

SKRIPSI

PENCARIAN JUMLAH KAMERA STATIS MINIMUM DALAM SUATU RUANGAN MENGGUNAKAN LINEAR PROGRAMMING



Prayogo Cendra

NPM: 2014730033

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2018

UNDERGRADUATE THESIS

**FINDING MINIMUM STATIC CAMERA IN A ROOM USING
LINEAR PROGRAMMING**



Prayogo Cendra

NPM: 2014730033

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2018**

ABSTRAK

Kamera CCTV merupakan kamera yang digunakan untuk memantau suatu lokasi dengan tujuan pengawasan dan keamanan. Kamera CCTV pada umumnya dipasang di tempat-tempat strategis sehingga mendapatkan jangkauan yang baik. Penempatan kamera CCTV di ruangan yang berbentuk sederhana (persegi panjang) relatif tidak sulit. Kamera CCTV yang dibutuhkan pada umumnya berjumlah dua buah dan dipasang di kedua sudut ruangan sehingga saling berhadapan. Namun, jika ruangan berukuran besar, maka tujuan penggunaan kamera CCTV bukan hanya untuk mendeteksi adanya orang, melainkan juga untuk mengenali orang tersebut. Hal ini dapat menyebabkan kesulitan dalam menentukan jumlah minimum dan lokasi penempatan kamera CCTV yang dapat mencakup seluruh ruangan.

Pada skripsi ini, masalah akan akan dipelajari lebih lanjut dengan memahami setiap elemen pembentuk masalah. Selanjutnya, masalah ini akan dirumuskan lebih lanjut agar menjadi lebih konkret. Untuk menyelesaikan masalah ini, penulis menggunakan metode linear programming karena metode ini dapat menyelesaikan masalah optimasi. Masalah ini termasuk ke dalam jenis masalah optimasi karena solusi yang diharapkan harus bersifat paling optimal, yaitu penempatan kamera CCTV yang berjumlah minimum yang dapat mencakup seluruh ruangan.

Selain merumuskan masalah, penulis juga membangun perangkat lunak yang dapat mensimulasikan masalah. Perangkat lunak ini dapat menerima masukan-masukan masalah dan menyelesaikannya menggunakan metode linear programming. Tidak hanya menyelesaikannya saja, perangkat lunak juga dapat memvisualisasikan solusinya sehingga penempatan-penempatan kamera CCTV dapat dipahami dengan lebih mudah.

Kata-kata kunci: cctv, linear programming

ABSTRACT

CCTV cameras are cameras used to monitor a location with the purpose of surveillance and security. CCTV cameras are generally installed in strategic places to get a good coverage. The placement of CCTV cameras in a simple room (rectangle) is relatively not difficult. CCTV cameras that are needed in general amount to two pieces and installed in both corners of the room so they are facing each other. However, if the room is large, then the purpose of using CCTV cameras is not only to detect people, but also to recognize the person. This can cause difficulties in determining the minimum number and location of CCTV camera placement that can cover the entire room.

In this thesis, the problem will be studied further by understanding every problem-forming element. Furthermore, this problem will be formulated further to be more concrete. To solve this problem, the author uses linear programming method because this method can solve the optimization problem. This problem belongs to the type of optimization problem because the expected solution should be the most optimal, that is the minimum placements of CCTV camera that can cover the entire room.

In addition to formulating the problem, the authors also build software that can simulate the problem. This software can receive input problems and solve them using linear programming method. Not only solve it, the software can also visualize the solution so that the placement of CCTV cameras can be understood more easily.

