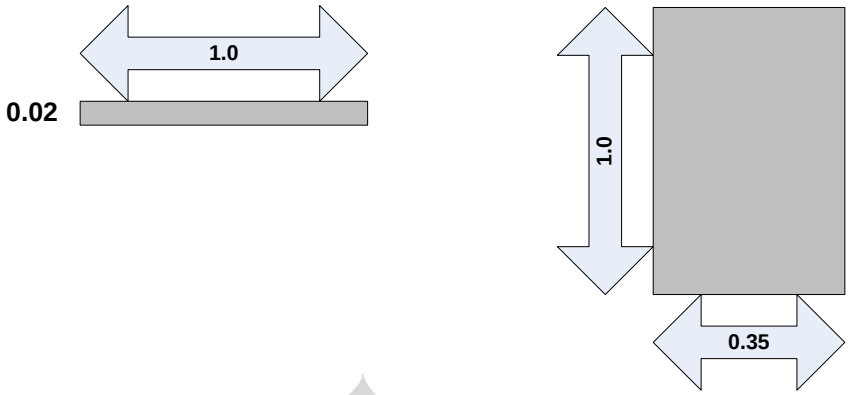
5.1.2 Sign

Sign dalam simulasi digambarkan sebagai sebuah balok tipis yang diletakan sebelum persimpangan. *Sign* berfungsi sebagai penanda bagi mobil bahwa ia sudah berada dekat dengan persimpangan, dimana *sign* ini akan dideteksi oleh sensor yang berada di bawah mobil.



Gambar 5.9 *Screenshot* dari objek *sign*

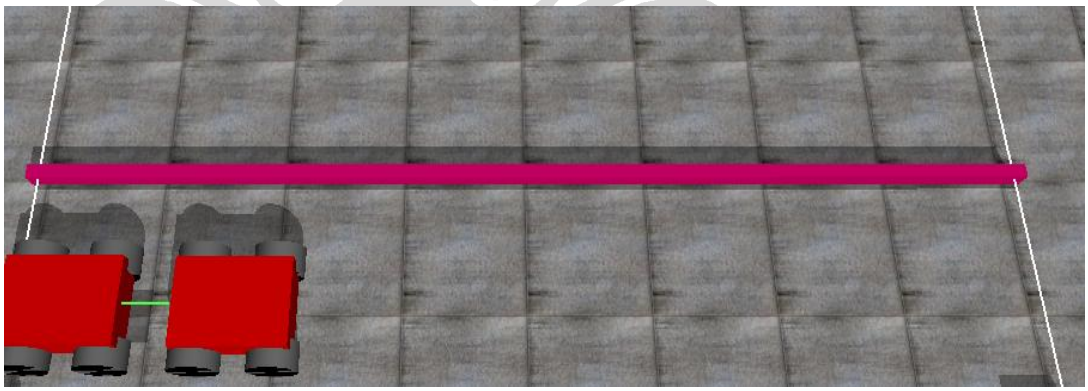
Berikut ini adalah tampak samping dan tampak atas dari *sign* yang akan disimulasikan berikut dengan detil ukuran dari objek *sign*:



Gambar 5.10 Tampak samping dan tampak depan dari desain *sign*

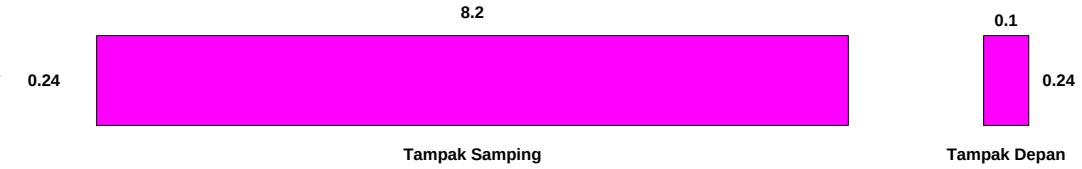
5.1.3 Border

Border atau pembatas jalan digambarkan sebagai sebuah balok panjang yang membagi ruas jalan menjadi dua bagian. *Border* berfungsi untuk membatasi jalur kiri dan kanan pada suatu ruas jalan, sehingga pengamatan untuk masing-masing ruas jalan yang dilalui mobil menjadi lebih mudah..



