BAB IV

RANCANGAN SIMULASI

Untuk memodelkan algoritma yang telah dijabarkan pada bab sebelumnya, penulis menggunakan simulasi untuk melihat apakah algoritma tersebut dapat diterapkan dengan baik. Rancangan simulasi ini berfungsi sebagai media uji coba sebelum algoritma ini diterapkan pada dunia nyata.

Dalam simulasi ini, penulis membuat beberapa *class*, yaitu:

1. *Class* LaluLintas

Pada *class* ini terdapat *constructor* yang berfungsi untuk menyiapkan objek jalur lalu lintas pada simulasi. Selain itu, *class* ini juga berfungsi untuk menggambar tampilan simulasi, mulai dari kendaraan yang akan melintas, letak lampu dan jalur lalu lintas, juga *array* Jalur untuk menampung jumlah kendaraan yang melintas pada jalur simulasi. *Class* ini juga melakukan fungsi *update* posisi kendaraan untuk melihat apakah ada kendaraan lain atau lampu lalu lintas didepannya.*Thread* utama dari simulasi ini adalah *method Run* yang berfungsi untuk meng-*update* dan menambahkan jumlah kendaraan. Setelah jumlah kendaraan ditambahkan, method ini lalu menggambar ulang seluruh tampilan simulasi.

2. *Class* ControlLampuLaluLintas

Class ini berfungsi untuk mengendalikan lampu lalu lintas dengan 3 *state* yang telah didefinisikan pada bab sebelumnya, yaitu *Cycle Time, Split Time* dan *Offset*. Dengan *state-state* tersebut, lampu lalu lintas dapat diset waktunya untuk berubah warna dari merah ke hijau dan juga sebaliknya.

3. *Class* ControlUpdater

Class ini menjalankan fungsi *sampling* yang berguna untuk menghitung banyaknya kendaraan dalam selang waktu tertentu. Dengan menggunakan data hasil *sampling*, *class* ini akan melakukan fungsi untuk meng-*update* nilai dari *Cycle Time, Split Time* dan *Offset*.

4. *Class* LampuLaluLintas

Class ini menyimpan keadaan lampu, apakah berwarna merah atau hijau. *Class* ini juga yang akan mengisi nilai *offset* pada tiap-tiap lampu lalu lintas pada waktu pertama kali dijalankan.

5. *Class* MathUtil

Class ini merupakan *class* tambahan yang menyimpan fungsi matematika yang tidak ditemukan pada *library* Java, seperti fungsi modulo untuk bilangan bertipe *double*.

6. *Class* Kendaraan

Class ini berfungsi untuk menyimpan state dari kendaraan. Selain itu, class ini juga memiliki fungsi untuk meng-update posisi kendaraan pada jalur simulasi.

7. *Class* Simulation

Class ini berfungsi untuk membedakan apakah simulasi dijalankan secara *random* atau menggunakan sinkronisasi.

8. *Class* Reporter

Class ini berfungsi untuk memberikan laporan hasi simulasi mengenai kecepatan rata-rata tiap kendaraan pada satu jalur.

Dengan *class-class* tersebut diatas, penulis memodelkan simulasi sebagai berikut:

IV.1 Pemodelan Kendaraan

Kendaraan dianggap sebagai sebuah titik. Untuk lebih kompatibel dengan dunia nyata, penulis membuat sebuah *constraint* untuk jarak antar kendaraan sebesar 4 m. Selain itu, *constraint-constraint* yang lain dapat dilihat sebagai berikut:

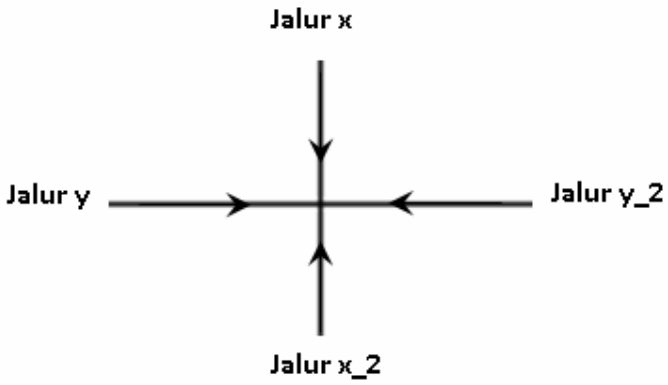
```
public class Kendaraan extends Thread
{
	static final double TIME_SLICE = 0.1; //dalam detik
	static final double V_MAX = 15; //dalam meter/detik
	static final double A_MAX = 4; //dalam meter/detik kuadrat

	double v_target;  //kecepatan/velocity (meter per detik)
	double v;         //kecepatan/velocity (meter per detik)
	double akselerasi; //percepatan (meter per detik kuadrat)
	double deselerasi; //perlambatan (meter per detik kuadrat)
	double x; //position dalam meter

	int last; //last lampu //bisa x atau y //tergantung konteks

	long startTime;


	public Kendaraan(double x, int last)
		{
			this.x = x;
			v=10;
			akselerasi=2;
			deselerasi=-5;
			this.last = last;
			startTime = System.currentTimeMillis();
		}
```

Pemodelan kendaraan dilakukan bersamaan dengan pembuatan jalur. Ketika jalur dibuat, kendaraan pun mulai ditambahkan pada simulasi. Penulis membuat jeda yang cukup jauh antara persimpangan pertama dengan tempat kendaraan pertama

kali muncul supaya kecepatannya tidak menjadi bias jika ada lampu merah pada persimpangan pertama tersebut.

```
for(int i=1;i<jalur_x.length-1;i++)
{
      if(Math.random()<Kendaraan.TIME_SLICE*Q_X)
            jalur_x[i].add(new Kendaraan(-JARAK_JEDA,0));
}
for(int i=1;i<jalur_x2.length-1;i++)
{
      if(Math.random()<Kendaraan.TIME_SLICE*Q_X2)
            jalur_x2[i].add(new Kendaraan(-JARAK_JEDA,0));
}
for(int i=1;i<jalur_y.length-1;i++)
{
      if(Math.random()<Kendaraan.TIME_SLICE*Q_Y)
            jalur_y[i].add(new Kendaraan(-JARAK_JEDA,0));
}
for(int i=1;i<jalur_y2.length-1;i++)
{
      if(Math.random()<Kendaraan.TIME_SLICE*Q_Y2)
            jalur_y2[i].add(new Kendaraan(-JARAK_JEDA,0));
}
```

Keterangan:



Gambar 6. Deskripsi Nama Jalur Pada Simulasi

- ✓ **Jalur_x** : Jalur kendaraan dari bagian atas yang menuju ke bagian bawah.
- ✓ **Jalur_x2** : Jalur kendaraan dari bagian bawah yang menuju ke bagian atas.

- ✓ **Jalur_y** : Jalur kendaraan dari bagian kiri yang menuju ke bagian kanan.
- ✓ **Jalur_y2** : Jalur kendaraan dari bagian kanan yang menuju ke bagian kiri.

Kecepatan kendaraan juga dipengaruhi oleh kecepatan kendaraan yang ada di depannya. Posisi kendaraan akan di-*update* berdasarkan posisi saat ini dan kecepatan dari laju kendaraan seperti berikut:

```
public void tick(double d)
{
      if(d>(V_MAX*V_MAX)/(2*A_MAX))
      {
            v_target = V_MAX;
      }

      else
      {
            v_target = Math.sqrt(2*A_MAX*d);
            if(Double.isNaN(v) )
            {
                  v_target=0;
            }
      }

      if(v>v_target)  //deselerasi
      {
            double xnext = x + v*TIME_SLICE +
                        0.5*deselerasi*TIME_SLICE*TIME_SLICE;
            if (xnext>x+d)
            {
                  xnext=x+d;
            }
            x=xnext;

            v = v + deselerasi*TIME_SLICE;
            if(v<v_target)
                  v = v_target;
      }
      else if(v<v_target) //akselerasi
      {
```

```
            double xnext = x + v*TIME_SLICE +
                        0.5*akselerasi*TIME_SLICE*TIME_SLICE;
            if (xnext>x+d)
            {
                  xnext=x+d;
            }
            x=xnext;
            v = v + akselerasi*TIME_SLICE;

            if(v>v_target)
                  v = v_target;
      }
      else  //tetap
      {
            double xnext = x + v*TIME_SLICE;
            if (xnext>x+d)
            {
                  xnext=x+d;
            }
            x=xnext;
      }
}
```

Pada setiap saat, kendaraan perlu untuk mengetahui lampu lalu lintas mana yang ada di depan dia. Untuk jalur x, kendaraan dapat mengetahui posisi y dari lampu lalu lintas yang bisa didapatkan dari posisi kendaraan dengan cara sebagai berikut:

```
lampu_x = i;
lampu_y = (int) Math.floor(currentKendaraan.x/GRID_Y);
                              //cari posisi Kendaraan yg didepan nya

lampu_y_depan = lampu_y + 1;
```