Keywords: cctv, linear programming

DAFTAR ISI

DAFTAR ISI	ix
DAFTAR GAMBAR	xi
DAFTAR TABEL	xiii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	2
1.4 Batasan Masalah	2
1.5 Metodologi	2
1.6 Sistematika Pembahasan	3
2 LANDASAN TEORI	5
2.1 Linear Programming [1]	5
2.1.1 Karakteristik	5
2.1.2 Daerah <i>Feasible</i> dan Solusi Optimal	6
2.1.3 Bentuk Standar	8
2.1.4 Variabel Basis dan Non-basis	10
2.1.5 Metode Simplex	10
2.2 Integer Programming [1]	16
2.2.1 Metode <i>Branch and Bound</i>	16
2.3 Kakas <i>lp_solve</i>	17
3 ANALISIS	19
3.1 Pemodelan Masalah	19
3.1.1 Ruang	19
3.1.2 Kamera CCTV	20
3.1.3 Penempatan Kamera CCTV	20
3.1.4 Daerah Cakupan	20
3.1.5 <i>Overlap</i> dan <i>Out of Bound</i>	23
3.2 Penyelesaian Masalah	24
3.2.1 Variabel Keputusan	24
3.2.2 Fungsi Tujuan	25
3.2.3 Batasan	25
3.2.4 Bentuk Masalah Linear Programming	26
3.3 Analisis Kebutuhan Perangkat Lunak	26
3.3.1 Diagram <i>Use Case</i>	26
3.3.2 Diagram Kelas	27
4 PERANCANGAN	29
4.1 Perancangan Antarmuka	29

4.2	Perancangan Kelas	30
4.2.1	Kelas <i>Angle</i>	31
4.2.2	Kelas <i>Point</i>	34
4.2.3	Kelas <i>CameraSpecification</i>	34
4.2.4	Kelas <i>CameraPlacement</i>	35
4.2.5	Kelas <i>Dimension</i>	35
4.2.6	Kelas <i>Cell</i>	36
4.2.7	Kelas <i>GridPoint</i>	36
4.2.8	Kelas <i>Room</i>	37
4.2.9	Kelas <i>MinimumCameraPlacementSolver</i>	39
4.2.10	Kelas <i>MinimumCameraPlacementSolverLPSolve</i>	40
5	IMPLEMENTASI DAN PENGUJIAN	41
5.1	Lingkungan Implementasi Perangkat Keras	41
5.2	Lingkungan Implementasi Perangkat Keras	41
5.3	Implementasi Antarmuka	41
5.4	Pengujian Fungsional	43
5.5	Pengujian Eksperimental	47
5.5.1	Eksperimen Ukuran Cell	47
5.5.2	Eksperimen Rasio Sisi Terpendek Ruangan dengan Jarak Pandang Kamera CCTV	48
5.5.3	Eksperimen Besar Sudut Pandang Kamera CCTV	50
6	KESIMPULAN DAN SARAN	53
6.1	Kesimpulan	53
6.2	Saran	53
	DAFTAR REFERENSI	55
	A KODE PROGRAM	57
	B HASIL EKSPERIMEN	69

DAFTAR GAMBAR

2.1	Contoh masalah linear programming dengan daerah <i>feasible</i>	7
2.2	<i>Corner points</i> pada daerah <i>feasible</i>	7
2.3	Contoh masalah linear programming tanpa daerah <i>feasible</i>	8
3.1	Pemodelan ruangan	19
3.2	Pemodelan kamera CCTV	20
3.3	Pemodelan penempatan kamera CCTV	21
3.4	Contoh masalah yang memiliki kasus bentuk daerah tidak sederhana	21
3.5	Pemodelan ruangan dalam bentuk grid point	22
3.6	Daerah cakupan sebelum pemodelan grid point	22
3.7	Daerah cakupan sesudah pemodelan grid point	23
3.8	Daerah overlap dan out of bound	24
3.9	Diagram <i>use case</i>	27
3.10	Diagram kelas sederhana	28
4.1	Perancangan antarmuka penerima masukan	29
4.2	Perancangan antarmuka penempatan kamera CCTV	31
4.4	Diagram kelas <i>Angle</i>	31
4.3	Diagram kelas <i>rinci</i>	32
4.5	Diagram kelas <i>Point</i>	34
4.6	Diagram kelas <i>CameraSpecification</i>	34
4.7	Diagram kelas <i>CameraPlacement</i>	35
4.8	Diagram kelas <i>Dimension</i>	35
4.9	Diagram kelas <i>Cell</i>	36
4.10	Diagram kelas <i>GridPoint</i>	36
4.11	Diagram kelas <i>Room</i>	37
4.12	Diagram kelas <i>MinimumCameraPlacementSolver</i>	39
4.13	Diagram kelas <i>MinimumCameraPlacementSolverLPSolve</i>	40
5.1	Antarmuka penerima masukan	42
5.2	Antarmuka penempatan kamera CCTV	42
5.3	Tampilan pengisian masukan masalah	43
5.4	Tampilan setelah pengisian masukan masalah	44
5.5	Tampilan panel informasi setelah pengisian masukan masalah	44
5.6	Tampilan panel penambahan penempatan kamera CCTV	45
5.7	Tampilan panel informasi setelah menambah penempatan kamera CCTV	45
5.8	Tampilan panel visualisasi setelah menambah penempatan kamera CCTV	45
5.9	Tampilan panel informasi setelah membuang penempatan kamera CCTV	46
5.10	Tampilan panel visualisasi setelah membuang penempatan kamera CCTV	47
B.1	Hasil eksperimen ukuran cell, pertama	69
B.2	Hasil eksperimen ukuran cell, kedua	70
B.3	Hasil eksperimen ukuran cell, ketiga	70