Gambar 5.11 *Screenshot* dari objek *border*

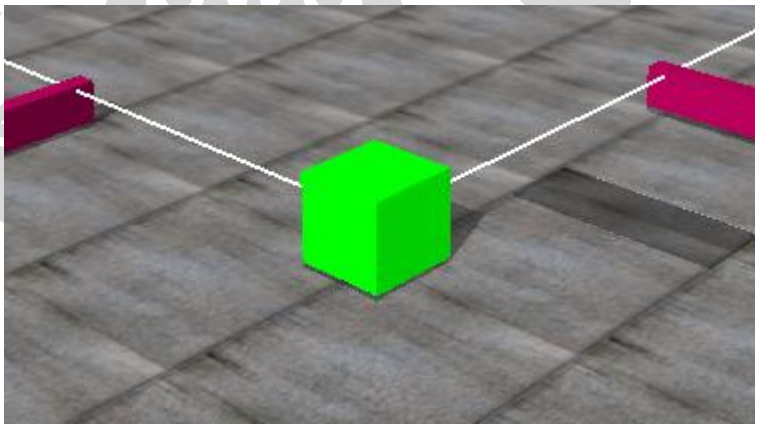
Pada Gambar 5.12 akan ditampilkan tampak tiga dimensi dari *border* yang akan disimulasikan berikut dengan detil ukuran dari objek *border*:



Gambar 5.12 Tampak samping dan tampak depan dari desain *border*

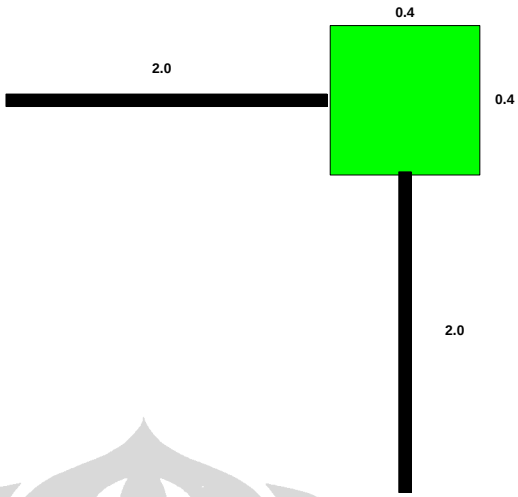
5.1.4 Carcntr

Carcntr atau sensor penghitung jumlah mobil digambarkan sebagai sebuah balok yang memiliki dua buah sensor. Kedua sensor ini berfungsi untuk menghitung jumlah kendaraan yang keluar atau masuk ke dalam ruas jalan. Carcntr juga berfungsi sebagai lampu lalu lintas pada persimpangan, yang dapat berubah warna menjadi warna hijau, merah, atau kuning.



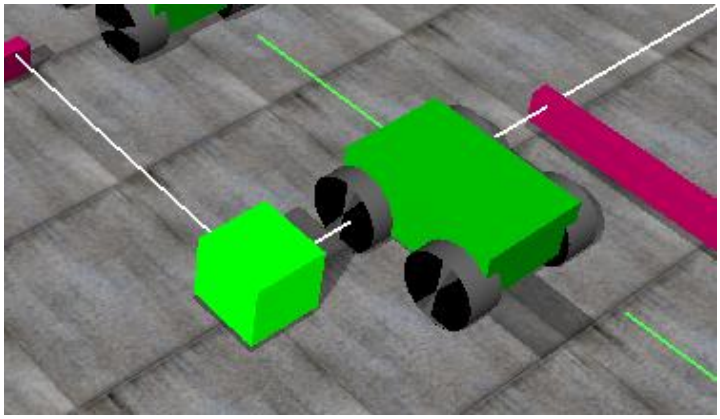
Gambar 5.13 *Screenshot* objek carcntr

Pada Gambar 5.14 akan ditampilkan tampak atas dari objek carcntr yang akan disimulasikan berikut dengan detil ukurannya:



Gambar 5.14 Tampak atas dari desain Carcntr

Dua buah sensor pada bagian samping Carcntr berfungsi untuk mendeteksi mobil yang melewati Carcntr. Apabila ada mobil yang menyentuh salah satu sensor yan g dimiliki oleh objek Carcntr maka Carcntr akan otomatis menambahkan atau mengurangi nilai dari variabel jumlah kendaraan yang dimiliki oleh persimpangan tempatnya berada lalu sensor Carcntr akan menunggu sampai badan mobil tidak mengenai sensor lagi untuk dapat mendeteksi mobil selanjutnya. *Pseudocode* dari metode Carcntr dalam mengecek mobil yang lewat di persimpangan dapat dilihat pada gambar 5.16. Metode Carcntr dalam mengecek mobil yang lewat berada di dalam *class* nearCallback. *Class* nearCallback merupakan sebuah kelas khusus dalam *Open Dynamic Engine* yang berfungsi untuk menentukan relasi antara dua objek. *Class* nearCallback akan dipanggil apabila ada dua objek yang saling bertumbukan atau bersentuhan.



Gambar 5.15 *Screenshot* mobil saat melewati sensor carcntr

```
for i <-- 0 to total crossroad - 1
        if Carcntr[0] sensor1 is collide
                if car detected and flag is 0
                        crossroad[i+1] num of car from left +=1
                        flag=1
                        return()
                else if car detected and flag is 0
                        return()//do nothing
                else if car not detected and flag is 1
                        flag=0
                        return()
        if Carcntr[0] sensor2 is collide
                if car detected and flag is 0
                        crossroad[i] num of car from up -=1
```

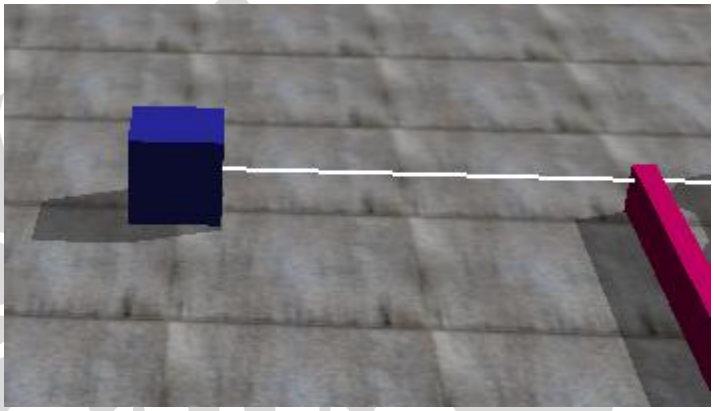
```
            else if car not detected and flag is 1
                    flag=0
                    return()
        if Carcntr[1] sensor2 is collide
             if car detected and flag is 0
                    crossroad[i-num of crossroad in 1 row] num of car
                    from bottom +=1
                    flag=1
                    return()
            else if car detected and flag is 0
                    return()//do nothing
            else if car not detected and flag is 1
                    flag=0
                    return()
        if Carcntr[2] sensor1 is collide
             if car detected and flag is 0
                    crossroad[i+1] num of car from right +=1
                    flag=1
                    return()
            else if car detected and flag is 0
```

```
else if car detected and flag is 0
        return()//do nothing
    else if car not detected and flag is 1
        flag=0
        return()
if Carcntr[3] sensor2 is collide
    if car detected and flag is 0
        crossroad[i+ num of crossroad in 1 row] num of
        car from up +=1
```

Gambar 5.16 *Pseudocode* metode carcntr dalam mengecek mobil

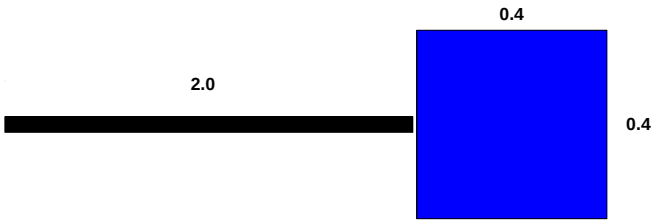
5.1.5 Init

Initcc digambarkan sebagai sebuah balok yang memiliki satu buah sensor. Sensor pada initcc berfungsi sebagai penghitung jumlah mobil yang masuk dan keluar dari sistem lalu lintas dan berfungsi juga sebagai penghitung lama waktu mobil berada dalam sistem lalu lintas, yang nantinya data tersebut akan menjadi ukuran keefektifan algoritma yang digunakan.