Selain itu, jika lampu lalu lintas berwarna merah maka simulasi akan memperlakukan lampu lalu lintas tersebut seperti kendaraan lain yang ada didepan kendaraan tersebut.

```
if(lll.lampu[lampu_x][lampu_y_depan]==LampuLaluLintas.JALUR_X)
{
      currentKendaraan.tick(Double.MAX_VALUE);
}
else
```

```
{
      currentKendaraan.tick(posisi-currentKendaraan.x-JARAK_JEDA);
}
currentKendaraan.tick(lastKendaraanposition-currentKendaraan.x-
      JARAK_JEDA);
```

Jika kendaraan sudah mencapai ujung dari jalurnya, maka kendaraan tersebut akan dihilangkan dari simulasi.

```
if(!jalur_x2[i].isEmpty() && jalur_x2[i].get(0).x > LEBAR_Y)
{
      Kendaraan hapus = jalur_x2[i].remove(0);
      lll.cLampu[i][hapus.last].jalur_x2_masuk();
      totalKecepatan+=LEBAR_X/(System.currentTimeMillis()
            - hapus.startTime);
      totalKendaraan++;
}
```

IV.2 Pemodelan Jalur

Pada pemodelan jalur, penulis hanya membuat suatu array yang menyimpan objek Kendaraan yang akan dikeluarkan pada selang waktu tertentu.

```
ArrayList<Kendaraan> jalur_x[] = new ArrayList[SIZE_X];
ArrayList<Kendaraan> jalur_x2[] = new ArrayList[SIZE_X];
ArrayList<Kendaraan> jalur_y[] = new ArrayList[SIZE_Y];
ArrayList<Kendaraan> jalur_y2[] = new ArrayList[SIZE_Y];
```

IV.3 Pemodelan Lampu Lalu Lintas

Penulis mendefinisikan pemodelan lampu lalu lintas sebagai berikut:

```
public class LampuLaluLintas
{
      public static final int JALUR_X = 0;
      public static final int JALUR_Y = 1;
      public static final int CLEARENCE_X = 2;
      public static final int CLEARENCE_Y = 3;
      int lampu[ ][ ] = new
            int[LaluLintas.SIZE_X][LaluLintas.SIZE_Y];
      ControlLampuLaluLintas cLampu[ ][ ] = new
```

```
      ControlLampuLaluLintas[LaluLintas.SIZE_X][LaluLintas.SIZE_Y];

      boolean random=false;

      LaluLintas parent;

      public LampuLaluLintas(LaluLintas parent, boolean random)
      {
            this.random = random;
            for(int i=0;i<LaluLintas.SIZE_X;i++)
            for(int j=0;j<LaluLintas.SIZE_Y;j++)
            {
                  lampu[i][j]=JALUR_X;
                  cLampu[i][j] = new ControlLampuLaluLintas
                        (this, lampu, i,j);
                  cLampu[i][j].start();
            }
if(!random)
{
      for(int i=0;i<LaluLintas.SIZE_X;i++)
      for(int j=0;j<LaluLintas.SIZE_Y;j++)
      {
            double t_star_h = LaluLintas.GRID_X/Kendaraan.V_MAX;
            double delta_phi_jalur_y_star = (t_star_h);

            if(i>0)
            {
                  cLampu[i][j].offset = cLampu[i-1][j].offset +
                              delta_phi_jalur_y_star;
            }
            else if(j>0)
            {
                  double t_star_v =  LaluLintas.GRID_Y/
                        Kendaraan.V_MAX;
                  double delta_phi_jalur_x_star = (t_star_v);
                  cLampu[i][j].offset = cLampu[i][j-1].offset +
                        delta_phi_jalur_x_star;
            }
      }
}
```

```
else
{
	for(int i=0;i<LaluLintas.SIZE_X;i++)
	for(int j=0;j<LaluLintas.SIZE_Y;j++)
	{
		cLampu[i][j].offset = Math.random()*
			cLampu[i][j].cycleTime;
	}
}
}
```

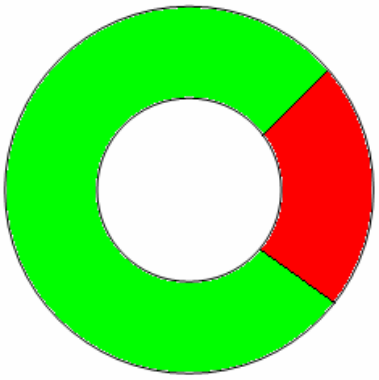
Pemodelan ini hanya digunakan untuk menentukan kendaraan dari arah mana yang dapat melintasi persimpangan jika lampu lalu lintas berubah warna.

Sedangkan fungsi yang digunakan untuk mengatur warna lampu lalu lintasnya adalah:

```
public void run()
{
	while(true)
	{
		if(offset > cycleTime)
		{
			offset = MathUtil.mod(cycleTime, offset);
		}

		if(offset> (cycleTime - splitTime))
		{
			lampu[a][b] = JALUR_Y;
			tidur(offset - (cycleTime - splitTime));
			lampu[a][b] = JALUR_X;
			tidur(cycleTime - splitTime);
			lampu[a][b] = JALUR_Y;
			if(cu.putToSleep)
			{
				indexCycleToUpdate++;

				if(indexCycleToUpdate==cycleToUpdate)
				{
					indexCycleToUpdate=0;
```

```
                                updateState();
                        }
                }
                tidur(cycleTime - offset);
        }
        else
        {
                lampu[a][b] = JALUR_X;
                tidur(offset);
                lampu[a][b] = JALUR_Y;
                if(cu.putToSleep)
                {
                        indexCycleToUpdate++;

                        if(indexCycleToUpdate==cycleToUpdate)
                        {
                                indexCycleToUpdate=0;
                                updateState();
                        }
                }
                tidur(splitTime);
                lampu[a][b] = JALUR_X;
                tidur(cycleTime-splitTime-offset);
        }
    }
}
```

Satu *cycle time* didefinisikan sebagai perpindahan warna lampu dari hijau ke merah lalu ke hijau lagi seperti gambar berikut ini:



Gambar 7. Satu *Cycle Time*

Nilai *split time* menyatakan berapa lama lampu hijau akan menyala. Sedangkan sisanya yaitu (nilai *cycle time* – nilai *split time*) merupakan lamanya lampu merah menyala. Perputaran ini akan berulang terus menerus.

Jika suatu *cycle time* direntangkan, maka akan menjadi gambar seperti ini:

Gambar 8. Satu *Cycle Time* Yang Direntangkan

Gambar tersebut mempunyai nilai *cycle time* dan *split time* yang sama dengan gambar berikut:

Gambar 9. Satu *Cycle Time* Dengan Nilai *Offset* Yang Berbeda

Perbedaan dari kedua gambar diatas hanya terletak pada nilai *offset*-nya yang menggeser lamanya lampu merah pada **Gambar 13**.

Untuk mendapatkan suatu simulasi yang optimal dan dapat menyesuaikan dengan keadaan, maka nilai dari *cycle time*, *split time* dan *offset* harus dapat berubah sesuai dengan keadaan saat itu.

IV.3.1 *PERUBAHAN SPLIT TIME*

Perubahan *split time* dilakukan dengan menggunakan method dari *class* ControlUpdater yang akan melakukan *sampling* seperti berikut:

```
parent.startSampling();
tidur(parent.cycleTime*2);
parent.endSampling();
```

Sampling ini digunakan untuk menghitung banyaknya kendaraan yang lewat dari masing-masing jalur pada persimpangan.

```
public void startSampling()
{
```

```
        jalur_x_counter=0;
        jalur_x2_counter=0;
        jalur_y_counter=0;
        jalur_y2_counter=0;
        isSampling = true;
}
public void endSampling()
{
        jalur_x  = jalur_x_counter;
        jalur_x2 = jalur_x2_counter;
        jalur_y  = jalur_y_counter;
        jalur_y2 = jalur_y2_counter;
        isSampling = false;
}
```