B.4	Hasil eksperimen rasio, pertama	71
B.5	Hasil eksperimen rasio, kedua	71
B.6	Hasil eksperimen rasio, ketiga	72
B.7	Hasil eksperimen rasio, keempat	72
B.8	Hasil eksperimen besar sudut pandang, pertama	73
B.9	Hasil eksperimen besar sudut pandang, kedua	73
B.10	Hasil eksperimen besar sudut pandang, ketiga	74
B.11	Hasil eksperimen besar sudut pandang, keempat	74

DAFTAR TABEL

2.1	Tabel simplex 0	11
2.2	Proses <i>pivotting</i> 1	12
2.3	Tabel simplex 1	13
2.4	Proses <i>pivotting</i> 2	14
2.5	Tabel simplex 2	15
2.6	Tabel simplex 0 untuk kasus minimasi	16
2.7	Tabel simplex 1 untuk kasus minimasi	16

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Kamera merupakan alat/komponen optik yang digunakan untuk mengambil citra/gambar. Salah satu penggunaan kamera dalam kehidupan sehari-hari adalah kamera CCTV (*closed-circuit television*). Kamera CCTV digunakan untuk memantau suatu lokasi dengan tujuan pengawasan dan keamanan. Kamera CCTV pada umumnya dipasang pada tempat strategis sehingga memiliki tingkat jangkauan yang baik. Kamera CCTV bekerja dengan cara merekam lokasi dalam jangkauannya secara terus menerus dan menyimpan hasil rekamannya dalam media penyimpanan. Rekaman ini biasanya digunakan oleh petugas keamanan untuk memantau lokasi tersebut dari tempat yang berbeda sehingga petugas tidak perlu mendatangi lokasi tersebut. Petugas hanya perlu datang apabila melihat hal-hal yang mencurigakan dari hasil rekaman CCTV.

Penempatan kamera CCTV di ruangan yang berbentuk sederhana (persegi panjang) relatif tidak sulit. Kamera CCTV yang dibutuhkan pada umumnya berjumlah dua buah dan dipasang di kedua sudut ruangan yang merupakan satu diagonal sehingga saling berhadapan. Namun, jika ruangan berukuran besar, maka tujuan penggunaan kamera CCTV tidak hanya untuk mendeteksi adanya orang, tetapi juga mengenali orang tersebut. Hal ini menyebabkan kesulitan dalam menentukan jumlah minimum dan lokasi penempatan kamera CCTV. Terdapat beberapa pendekatan yang dapat digunakan untuk menyelesaikan masalah ini, seperti dengan cara memasang kamera CCTV pada daerah-daerah yang dapat dimasuki orang. Tetapi pada kasus terburuk, orang bisa saja masuk melewati jalur-jalur yang tidak diduga, seperti tembok, atap, bawah tanah, dsb. Oleh karena itu, alangkah baiknya pemasangan kamera CCTV dilakukan hingga seluruh daerah pada ruangan tersebut dapat tercakup.

Penempatan kamera CCTV dapat dilakukan di berbagai lokasi dalam berbagai arah pandang. Apabila penempatan kamera CCTV dilakukan tanpa adanya perhitungan, maka terdapat kemungkinan di mana jumlah kamera akan terlalu banyak dan/atau. Dalam penempatan kamera CCTV terdapat perhitungan tingkat *overlap* dan tingkat *out of bound*. *Overlap* merupakan bagian ruang yang dicakup oleh lebih dari 1 kamera CCTV. Sedangkan, *out of bound* adalah cakupan kamera CCTV yang terhalang oleh sisi ruangan.