Gambar 5.17 *Screenshot* dari objek Initcc

Pada Gambar 5.18 akan ditampilkan tampak atas dari initcc yang akan disimulasikan berikut dengan detil ukuran yang dimiliki objek initcc:



Gambar 5.18 Tampak atas dari desain Initcc

5.2 Pemodelan Algoritma Pada Simulasi

Setelah memodelkan lingkungan serta objek-objek pada simulasi, selanjutnya algoritma yang sudah dijabarkan pada bab 3 Desain dan Pengendalian Sistem Lampu Lalu Lintas dimasukan ke dalam program simulasi untuk mengatur nyala lampu lalu lintas. Sebagian besar perhitungan algoritma terdapat pada kelas simloop. Kelas simloop merupakan kelas khusus dalam *Open Dynamics Engine* yang berfungsi untuk melakukan penggambaran objek setiap iterasinya. Pada subsubbab berikut ini akan dijelaskan urutan perhitungan rumus sampai didapatkan lama waktu lampu merah dan waktu lampu hijau yang diinginkan.

5.2.1 Perhitungan ρ_{ij} , ε_{ij} , dan ϕ_i

Variabel ρ_{ij} , ε_{ij} , dan ϕ_i adalah variabel-variabel pertama yang dicari sebelum mencari variabel lainnya dimana ketiga variabel ini dimiliki oleh kelas crossroad. ρ_{ij} adalah variabel yang menyatakan kepadatan dari jalan suatu persimpangan dengan persimpangan-persimpangan tetangganya. Karena dalam simulasi ini setiap persimpangan selalu memiliki empat persmpangan tetangga maka nilai j dari ρ_{ij} selalu bernilai empat (konstan). Dalam program ρ_{ij} memiliki nama variabel rho[1], rho[2], rho[3], dan rho[4] yang memiliki tipe *double*. rho[1] menyatakan kepadatan dari jalan antara suatu persimpangan dengan persimpangan tetangga bagian kanan, rho[2] menyatakan kepadatan dari jalan antara suatu persimpangan dengan persimpangan tetangga bagian atas, rho[3] menyatakan kepadatan dari jalan antara suatu persimpangan dengan persimpangan tetangga bagian kiri, rho[4] menyatakan kepadatan dari jalan antara suatu persimpangan dengan persimpangan tetangga bagian bawah.

ϕ_i adalah variabel yang menyatakan besar fase *timer* yang dimiliki suatu persimpangan. Dalam program ϕ_i memiliki nama variabel phi yang memiliki tipe *double*.

```
for i <-- 0 to total crossroad – 1
        crossroad[i]->psi=(seconds in crossroad's timer/ crossroad's
        cycle time)*360
        for j <--1 to 4
```

Gambar 5.19 *Pseudocode* mencari nilai ϕ_i

Variabel v[i] adalah jumlah kendaraan pada ruas jalan antara i dan j yang nilainya berasal dari pendeteksian kendaraan yang dilakukan oleh sensor pada objek Carcntr dan Initcc. Nilai "6" adalah jumlah maksimal kendaraan yang dapat ditampung oleh ruas jalan. Lama waktu *cycle time* adalah total lama waktu lampu merah untuk jalan pada ruas vertikal, lama waktu lampu merah untuk jalan pada ruas horisontal dan lama waktu lampu kuning untuk jalan pada ruas horisontal dan vertikal.