IV.3.2 *PERUBAHAN CYCLE TIME*

Dengan informasi *split time* tersebut dan tambahan informasi dari persimpangan disampingnya, maka lampu lalu lintas akan dapat menentukan *cycle time* yang selanjutnya.

```
public void updateCycle()
{
        double epsilon_x = (double)parent.jalur_x/(2*25);
        double epsilon_x2 = (double)parent.jalur_x2/(2*25);
        double epsilon_y = (double)parent.jalur_y/(2*25);
        double epsilon_y2 = (double)parent.jalur_y2/(2*25);


        double t_star_h = LaluLintas.GRID_X /
                    Kendaraan.V_MAX;
        double delta_phi_jalur_y_star = (t_star_h/
                    parent.cycleTime) - Math.floor
                    (t_star_h/parent.cycleTime);
        double delta_phi_jalur_y2_star = (t_star_h/
                    parent.cycleTime) - Math.floor
                    (t_star_h/parent.cycleTime);
```

```
double t_star_v = 4;
double delta_phi_jalur_x2_star = 2*Math.PI*
            (t_star_v/parent.cycleTime);
double delta_phi_jalur_x_star = 2*Math.PI*
            (t_star_v/parent.cycleTime);

double temp1,temp2;


temp1 = epsilon_x*Math.cos(delta_phi_jalur_x_star) +
         epsilon_y*Math.cos(delta_phi_jalur_y_star) +
         epsilon_x2*Math.cos(delta_phi_jalur_x2_star)
         +epsilon_y2*Math.cos(delta_phi_jalur_y2_star);
temp2 = epsilon_x*Math.sin(delta_phi_jalur_x_star) +
        epsilon_y*Math.sin(delta_phi_jalur_y_star) +
        epsilon_x2*Math.sin(delta_phi_jalur_x2_star) +
        epsilon_y2*Math.sin(delta_phi_jalur_y2_star);

double sigma_i_tilde_star = 1/4 * (Math.sqrt
      (Math.pow(temp1,2)+Math.pow(temp2,2)));

double delta_phi_i_bar_star = Math.atan(temp2/temp1);

double w_x = 2*Math.PI/parent.parent.cLampu
      [parent.a-1][parent.b].cycleTime;
double w_x2 = 2*Math.PI/parent.parent.cLampu
      [parent.a+1][parent.b].cycleTime;
double w_y = 2*Math.PI/parent.parent.cLampu
      [parent.a-1][parent.b].cycleTime;
double w_y2 = 2*Math.PI/parent.parent.cLampu
      [parent.a+1][parent.b].cycleTime;

double big_omega_i = (2*Math.PI/parent.cycleTime +
      (w_x*epsilon_x/epsilon_x2 + w_x2*epsilon_x2
      /epsilon_x + w_y*epsilon_y/epsilon_y2 +
      w_y2*epsilon_y2/epsilon_y)/4)/(1+(1/4)
      *(epsilon_x/epsilon_x2 + epsilon_x2/epsilon_x +
      epsilon_y/epsilon_y2 + epsilon_y2/epsilon_y));

double K = 0.1;
```

```
        double omega_i_star = big_omega_i - sigma_i_tilde_star*
              K*Math.sin(delta_phi_i_bar_star);

        double alpha=0.3;
        double beta=0.5;

        double omega_i =  2*Math.PI/parent.cycleTime;
        double omega_i_next = omega_i + alpha*(omega_i_star –
              omega_i) + beta*(0.628 - omega_i);

        if(!Double.isNaN(omega_i_next))
        {
              System.out.println("next = "+
                    (2*Math.PI/omega_i_next));
              parent.cycleTime = 2*Math.PI/omega_i_next;
        }
}
```

IV.3.3 *PERUBAHAN OFFSET*

Pertama kali, nilai *offset* dari setiap lampu lalu lintas akan di set sebagai berikut:

```
if(parent.a>0)
{
      parent.lastOffset = parent.offset;
      parent.offset = parent.parent.cLampu[parent.a-1]
      [parent.b].lastOffset + delta_phi_jalur_y_star;
}
else if(parent.b>0)
{
      double t_star_v = 4;
      double delta_phi_jalur_x_star = MathUtil.mod(2*Math.PI,
            2*Math.PI*((t_star_v)/parent.cycleTime));
      parent.lastOffset = parent.offset;
      parent.offset = parent.parent.cLampu[parent.a]
            [parent.b-1].lastOffset + delta_phi_jalur_x_star;
}
```

Perubahan dari *cycle time* dapat mempengaruhi nilai *offset* juga. Oleh karena itu jika nilai *offset* lebih dari nilai *cycle time*, maka nilai *offset* pun akan ikut berubah seperti berikut ini:

```
if(offset > cycleTime)
{
      offset = MathUtil.mod(offset, cycleTime);
}
```

Setelah ketiga parameter tersebut berhasil di-*update*, maka suatu saat akan tercapai titik keseimbangan dimana nilai dari ketiga parameter pada setiap persimpangan sudah saling bersinkronisasi dengan persimpangan disampingnya.

BAB V

ANALISIS HASIL SIMULASI

Setelah pembuatan simulasi selesai, penulis mencoba untuk menganalisa perbedaan kecepatan rata-rata setiap kendaraan mulai dari masuk dalam jalur simulasi sampai dengan keluar lagi di ujung jalur tersebut. Hasil yang didapat akan membandingkan antara sistem lalu lintas yang bersifat random dan tidak menggunakan sinkronisasi dengan sistem lalu lintas yang menggunakan sinkronisasi.

Untuk sistem lalu lintas yang random, penulis mendefinisikannya seperti berikut:

- √ Lama waktu lampu hijau dan lampu merah selalu tetap, yaitu 10 detik.
- √ Nilai *offset* dibuat acak.
- √ Tidak ada proses *update* nilai *cycle time.*

Dalam pengujian, penulis menggunakan skenario sebagai berikut:

1. Keadaan lalu lintas sepi dan seimbang. Jumlah kendaraan dari jalur x dan jalur y sama, yaitu 3 kendaraan per menit.
2. Keadaan lalu lintas padat dan seimbang. Jumlah kendaraan dari jalur x dan jalur y sama, yaitu 12 kendaraan per menit.
3. Keadaan lalu lintas sepi tapi tidak seimbang. Jumlah kendaraan dari jalur x sekitar 1 kendaraan per menit, sedangkan jumlah kendaraan dari jalur y sekitar 6 kendaraan per menit.
4. Keadaan lalu lintas padat tapi tidak seimbang. Jumlah kendaraan dari jalur x sekitar 9 kendaraan per menit, sedangkan jumlah kendaraan dari jalur y sekitar 18 kendaraan per menit.

Pengujian dilakukan sebanyak 10 kali dalam rentang waktu 1 pengujian selama 640 detik. Hasil dari pengujian tersebut selanjutnya dicari nilai rata-rata dan standar deviasinya.

Pada bab ini, penulis hanya memberikan hasil pengujian beserta analisa dari salah satu uji coba yang telah dilakukan. Grafik dan tabel kecepatan rata-rata kendaraan dan standar deviasi pada keseluruhan pengujian penulis letakkan pada lampiran.

Keterangan jalur pada simulasi:

- ✓ Jalur x : Jalur kendaraan dari bagian atas yang menuju ke bagian bawah.
- ✓ Jalur x2 : Jalur kendaraan dari bagian bawah yang menuju ke bagian atas.
- ✓ Jalur y : Jalur kendaraan dari bagian kiri yang menuju ke bagian kanan.
- ✓ Jalur y2 : Jalur kendaraan dari bagian kanan yang menuju ke bagian kiri.

V.1 Hasil Analisa Kondisi Sepi Dan Seimbang

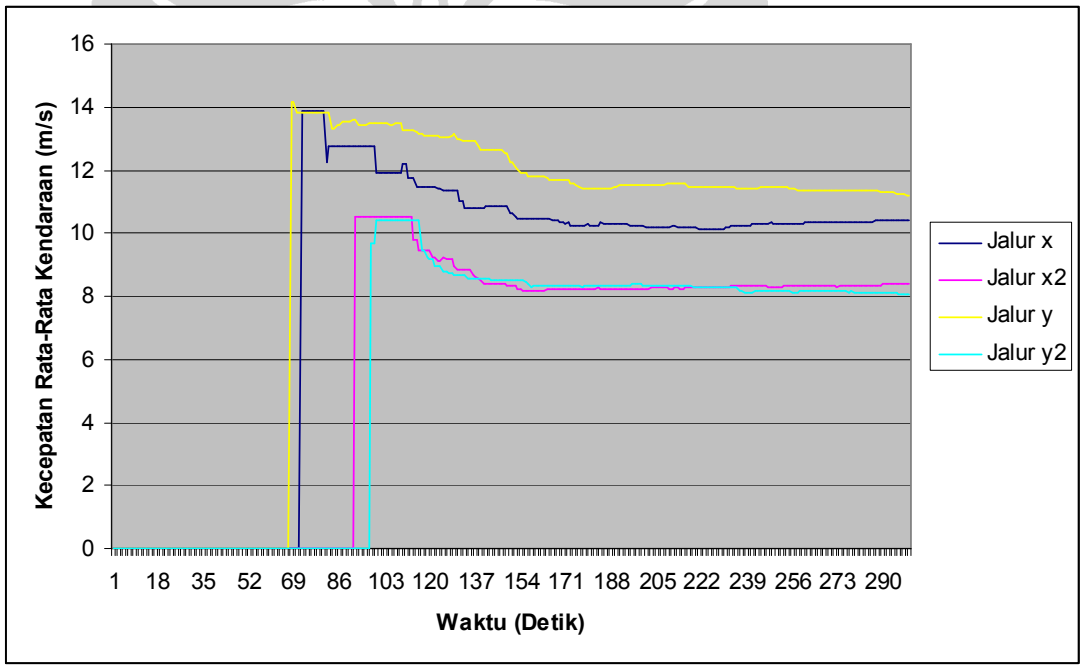
Nilai dari kecepatan rata-rata kendaraan baru bisa didapatkan mulai dari detik ke-65 ketika sudah ada satu kendaraan yang berhasil keluar dari jalur. Penulis akan menampilkan hasil simulasi per 10 detik, mulai dari detik ke 70 sampai dengan detik ke 300.