Pada skripsi ini, akan dibuat sebuah perangkat lunak yang akan mencari penempatan-penempatan kamera CCTV yang berjumlah minimum berdasarkan ukuran ruangan, jarak pandang efektif kamera CCTV, dan sudut pandang kamera CCTV. Penempatan kamera CCTV terdiri dari lokasi penempatan dan sudut arah pandang yang dituju. Selain itu, perangkat lunak juga akan menghasilkan visualisasi dari solusi yang didapatkan. Dengan adanya visualisasi, penempatan-penempatan kamera

1 CCTV dapat dipahami dengan lebih baik.

2 1.2 Rumusan Masalah

3 Berdasarkan latar belakang masalah, maka ditetapkan rumusan masalah sebagai berikut:

- 4 • Bagaimana cara mencari jumlah minimum penempatan kamera CCTV dalam suatu ruangan
- 5 yang dapat mencakup seluruh ruangan?
- 6 • Bagaimana cara memvisualisasikan penempatan-penempatan kamera CCTV dalam suatu
- 7 ruangan?

8 1.3 Tujuan

9 Berdasarkan rumusan masalah, maka tujuan dalam skripsi ini adalah:

- 10 • Mempelajari cara menentukan jumlah minimum penempatan kamera CCTV dalam suatu
- 11 ruangan yang dapat mencakup seluruh ruangan.
- 12 • Membangun perangkat lunak yang dapat mencari jumlah minimum penempatan kamera
- 13 CCTV dan memvisualisasikan penempatan-penempatan kamera CCTV dalam suatu ruangan.

14 1.4 Batasan Masalah

15 Dalam pembahasan masalah ini, terdapat batasan-batasan masalah sebagai berikut:

- 16 • Ruangan dimodelkan dalam bidang 2 dimensi berbentuk persegi panjang.
- 17 • Spesifikasi kamera CCTV yang digunakan terdiri dari jarak pandang efektif dan besar sudut
- 18 pandang.

19 1.5 Metodologi

- 20 • Mempelajari metode linear programming
- 21 • Mempelajari metode integer programming
- 22 • Mempelajari penggunaan kakas *lp_solve*
- 23 • Melakukan pemodelan masalah
- 24 • Menerapkan metode linear programming dalam penyelesaian masalah
- 25 • Merancang perangkat lunak
- 26 • Membangun perangkat lunak
- 27 • Menguji perangkat lunak
- 28 • Membuat kesimpulan

1.6 Sistematika Pembahasan

- **Bab 1 Pendahuluan**

Pada bagian ini akan dijelaskan latar belakang masalah, rumusan masalah, tujuan dan batasan masalah. Terdapat penjelasan mengenai metodologi yang digunakan dalam melakukan penelitian ini.

- **Bab 2 Landasan Teori**

Pada bagian ini terdapat pembahasan teori-teori yang digunakan untuk menyelesaikan masalah. Terdapat teori mengenai linear programming dan integer programming. Selain itu, terdapat penjelasan mengenai kakas *lp_solve* yang digunakan untuk membantu menyelesaikan masalah.

- **Bab 3 Analisis**

Pada bagian ini terdapat penjelasan mengenai pemodelan masalah dan cara penyelesaian masalah menggunakan metode linear programming. Selain itu, terdapat analisis kebutuhan perangkat lunak yang terdiri dari diagram *use case* dan diagram kelas sederhana.

- **Bab 4 Perancangan**

Pada bagian ini dijelaskan bentuk perancangan antarmuka dan perancangan kelas diagram rinci yang digunakan untuk membangun perangkat lunak.

- **Bab 5 Pengujian**

Pada bagian ini dibahas hasil-hasil pengujian yang dilakukan. Pengujian terdiri dari pengujian fungsional dan pengujian eksperimental.

- **Bab 6 Kesimpulan dan Saran**

Pada bagian ini terdapat hal-hal apa saja yang didapatkan melalui penelitian ini. Selain itu, terdapat saran dari penulis bagi orang lain yang ingin melanjutkan penelitian ini.

BAB 2

LANDASAN TEORI

2.1 Linear Programming [1]

Linear programming adalah sarana untuk menyelesaikan masalah optimasi [1]. Optimasi merupakan usaha untuk memilih solusi terbaik dari suatu kumpulan solusi yang ada. Setiap masalah optimasi perlu diubah terlebih dahulu ke dalam bentuk masalah linear programming agar dapat diselesaikan menggunakan linear programming. Pada bagian ini, akan dibahas karakteristik dan cara menyelesaikan masalah linear programming.