ε_{ij} adalah variabel yang menyatakan besar aliran kendaraan yang menuju suatu persimpangan dari persimpangan-persimpangan tetangganya. Sama seperti sebelumnya karena dalam simulasi ini setiap persimpangan selalu memiliki empat persmpangan tetangga maka nilai j dari ε_{ij} selalu bernilai empat (konstan). Dalam program ε_{ij} memiliki nama variabel epsilon[1], epsilon[2], epsilon[3], dan rhepsilono[4] yang memiliki tipe *double.* epsilon[1] menyatakan besar aliran kendaraan dari persimpangan tetangga bagian kanan, epsilon[2] menyatakan besar aliran kendaraan dari tetangga bagian atas, epsilon[3] menyatakan besar aliran

kendaraan dari tetangga bagian kiri, epsilon[4] menyatakan besar aliran kendaraan dari tetangga bagian bawah.

```
for each sample time
        for i <-- 0 to total crossroad – 1
                for j <--1 to 4
                        crossroad[i]->epsilon[j]=(1/(6*sample
                        time))*(crossroad[i]->rho[i])
                crossroad[i]->rho[i]=0.0;
                end
        end
end
```

Gambar 5.20 *Pseudocode* mencari nilai ε_{ij}

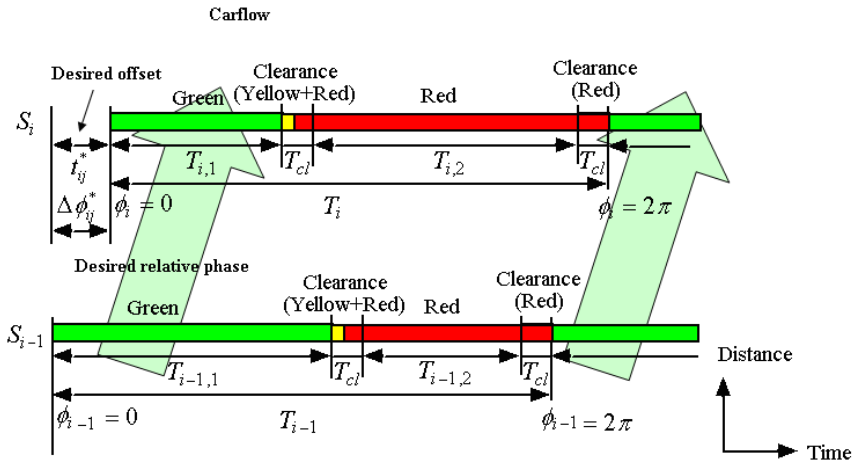
5.2.2 Perhitungan $\Delta\phi_{ij}$ dan $\tilde{\sigma}_i$

Setelah nilai dari ρ_{ij} , ε_{ij} , dan ϕ_i didapatkan, selanjutnya akan dicari nilai dari $\Delta\phi_{ij}$ dan nilai dari $\tilde{\sigma}_i$. Nilai dari $\Delta\phi_{ij}$ dan $\tilde{\sigma}_i$ dihitung setiap kali suatu persimpangan selesai menjalankan satu *cycle time*. $\Delta\phi_{ij}$ adalah variabel yang menyatakan selisih antara nilai fase suatu persimpangan dengan nilai fase persimpangan tetangga-tetangganya. Nilai j dari $\Delta\phi_{ij}$ berkisar antara satu sampai dengan empat, karena setiap persimpangan pasti memiliki empat buah persimpangan tetangga. Pada program nilai $\Delta\phi_{ij}$ diberi nama variabel deltaPhi[1], deltaPhi[2], deltaPhi[3], dan deltaPhi[4] yang memiliki tipe *double*. deltaPhi[1] menyatakan besar selisih fase antara suatu persimpangan dengan persimpangan tetangga bagian kanan, deltaPhi[2] menyatakan besar selisih fase antara suatu persimpangan dengan persimpangan tetangga bagian atas, deltaPhi[3] menyatakan