Tabel 3. Hasil Analisa Kondisi Sepi dan Seimbang Yang Tersinkronisasi

Detik ke-	Tersinkronisasi				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
70	0	0	13.83649	0	3.459123
80	13.87946	0	13.83649	0	6.92899
90	12.74667	0	13.52312	0	6.567449
100	11.90902	10.49563	13.49393	10.41179	11.57759
110	12.21274	10.49563	13.27349	10.41179	11.59841
120	11.49624	9.464847	13.111	9.150863	10.80326
130	11.36156	8.85184	12.9895	8.664003	10.46673
140	10.77173	8.390744	12.62272	8.542877	10.08202
150	10.61428	8.359177	12.23858	8.524906	9.934238
160	10.43616	8.179298	11.82412	8.315724	9.688824
170	10.35078	8.216871	11.70672	8.359182	9.658388
180	10.25574	8.222631	11.43372	8.322523	9.558652
190	10.302	8.212482	11.46206	8.327737	9.575599
200	10.23764	8.211374	11.53038	8.334516	9.578476

Detik ke-	Tersinkronisasi				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
210	10.20004	8.246264	11.56803	8.321103	9.58386
220	10.15425	8.262768	11.49262	8.292418	9.550514
230	10.13083	8.262768	11.49262	8.292418	9.544659
240	10.26511	8.329477	11.40694	8.131897	9.533356
250	10.3093	8.3042	11.45317	8.14606	9.553183
260	10.31028	8.324837	11.3782	8.172424	9.546434
270	10.36958	8.31544	11.3782	8.161062	9.55607
280	10.36958	8.318924	11.3782	8.127413	9.548529
290	10.39183	8.381028	11.28452	8.124058	9.545359
300	10.38487	8.37226	11.19351	8.051296	9.500484
Rata-Rata Kecepatan Kendaraan Tiap Jalur	**10.8731**	**8.61984**	**12.1268**	**8.5258**	**9.49332**

Berikut ini merupakan grafik kecepatan kendaraan dari tabel diatas:



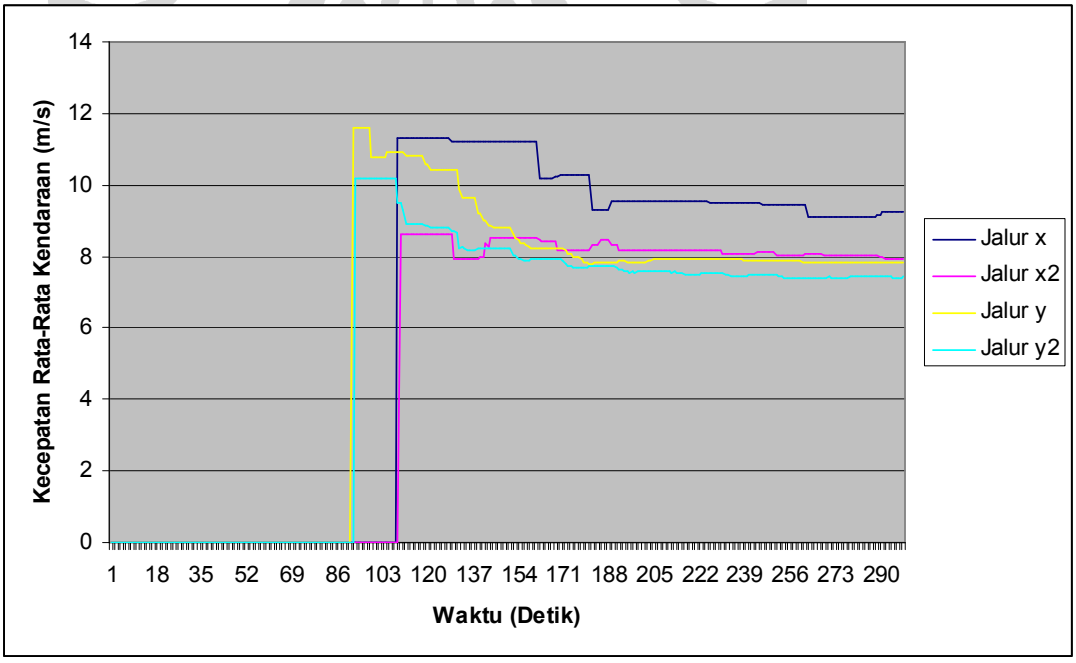
Gambar 10. Grafik Kecepatan Kendaraan Kondisi Sepi, Seimbang dan Tersinkronisasi

Tabel 4. Hasil Analisa Kondisi Sepi dan Seimbang Yang *Random*

Detik ke-	*Random*				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
70	0	0	14.26104	0	3.56526
80	0	0	14.26104	0	3.56526
90	0	0	12.75883	0	3.189707
100	0	10.25816	12.75883	0	5.754248
110	9.956743	10.25816	11.58183	0	7.949184

Detik ke-	*Random*				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
120	9.956743	10.25816	11.58183	8.38551	10.04556
130	8.365588	8.732842	11.512	8.38551	9.124764
140	8.365588	8.732842	9.02239	7.739914	8.465183
150	7.950294	7.918678	9.066021	7.504337	8.109832
160	7.950294	8.533328	8.595638	7.523077	8.150584
170	7.780597	8.377274	8.498505	7.523077	8.044863
180	7.780597	8.293716	8.066791	7.552678	7.923445
190	7.6564	8.095329	8.243049	7.476465	7.867811
200	7.6564	8.079888	8.208062	7.4591	7.850863
210	7.573249	8.023854	8.208062	7.433216	7.809595
220	7.573249	7.994916	8.048102	7.486703	7.775742
230	7.632133	7.827558	8.09707	7.486703	7.760866
240	7.632133	7.755534	8.100252	7.451642	7.73489
250	7.597588	7.60827	8.100252	7.442909	7.687255
260	7.597588	7.60827	8.006169	7.393566	7.651398
270	7.609441	7.612852	8.096144	7.393566	7.6780
280	7.609441	7.61592	8.00024	7.4792	7.656848
290	7.5842	7.563375	8.055118	7.395005	7.64841
300	7.5842	7.535355	8.058925	7.399046	7.643367
Rata-Rata Kecepatan Kendaraan Tiap Jalur	**7.98983**	**8.35653**	**9.56183**	**7.57849**	**7.42569**

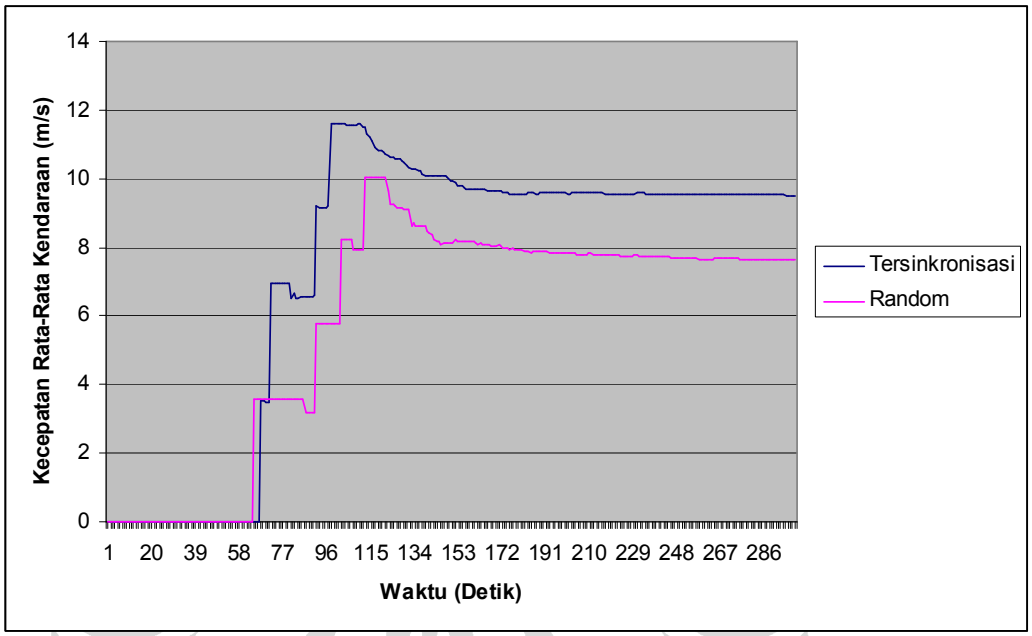
Berikut ini merupakan grafik kecepatan kendaraan dari tabel diatas:



Gambar 11. Grafik Kecepatan Kendaraan Kondisi Sepi, Seimbang dan *Random*

Terlihat dari pengujian diatas bahwa kecepatan rata-rata kendaraan yang menggunakan algoritma yang tersinkronisasi lebih cepat dari yang *random* pada kondisi lalu lintas yang sepi dan seimbang.