2.1.1 Karakteristik

Masalah linear programming memiliki karakteristik sebagai berikut:

- **Variabel keputusan**

Dalam masalah linear programming, terdapat variabel keputusan yang mendeskripsikan keputusan yang akan diambil. Contoh:

x_1 = jumlah produk A yang diproduksi

x_2 = jumlah produk B yang diproduksi

- **Fungsi Tujuan**

Dalam masalah linear programming, terdapat suatu keuntungan yang ingin dimaksimalkan atau suatu kerugian yang ingin diminimalkan dengan menggunakan fungsi tujuan yang terdiri dari variabel-variabel keputusan. Contoh:

$$\max z = 3x_1 + 2x_2$$

- **Batasan**

Dalam masalah linear programming, terdapat batasan-batasan yang membuat variabel keputusan memiliki nilai yang dibatasi. Batasan juga membuat suatu variabel keputusan berkaitan dengan variabel keputusan lainnya. Contoh:

$$2x_1 + x_2 \leq 100$$

$$x_1 + x_2 \leq 80$$

$$x_1 \leq 40$$

• Batasan non-negatif

Dalam masalah linear programming, batasan non-negatif ($x_i \geq 0$) pada variabel keputusan menyatakan bahwa variabel keputusan tersebut tidak dapat bernilai negatif. Dengan batasan non-negatif, variabel keputusan diharuskan untuk bernilai 0 atau positif. Contoh:

$$x_1 \geq 0$$

$$x_2 \geq 0$$

Dengan keempat karakteristik di atas, setiap masalah optimasi dapat diubah ke dalam masalah linear programming untuk diselesaikan. Berikut contoh masalah linear programming:

$$\max z = 3x_1 + 2x_2$$

$$s.t. \ 2x_1 + x_2 \leq 100$$

$$x_1 + x_2 \leq 80$$

$$x_1 \leq 40$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

Tulisan "s.t." atau "subject to" menandakan bahwa nilai dari setiap variabel keputusan harus memenuhi setiap batasan dan batasan non-negatif.

2.1.2 Daerah *Feasible* dan Solusi Optimal

Dalam masalah linear programming, terdapat daerah yang bernama daerah *feasible*. Daerah *feasible* dalam suatu masalah linear programming merupakan himpunan yang terdiri dari seluruh titik yang memenuhi setiap batasan dan batasan non-negatif [1]. Dengan demikian, setiap titik yang berada di dalam daerah *feasible* merupakan solusi terhadap permasalahan tersebut. Apabila dipilih suatu titik yang berada di luar daerah *feasible*, maka titik ini disebut dengan titik *infeasible*. Titik *infeasible* bukan merupakan solusi bagi permasalahan karena titik ini tidak memenuhi setiap batasan dan batasan non-negatif. Untuk menggambarkan daerah *feasible*, digunakan contoh batasan-batasan sebagai berikut:

$$2x_1 + x_2 \leq 100$$

$$x_1 + x_2 \leq 80$$

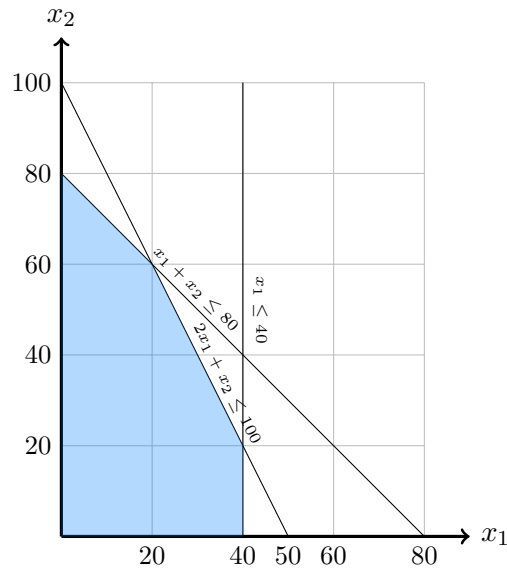
$$x_1 \leq 40$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