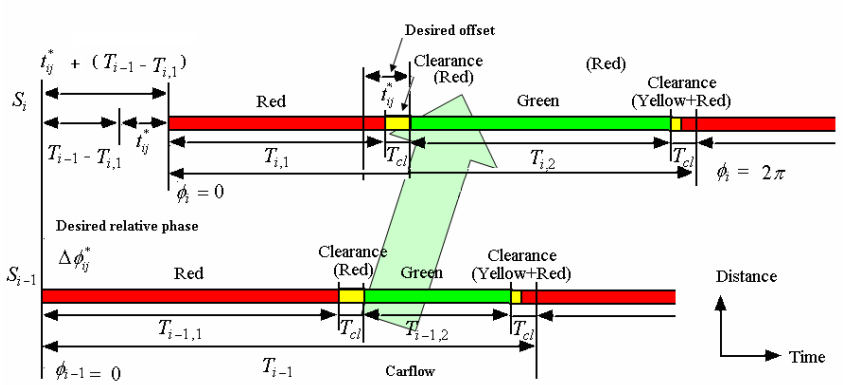
besar selisih fase antara suatu persimpangan dengan persimpangan tetangga bagian kiri, deltaPhi[4] menyatakan besar selisih fase antara suatu persimpangan dengan persimpangan tetangga bagian bawah. Untuk persimpangan yang memiliki tetangga kurang dari empat karena berada di bagian pinggir atau pojok, nilai dari deltaPhi diasumsikan sebesar nilai phi dari persimpangan itu sendiri.

5.2.3 Perhitungan $\Delta\phi_{ij}^{*}$

Setelah nilai dari $\Delta\phi_{ij}$ didapatkan, selanjutnya dapat dicari nilai dari $\Delta\phi_{ij}^{*}$. $\Delta\phi_{ij}^{*}$ menyatakan perbedaan *offset*/fase yang diinginkan agar suatu mobil setelah mendapatkan lampu hijau di suatu persimpangan cenderung akan mendapatkan lampu hijau untuk persimpangan berikutnya. Nilai dari $\Delta\phi_{ij}^{*}$ memiliki perbedaan rumus untuk arah prioritas utama dan arah prioritas kedua. Dalam program simulasi ini arah vertikal menjadi arah utama dan arah horisontal menjadi arah dengan prioritas kedua.



Gambar 5.21 Ilustrasi $\Delta\phi_{ij}^{*}$ dalam *cycle time* untuk arah vertikal



Gambar 5.22 Ilustrasi $\Delta\phi_{ij}^{*}$ dalam *cycle time* untuk arah horisontal

Gambar 5.21 menjelaskan bahwa untuk arah vertikal (arah utama) dimana apabila pada persimpangan sebelumnya sebuah mobil mendapatkan lampu hijau setelah dia berjalan dan sampai pada persimpangan selanjutnya dia akan mendapatkan lampu hijau lagi. Sama halnya dengan gambar 5.22 , untuk arah horisontal (arah prioritas kedua) dimana apabila pada persimpangan sebelumnya sebuah mobil mendapat lampu hijau maka setelah dia berjalan dan sampai pada persimpangan selanjutnya dia akan mendapatkan lampu hijau lagi, tetapi perbedaannya selisih *offset*/fase pada arah horisontal menambahkan selisih waktu lampu merah sehingga selisih *offset* pada arah horisontal menjadi lebih tepat dan sesuai. Pada program nilai $\Delta\phi_{ij}^{*}$ diberi nama variabel deltaPhiStar[1], deltaPhiStar[2], deltaPhiStar[3], dan deltaPhiStar[4] yang memiliki tipe *double*.

5.2.4 Perhitungan $\Delta\bar{\phi}_{i}^{*}$

Setelah nilai dari $\Delta\phi_{ij}^{*}$ didapatkan, selanjutnya dapat dicari nilai dari $\Delta\bar{\phi}_{i}^{*}$. $\Delta\bar{\phi}_{i}^{*}$ menyatakan fase relatif yang diinginkan dalam suatu persimpangan. Dalam program nilai $\Delta\bar{\phi}_{i}^{*}$ diberi nama variabel deltaPhiStarI yang memiliki tipe *double*.

5.2.5 Perhitungan ω_i^* dan T_i^*

Setelah nilai dari $\Delta\bar{\phi}_i^*$ diperoleh, selanjutnya dapat dicari nilai dari ω_i^*. ω_i^* menyatakan frekuensi alami yang diinginkan dari *oscillator* S_i. Karena ω_i^* merupakan frekuensi alami yang diinginkan, kita dapat mencari periode *cycle time* yang diinginkan dengan menggunakan rumus sebagai berikut:

$$T_i^* = \frac{2\pi}{\omega_i *} \qquad \textbf{(5.1)}$$

Dimana nilai dari T_i^* dibutuhkan untuk mengatur *split setting* dari lampu lalu lintas. Dalam program simulasi nilai ω_i^* diberi nama variabel omegaStar yang memiliki tipe *double* dan nilai T_i^* diberikan nama variabel tStar dengan tipe *double*.