Berikut ini merupakan grafik perbandingan kecepatan kendaraannya:



Gambar 12. Grafik Perbandingan Kecepatan Kendaraan Pada Kondisi Sepi dan Seimbang

V.2 Hasil Analisa Kondisi Sepi Dan Tidak Seimbang

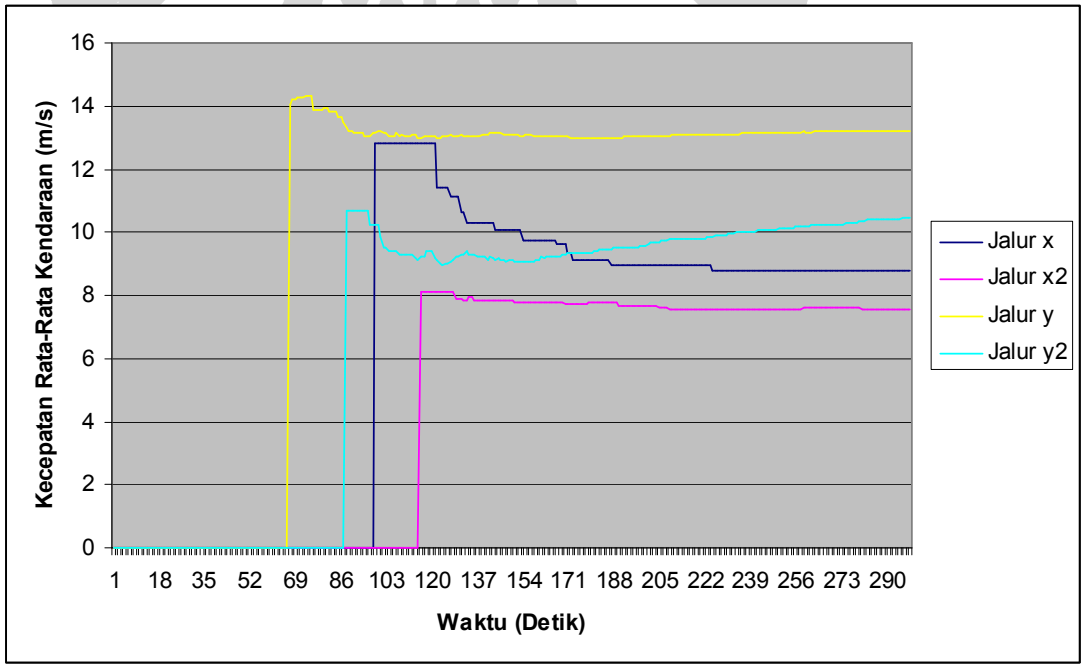
Nilai dari kecepatan rata-rata kendaraan baru bisa didapatkan mulai dari detik ke-68 ketika sudah ada satu kendaraan yang berhasil keluar dari jalur y pada simulasi. Penulis akan menampilkan hasil simulasi per 10 detik, mulai dari detik ke 70 sampai dengan detik ke 300.

Tabel 5. Hasil Analisa Kondisi Sepi dan Tidak Seimbang Yang Tersinkronisasi

Detik ke-	Tersinkronisasi				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
70	0	0	14.22941	0	3.557352
80	0	0	13.87035	0	3.467587
90	0	0	13.215	10.71033	5.981337
100	12.81996	0	13.12816	10.21924	9.041843
110	12.81996	0	13.08138	9.296218	8.79939

Detik ke-	Tersinkronisasi				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
120	12.81996	8.089888	13.02064	9.396109	10.83165
130	11.1367	7.883253	13.04809	9.217105	10.32129
140	10.28936	7.825763	13.08327	9.213625	10.103
150	10.08985	7.825763	13.07768	9.115113	10.0271
160	9.74471	7.771753	13.05504	9.118811	9.922578
170	9.621682	7.771753	13.002	9.275419	9.919719
180	9.103462	7.771695	12.97718	9.329044	9.795346
190	8.959045	7.751859	12.99259	9.489072	9.798141
200	8.943359	7.652997	13.04144	9.568864	9.8664
210	8.943359	7.574682	13.05032	9.76415	9.833128
220	8.943359	7.559312	13.08247	9.809722	9.848715
230	8.8561	7.5621	13.0813	9.908973	9.838462
240	8.8561	7.577552	13.13514	10.00377	9.879505
250	8.8561	7.578898	13.13033	10.09621	9.9748
260	8.8587	7.582985	13.17611	10.19219	9.938218
270	8.8587	7.582985	13.18788	10.26568	9.959532
280	8.8587	7.582985	13.2094	10.31342	9.976847
290	8.8587	7.562581	13.21818	10.40217	9.99613
300	8.8587	7.569749	13.21087	10.44348	10.00642
Rata-Rata Kecepatan Kendaraan Tiap Jalur	**9.66661**	**7.68971**	**10.2677**	**9.73902**	**9.11225**

Berikut ini merupakan grafik kecepatan kendaraan dari tabel diatas:

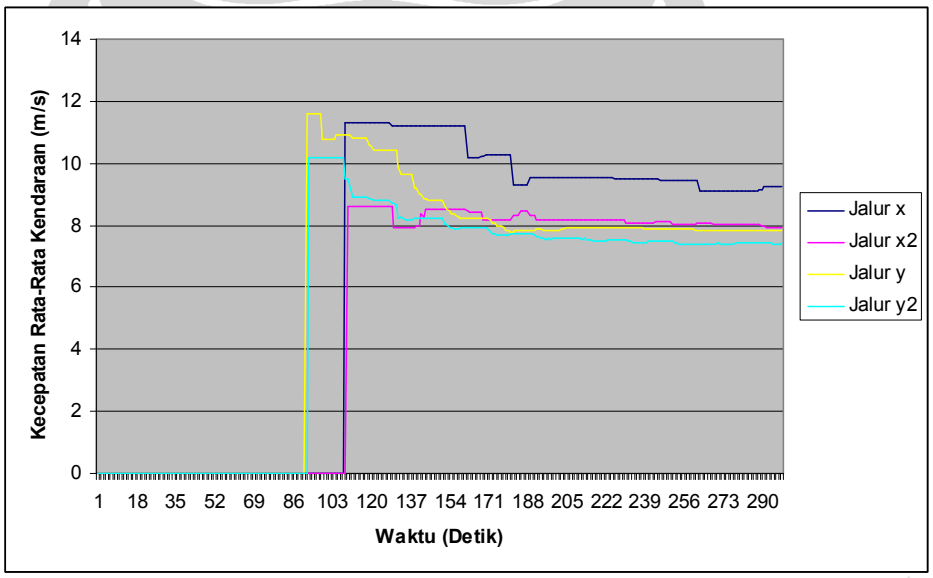


Gambar 13. Grafik Kecepatan Kendaraan Kondisi Sepi, Tidak Seimbang dan Tersinkronisasi

Tabel 6. Hasil Analisa Kondisi Sepi dan Tidak Seimbang Yang *Random*

Detik ke-	*Random*				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
70	0	0	0	0	0
80	0	0	0	0	0
90	0	0	0	0	0
100	0	0	10.78256	10.18918	5.242935
110	11.31179	0	10.906	9.518585	7.934097
120	11.31179	8.61244	10.54957	8.866809	9.835153
130	11.233	8.61244	10.40733	8.731606	9.746093
140	11.233	7.907142	9.222121	8.215735	9.1445
150	11.233	8.537762	8.82294	8.225631	9.204834
160	11.233	8.537762	8.231945	7.909803	8.978128
170	10.21314	8.183266	8.227	7.920477	8.63597
180	10.2773	8.183266	7.8521	7.702734	8.50385
190	9.54862	8.303598	7.835542	7.711574	8.349833
200	9.54862	8.18814	7.838984	7.568799	8.286135
210	9.54862	8.166612	7.92856	7.597992	8.310446
220	9.54862	8.186381	7.908444	7.499891	8.285834
230	9.47814	8.186381	7.95113	7.526999	8.285662
240	9.47814	8.090003	7.9051	7.45	8.230814
250	9.454036	8.11245	7.887948	7.467358	8.230448
260	9.454036	8.029405	7.859469	7.404502	8.186853
270	9.106198	8.024818	7.8502	7.413604	8.098705
280	9.106198	8.024818	7.82211	7.416603	8.092432
290	9.144057	8.024818	7.842161	7.439225	8.112565
300	9.237251	7.941479	7.813233	7.429707	8.105418
Rata-Rata Kecepatan Kendaraan Tiap Jalur	**9.99287**	**8.21575**	**8.61528**	**7.98658**	**8.31109**