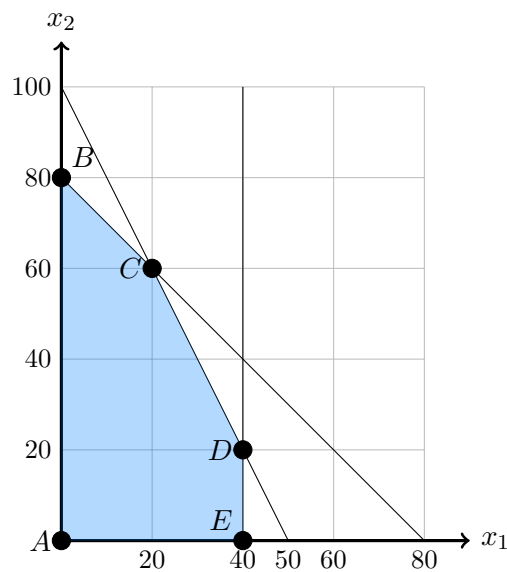
Pada contoh masalah tersebut, hanya terdapat 2 variabel keputusan sehingga dapat digambarkan dalam diagram kartesius. Pada gambar 2.1 terlihat bahwa batasan-batasan membentuk daerah

1 *feasible*.



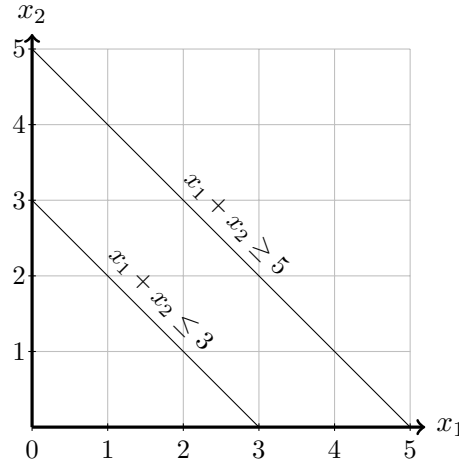
Gambar 2.1: Contoh masalah linear programming dengan daerah *feasible*

2 Agar suatu masalah linear programming memiliki solusi optimal, maka daerah *feasible* harus
 3 berbentuk *convex set*. Suatu himpunan titik S merupakan *convex set* apabila segmen garis yang
 4 menghubungkan setiap pasang titik dalam himpunan S berada dalam himpunan S [1]. Apabila
 5 masalah linear programming memiliki daerah *feasible*, maka daerah tersebut berbentuk *convex set*.
 6 Solusi optimal dari masalah tersebut berada pada salah satu *corner point* pada daerah *feasible*.
 7 *Extreme point* adalah titik perpotongan antar batasan dan *corner point* adalah *extreme point* yang
 8 berada dalam daerah *feasible*. Dengan adanya *corner points*, solusi optimal dapat dicari karena
 9 berjumlah terhingga, sehingga tidak perlu memeriksa seluruh kemungkinan solusi masalah yang
 10 berjumlah tak hingga. Pada gambar 2.2, titik A, B, C, D, dan E merupakan titik *corner points*
 11 yang salah satu di antaranya akan menghasilkan solusi optimal.



Gambar 2.2: *Corner points* pada daerah *feasible*

1 Tidak semua masalah linear programming memiliki daerah *feasible*. Apabila batasan-batasan
 2 dalam masalah tidak dapat membentuk daerah *feasible*, maka masalah tersebut tidak memiliki solusi
 3 apapun, sehingga solusi optimal dari masalah tersebut tidak dapat dicari. Gambar 2.3 menunjukkan
 4 contoh masalah linear programming yang tidak memiliki daerah *feasible*.



Gambar 2.3: Contoh masalah linear programming tanpa daerah *feasible*

5 2.1.3 Bentuk Standar

6 Setiap masalah linear programming harus diubah ke dalam bentuk standar sebelum diselesaikan.
 7 Berikut ini merupakan struktur dari bentuk standar pada masalah linear programming:

$$\begin{aligned}
 \max z &= c_1x_1 + c_2x_2 + \dots + c_nx_n \\
 \text{s.t. } a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\
 a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\
 &\vdots \\
 a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n &= b_m \\
 x_1 \geq 0, x_2 \geq 0, \dots, x_n &\geq 0
 \end{aligned}$$

8 Bentuk standar masalah linear programming terdiri dari n buah variabel keputusan x dan m
 9 buah persamaan batasan. Setiap batasan dalam bentuk standar harus dalam bentuk persamaan
 10 dan batasan non-negatif diberlakukan untuk setiap variabel keputusan. Bentuk standar linear
 11 programming dapat dinyatakan dalam notasi matriks seperti berikut ini:

$$\begin{aligned}
 \max z &= C^T x \\
 \text{s.t. } Ax &= b \text{ and } x \geq 0
 \end{aligned}$$