5.2.6 Perhitungan T_{ik}^* dan T_{ik}

Setelah nilai dari T_i^* diperoleh, selanjutnya dapat dicari nilai dari T_{ik}^*. T_{ik}^* menyatakan *split time* yang diinginkan untuk arus kendaraan yang masuk dari arah horizontal dan vertikal, dimana untuk k=1 menyatakan arus kendaraan vertikal dan untuk k=2 menyatakan arus kendaraan horisontal. Setelah nilai T_{i1}^* dan T_{i2}^* diperoleh, selanjutnya dicari nilai T_{ik} untuk iterasi selanjutnya. T_{i1} menyatakan lamanya lampu hijau untuk arah vertikal (lama waktu lampu merah bagi arah horisontal) dan T_{i2} menyatakan lamanya lampu hijau untuk arah horisontal (lama waktu merah bagi arah vertikal). Dalam mencari nilai T_{ik} dibutuhkan nilai γ yang merupakan suatu konstatnta perubahan yang mengatur seberapa cepat perubahan *split time* menuju *split time* yang diinginkan. Dalam program nilai γ diberi nilai 0.2.

BAB 6
ANALISIS HASIL SIMULASI

Pada bab ini akan ditampilkan hasil percobaan yang dilakukan dalam simulasi dengan menggunakan algoritma sinkronisasi kuramoto dan algoritma *random* beserta dengan analisis untuk masing-masing hasil. Pembahasan pada bab ini mencakup parameter-parameter yang digunakan dalam percobaan, perbedaan waktu rata-rata suatu mobil berada dalam sistem untuk metode sinkronisasi kuramoto dan metode *random* serta perbedaan waktu rata-rata mobil berdasarkan urutan masuk ke dalam sistem untuk metode sinkronisasi kuramoto dan metode *random*.

6.1 Parameter Percobaan

Untuk menghasilkan percobaan yang mengrepresentasikan seluruh kemungkinan dalam keadaan nyata dibutuhkan parameter-parameter percobaan yang sesuai. Beberapa parameter yang digunakan sebagai variasi percobaan adalah sebagai berikut:

- **Jumlah Persimpangan**

  Seperti yang telah dijelaskan pada bab sebelumnya, dalam simulasi ini persimpangan digambarkan dalam bentuk *grid* dengan panjang dan lebar yang sama. Percobaan dilaksanakan dalam dua parameter jumlah persimpangan yang berbeda, yaitu persimpangan yang berbentuk *grid* 5X5 (25 buah persimpangan) dan persimpangan yang berbentuk *grid* 7X7 (49 buah persimpangan). Jumlah persimpangan yang dibuat dalam simulasi hanya dibatasi sampai 7x7 saja karena untuk jumlah persimpangan *grid* yang lebih besar, simulasi akan berjalan sangat lambat.

- **Jumlah Kendaraan**

  Percobaan dilaksanakan dalam dua parameter jumlah mobil per-ruas jalan yang berbeda , yaitu enam buah mobil untuk masing-masing ruas jalan dan

sepuluh buah mobil untuk masing-masing ruas jalan. Pembuatan kendaraan dalam simulasi mungkin terlihat masih sedikit, hal ini karena semakin banyaknya kendaraan yang diciptakan, simulasi akan berjalan semakin lambat dan simulasi akan menjadi semakin tidak stabil. Ketidakstabilan simulasi dapat menyebabkan jalan mobil menjadi tidak lurus, sehingga menyebabkan tabrakan dan kegagalan simulasi

- **Algoritma Pengaturan Lampu Lalu Lintas**
  Percobaan dilaksanakan dengan menggunakan dua algoritma pengaturan lampu lalu lintas yang berbeda, yaitu algoritma sinkronisasi Kuramoto dan algoritma *random*.