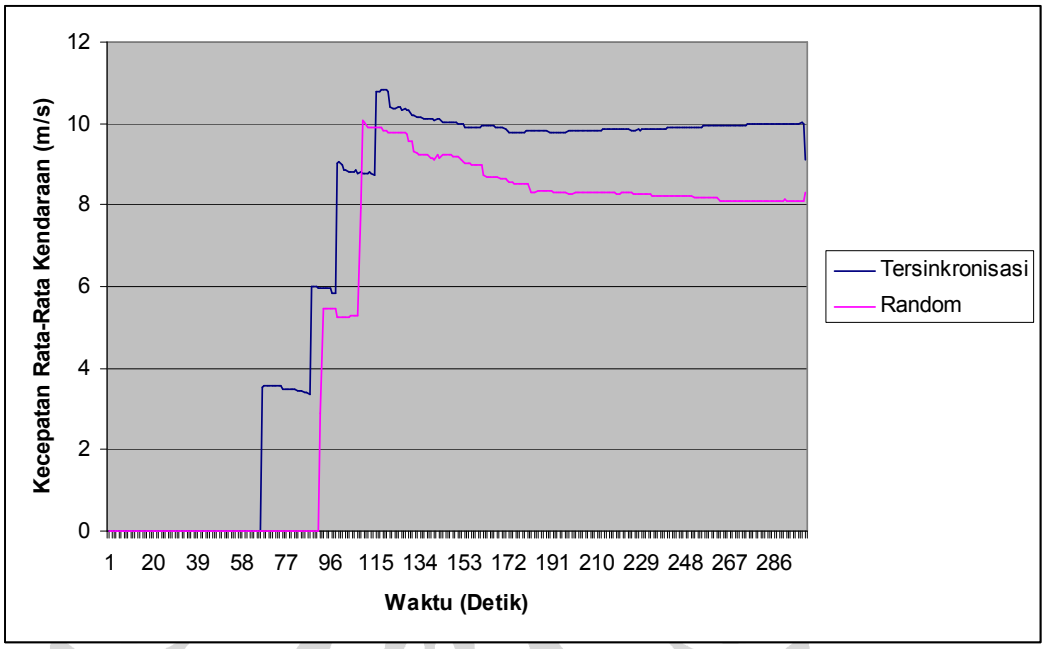
Berikut ini merupakan grafik kecepatan kendaraan dari tabel diatas:



Gambar 14. Grafik Kecepatan Kendaraan Kondisi Sepi, Tidak Seimbang dan *Random*

Terlihat dari pengujian diatas bahwa kecepatan rata-rata kendaraan yang menggunakan algoritma yang tersinkronisasi lebih cepat dari yang *random* pada kondisi lalu lintas yang sepi dan tidak seimbang.

Berikut ini merupakan grafik perbandingan kecepatan kendaraannya:



Gambar 15. Grafik Perbandingan Kecepatan Kendaraan Pada Kondisi Sepi dan Tidak Seimbang

V.3 HASIL ANALISA KONDISI PADAT DAN SEIMBANG

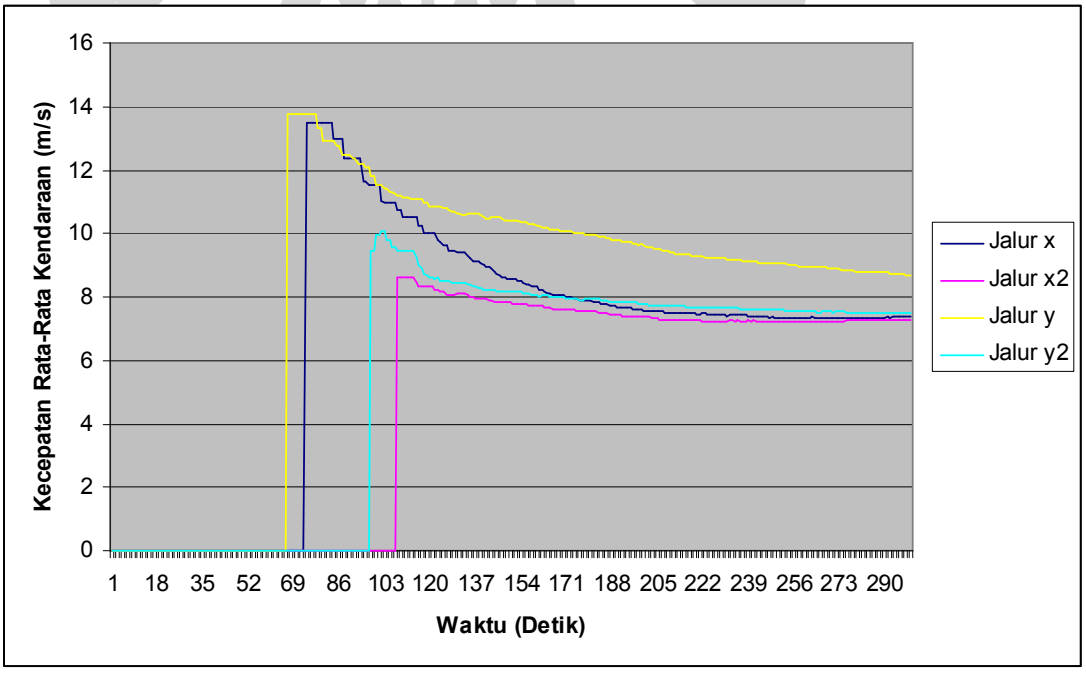
Nilai dari kecepatan rata-rata kendaraan baru bisa didapatkan mulai dari detik ke-67 ketika sudah ada satu kendaraan yang berhasil keluar dari jalur y pada simulasi. Penulis akan menampilkan hasil simulasi per 10 detik, mulai dari detik ke 70 sampai dengan detik ke 300.

Tabel 7. Hasil Analisa Kondisi Padat dan Seimbang Yang Tersinkronisasi

Detik ke-	Tersinkronisasi				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
70	0	0	13.77347	0	3.443368
80	13.45551	0	12.89816	0	6.588417
90	12.37686	0	12.47095	0	6.211954
100	11.54562	0	11.52185	9.951781	8.254811
110	10.54541	8.627962	11.14328	9.430891	9.936886

Detik ke-	Tersinkronisasi				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
120	9.997757	8.327757	10.86315	8.622861	9.452882
130	9.409594	8.084265	10.64787	8.474755	9.15412
140	8.992319	7.916381	10.5086	8.231676	8.912245
150	8.577544	7.804628	10.39521	8.174987	8.738093
160	8.310907	7.71732	10.23174	8.045162	8.576283
170	8.034771	7.594147	10.07195	7.985146	8.421504
180	7.870034	7.577758	9.954755	7.930569	8.333279
190	7.685531	7.414821	9.791187	7.83725	8.182197
200	7.576893	7.380516	9.603553	7.772022	8.083246
210	7.509528	7.278087	9.417478	7.69805	7.975786
220	7.465689	7.245502	9.284621	7.660835	7.914162
230	7.423796	7.2186	9.207645	7.663529	7.878393
240	7.403429	7.244072	9.112117	7.609568	7.842297
250	7.348382	7.222324	9.070629	7.587454	7.807197
260	7.338175	7.235581	8.953183	7.537505	7.766111
270	7.313239	7.243539	8.918224	7.526975	7.750494
280	7.341716	7.259951	8.806773	7.500741	7.727295
290	7.344357	7.26564	8.770406	7.506378	7.721695
300	7.376596	7.288752	8.679756	7.503345	7.712112
Rata-Rata Kecepatan Kendaraan Tiap Jalur	**8.67814**	**7.53498**	**10.2023**	**8.0**	**8.67814**

Berikut ini merupakan grafik kecepatan kendaraan dari tabel diatas:

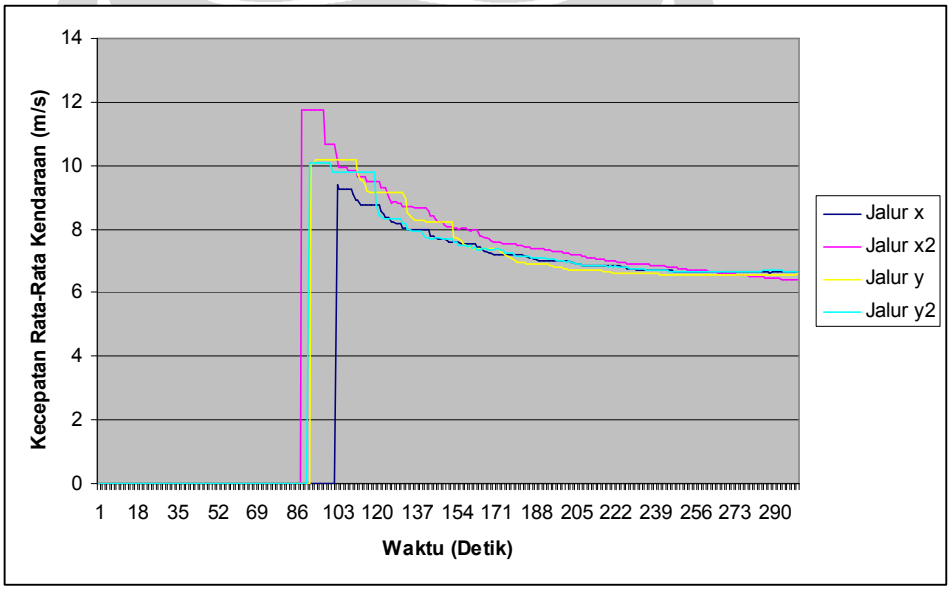


Gambar 16. Grafik Kecepatan Kendaraan Kondisi Padat, Seimbang dan Tersinkronisasi

Tabel 8. Hasil Analisa Kondisi Padat dan Seimbang Yang *Random*

Detik ke-	*Random*				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
70	**0**	0	0	0	0
80	**0**	0	0	0	0
90	**0**	11.72638	0	0	2.931596
100	**0**	10.6568	10.16962	10.08584	7.728064
110	**9.10278**	9.861472	10.16962	9.807195	9.735267
120	**8.77608**	9.505038	9.145067	8.776671	9.050714
130	**8.16426**	8.812214	9.145067	8.30203	8.605893
140	**7.984**	8.665987	8.292831	7.795354	8.183579
150	**7.64714**	8.0878	8.2396	7.693452	7.907447
160	**7.53854**	7.933077	7.43883	7.465886	7.594083
170	**7.19984**	7.607213	7.402662	7.365317	7.393758
180	**7.17324**	7.48385	6.9713	7.157453	7.196164
190	**7.00492**	7.368045	6.916639	7.088703	7.094578
200	**6.98793**	7.255155	6.758324	6.999008	7.0004
210	**6.85627**	7.108148	6.70661	6.877069	6.887025
220	**6.85194**	6.982553	6.646561	6.836531	6.829396
230	**6.73928**	6.914177	6.620496	6.758744	6.758175
240	**6.72423**	6.844615	6.594023	6.700232	6.715775
250	**6.65155**	6.750834	6.569873	6.655898	6.657039
260	**6.65877**	6.68324	6.575556	6.643974	6.640386
270	**6.63861**	6.593394	6.574232	6.664791	6.617757
280	**6.65337**	6.518176	6.563627	6.662896	6.599517
290	**6.63518**	6.453728	6.559309	6.69077	6.584746
300	**6.64098**	6.4	6.590484	6.677343	6.577452
Rata-Rata Kecepatan Kendaraan Tiap Jalur	**7.25874**	**7.81329**	**7.49566**	**7.48878**	**7.32477**

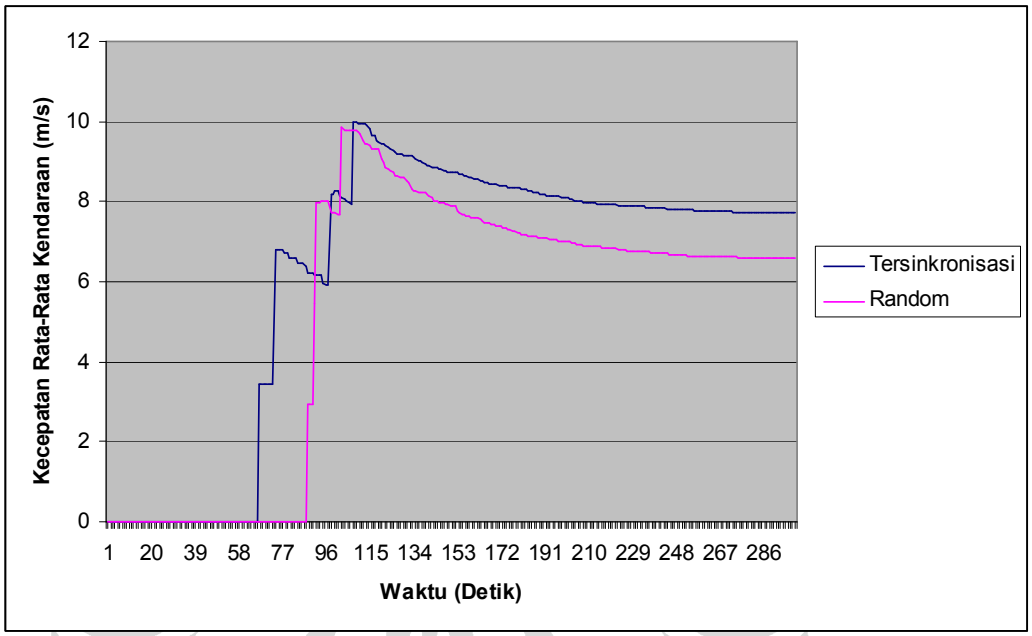
Berikut ini merupakan grafik kecepatan kendaraan dari tabel diatas:



Gambar 17. Grafik Kecepatan Kendaraan Kondisi Padat, Seimbang dan *Random*

Terlihat dari pengujian diatas bahwa kecepatan rata-rata kendaraan yang menggunakan algoritma yang tersinkronisasi lebih cepat dari yang *random* pada kondisi lalu lintas yang padat dan seimbang.

Berikut ini merupakan grafik perbandingan kecepatan kendaraannya:



Gambar 18. Grafik Perbandingan Kecepatan Kendaraan Pada Kondisi Padat dan Seimbang

V.4 Hasil Analisa Kondisi Padat Dan Tidak Seimbang

Nilai dari kecepatan rata-rata kendaraan baru bisa didapatkan mulai dari detik ke-68 ketika sudah ada satu kendaraan yang berhasil keluar dari jalur y pada simulasi. Penulis akan menampilkan hasil simulasi per 10 detik, mulai dari detik ke 70 sampai dengan detik ke 300.

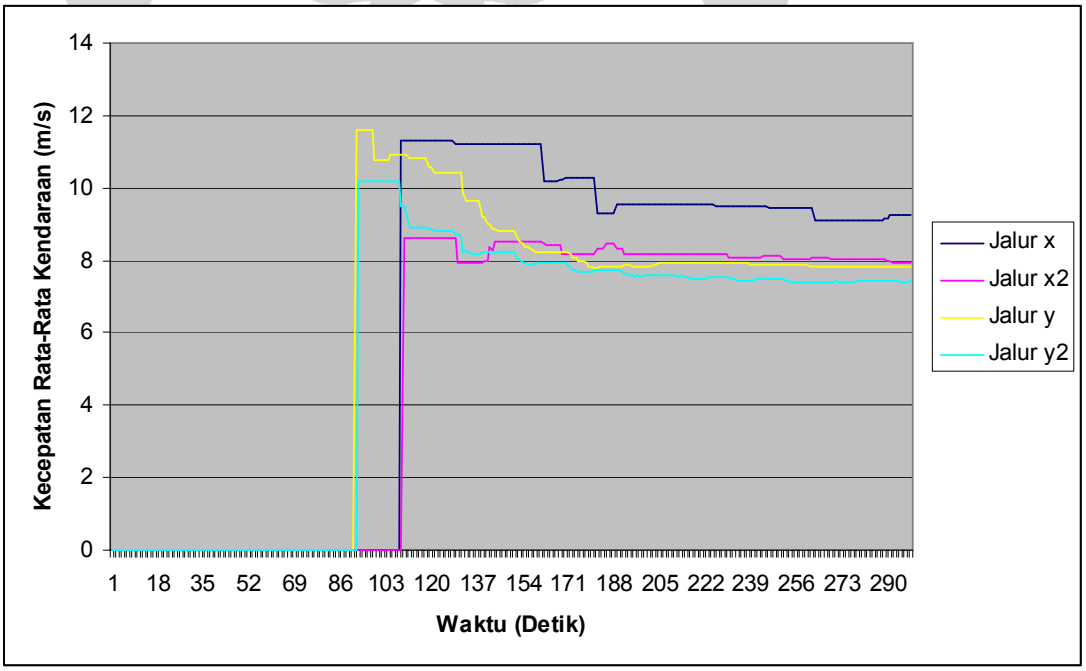
Tabel 9. Hasil Analisa Kondisi Padat dan Tidak Seimbang Yang Tersinkronisasi