6.2 Hasil Percobaan Simulasi

Percobaan simulasi pada persimpangan berbentuk *grid* 5x5 , *grid* 7x7 dan *grid* 10x10 masing-masing dilakukan dalam empat kali percobaan dengan parameter sebagai berikut:

- Simulasi dengan menggunakan algoritma sinkronisasi kuramoto dengan jumlah mobil sedang (enam mobil setiap ruas jalan).
- Simulasi dengan menggunakan algoritma *random* dengan jumlah mobil sedang (enam mobil setiap ruas jalan).
- Simulasi dengan menggunakan algoritma sinkronisasi kuramoto dengan jumlah mobil padat (sepuluh mobil setiap ruas jalan).
- Simulasi dengan menggunakan algoritma *random* dengan jumlah mobil sedang (sepuluh mobil setiap ruas jalan).

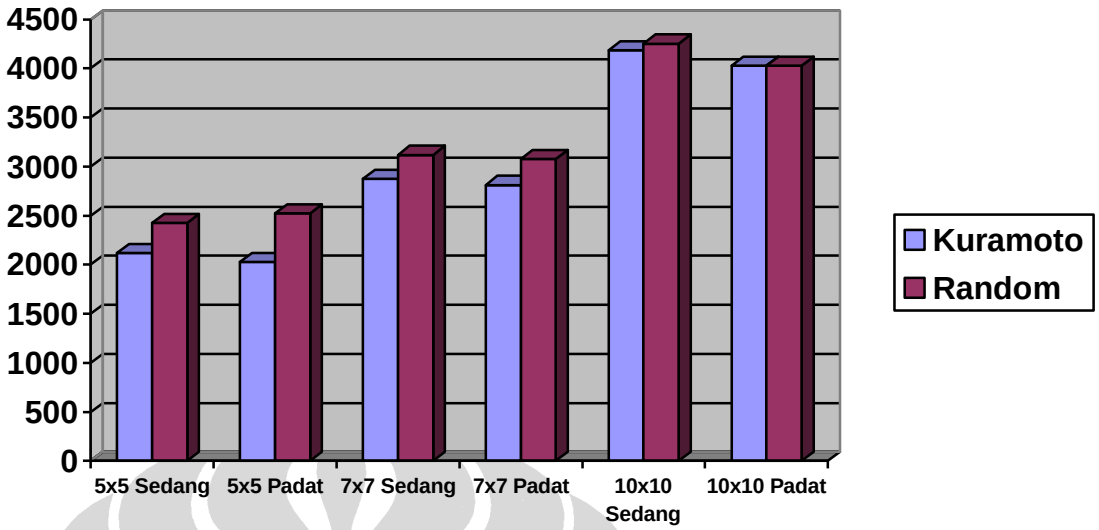
Dari setiap percobaan yang dilakukan dalam simulasi didapatkan nilai lama waktu setiap mobil di dalam sistem simulasi lalu lintas, dari nilai-nilai tersebut nantinya akan diolah untuk mencari waktu rata-rata dari seluruh mobil di dalam sistem . Dari delapan kali percobaan di atas didapatkan hasil analisis sebagai berikut:

- Pada percobaan di persimpangan *grid* 5x5, persimpangan *grid* 7x7 dan di persimpangan *grid* 10x10 dengan keadaan jumlah mobil sedang maupun padat, waktu tempuh rata-rata mobil pada simulasi yang menggunakan algoritma sinkronisasi kuramoto lebih kecil dibandingkan dengan simulasi yang menggunakan algoritma sinkronisasi *random*.

Berikut ini adalah hasil dari percobaan simulasi yang direpresentasikan dalam bentuk tabel dan grafik:

Tabel 6.1 Tabel waktu rata-rata mobil untuk menyelesaikan simulasi

Algoritma	**Persimpangan 5 x 5**		**Persimpangan 7 x 7**		**Persimpangan 10 x 10**	
	Sedang	**Padat**	**Sedang**	**Padat**	**Sedang**	**Padat**
Kuramoto	2121 detik	2028 detik	2876 detik	2814 detik	4183 detik	4028 detik
Random	2426 detik	2521 detik	3120 detik	3075 detik	4253 detik	4030 detik



Gambar 6.1 Grafik percobaan simulasi