Detik ke-	Tersinkronisasi				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
70	0	0	13.75684	0	3.43921
80	12.998	0	12.47273	0	6.367692
90	12.998	0	12.00471	0	6.250685
100	11.3955	0	11.40463	9.517465	8.079393
110	10.9732	0	10.92127	8.982507	7.719248

Detik ke-	Tersinkronisasi				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
120	10.262	8.20895	10.6118	8.313005	9.348937
130	9.96943	7.98297	10.36189	7.989691	9.075996
140	9.38613	7.82355	10.045	7.899581	8.788569
150	9.23434	7.68449	9.703882	7.780032	8.600685
160	8.86339	7.57176	9.405384	7.711464	8.387998
170	8.74689	7.5123	9.182268	7.677898	8.279838
180	8.4946	7.50988	9.056842	7.644073	8.176348
190	8.35079	7.46228	8.969645	7.583948	8.091664
200	8.14332	7.4381	8.962088	7.530565	8.8518
210	8.02094	7.4346	8.863617	7.474417	7.948394
220	7.86472	7.38544	8.870785	7.457882	7.894706
230	7.7203	7.35667	8.800357	7.446299	7.830906
240	7.57202	7.28022	8.783253	7.415431	7.762733
250	7.44382	7.27874	8.749465	7.397482	7.717378
260	7.35992	7.20741	8.713217	7.392435	7.668247
270	7.26373	7.17834	8.681517	7.38891	7.628125
280	7.16525	7.10814	8.637715	7.377348	7.572113
290	7.07295	7.07119	8.620375	7.388392	7.538227
300	6.97131	6.99336	8.581043	7.36966	7.478843
Rata-Rata Kecepatan Kendaraan Tiap Jalur	7.115	7.66927	7.3549	7.3488	7.18975

Berikut ini merupakan grafik kecepatan kendaraan dari tabel diatas:

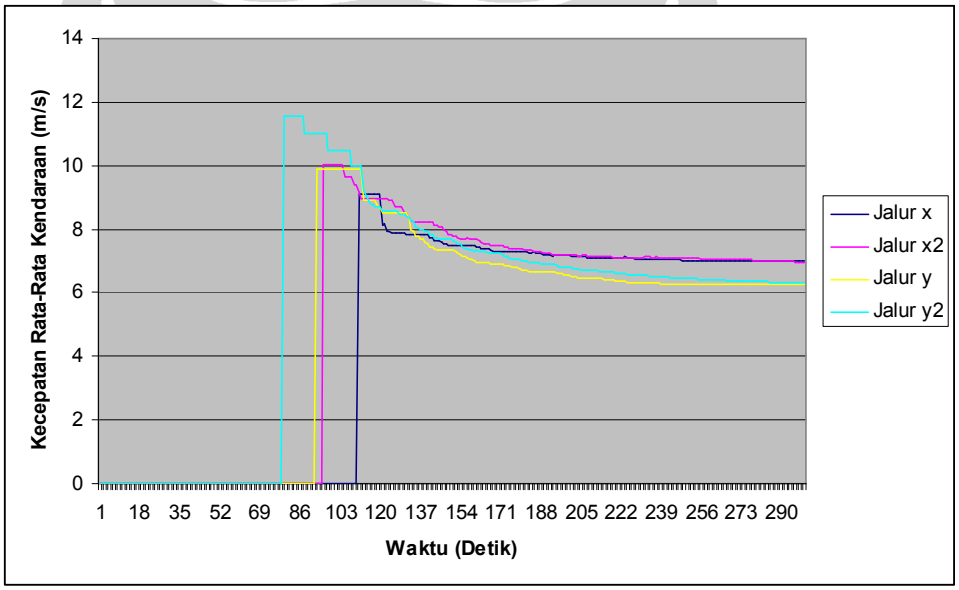
Gambar 19. Grafik Kecepatan Kendaraan Kondisi Padat, Tidak Seimbang dan Tersinkronisasi



Tabel 10. Hasil Analisa Kondisi Padat dan Tidak Seimbang Yang *Random*

Detik ke-	*Random*				Rata-Rata Kendaraan Per Detik
	Jalur x	Jalur x2	Jalur y	Jalur y2	
70	0	0	0	0	0
80	0	0	0	11.550	2.887503
90	0	0	0	10.99143	2.747857
100	0	10.03658	9.997	10.48864	7.606798
110	0	9.409075	9.997	9.991308	7.325588
120	9.102402	8.959643	8.712311	8.706777	8.870283
130	7.904554	8.53832	8.502168	8.47983	8.356218
140	7.831613	8.247646	7.531464	7.932757	7.88587
150	7.465554	7.81476	7.328611	7.668063	7.569247
160	7.483423	7.679583	6.98428	7.314595	7.36547
170	7.293368	7.487819	6.919889	7.220837	7.230478
180	7.311754	7.388	6.726335	7.1728	7.109457
190	7.192128	7.230389	6.658034	6.909265	6.997454
200	7.188582	7.190508	6.544987	6.789966	6.928511
210	7.103385	7.153632	6.453776	6.690774	6.850392
220	7.122344	7.121519	6.374671	6.629798	6.812083
230	7.051306	7.112506	6.324713	6.553481	6.7605
240	7.051586	7.104713	6.280374	6.499339	6.734003
250	7.4503	7.095571	6.272313	6.44868	6.707767
260	7.023808	7.065733	6.256545	6.413925	6.690003
270	6.983874	7.049329	6.257632	6.381571	6.6681
280	6.9985	7.02367	6.260836	6.35448	6.659371
290	6.987346	7.0209	6.260493	6.32765	6.644174
300	6.990286	6.961987	6.264874	6.298697	6.628961
Rata-Rata Kecepatan Kendaraan Tiap Jalur	**6.174411**	**7.477934**	**7.112969**	**7.332807**	**7.178296**

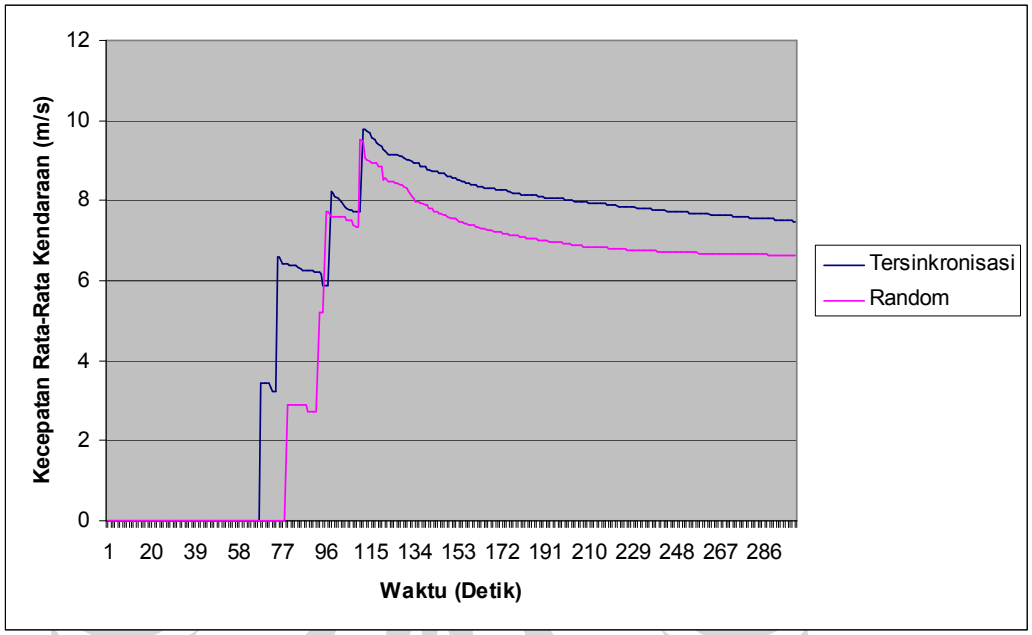
Berikut ini merupakan grafik kecepatan kendaraan dari tabel diatas:



Gambar 20. Grafik Kecepatan Kendaraan Kondisi Padat, Tidak Seimbang dan *Random*

Terlihat dari pengujian diatas bahwa kecepatan rata-rata kendaraan yang menggunakan algoritma yang tersinkronisasi lebih cepat dari yang *random* pada kondisi lalu lintas yang padat dan tidak seimbang.

Berikut ini merupakan grafik perbandingan kecepatan kendaraannya:



Gambar 21. Grafik Perbandingan Kecepatan Kendaraan Pada Kondisi Padat dan Tidak Seimbang

Terlihat dari pengujian diatas, sebagian besar hasil menunjukkan bahwa *worst case* untuk simulasi yang menggunakan sinkronisasi masih lebih baik daripada rata-rata *best case* untuk simulasi yang dilakukan secara *random.* Untuk lebih jelasnya, hasil keseluruhan pengujian dapat dilihat pada lampiran.