12 dengan

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Pada notasi matriks, x menunjukkan matriks kolom berdimensi n , C^T menunjukkan matriks baris berdimensi n , A menunjukkan matriks berdimensi $m \times n$, dan b menunjukkan matriks kolom berdimensi m . Matriks $x \geq 0$ menunjukkan batasan non-negatif untuk setiap variabel keputusan x_i . Untuk mengubah masalah linear programming ke bentuk standar, maka setiap batasan perlu diubah. Perubahan batasan dibedakan berdasarkan tandanya, yaitu sebagai berikut:

• **Batasan dengan tanda pertidaksamaan \leq**

Pertidaksamaan pada batasan ini diubah ke dalam bentuk persamaan dengan menambahkan variabel positif s_i yang disebut *slack* pada sisi kiri. Lalu ditambahkan batasan non-negatif untuk variabel *slack* tersebut. Contoh:

$$x_1 + 2x_2 \leq 40$$

diubah menjadi

$$x_1 + 2x_2 + s_1 = 40$$

$$s_1 \geq 0$$

• **Batasan dengan tanda pertidaksamaan \geq**

Pertidaksamaan pada batasan ini diubah ke dalam bentuk persamaan dengan menambahkan variabel negatif e_i yang disebut *excess* pada sisi kiri. Lalu ditambahkan batasan non-negatif untuk variabel *excess* tersebut. Contoh:

$$x_1 + 2x_2 \geq 40$$

diubah menjadi

$$x_1 + 2x_2 - e_1 = 40$$

$$e_1 \geq 0$$

2.1.4 Variabel Basis dan Non-basis

Masalah linear programming $Ax = b$ terdiri dari m buah persamaan batasan dan n buah variabel keputusan. Masalah ini memiliki solusi yang disebut dengan solusi basis apabila membuat $(n - m)$ buah variabel keputusan bernilai 0 dan menyelesaikan m buah variabel keputusan lainnya. Sebanyak $(n - m)$ buah variabel yang dibuat bernilai 0 disebut dengan variabel non-basis. Sedangkan m buah variabel lainnya disebut dengan variabel basis. Berikut contoh sistem persamaan linear:

$$\begin{aligned} x_1 + x_2 &= 3 \\ -x_2 + x_3 &= -1 \end{aligned}$$

Pada sistem persamaan linear di atas, dipilih sebanyak $3 - 2 = 1$ (3 variabel dan 2 persamaan) buah variabel yang akan menjadi variabel non basis. Jika himpunan variabel non basis NBV = $\{x_3\}$, maka himpunan variabel basis BV = $\{x_1, x_2\}$. Solusi dari sistem persamaan tersebut dapat dicari dengan membuat setiap variabel NBV menjadi variabel non basis dan menyelesaikan variabel BV.

$$\begin{aligned} x_1 + x_2 &= 3 \\ -x_2 + 0 &= -1 \end{aligned}$$

Pada persamaan di atas didapatkan nilai $x_1 = 2$ dan $x_2 = 1$. Dengan demikian didapatkan solusi basis dengan nilai $x_1 = 2$, $x_2 = 1$, dan $x_3 = 0$.

2.1.5 Metode Simplex

Metode simplex merupakan metode yang digunakan untuk menyelesaikan masalah linear programming. Berikut ini merupakan langkah-langkah metode simplex:

1. Mengubah masalah linear programming ke dalam bentuk standar
2. Mencari *basic feasible solution* (*bfs*) dari bentuk standar.
3. Memeriksa apakah *bfs* pada saat ini sudah optimal.
4. Apabila *bfs* belum optimal, tentukan variabel non-basis mana yang akan menjadi variabel basis dan tentukan variabel basis mana yang akan menjadi variabel non-basis dengan tujuan mencari *bfs* baru yang dapat meningkatkan nilai fungsi objektif.
5. Lakukan Operasi Baris Elementer (OBE) untuk mencari *bfs* baru yang dapat meningkatkan nilai fungsi objektif. Lanjut ke langkah 3.

Fungsi objektif yang berbentuk

$$z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

ditulis dalam bentuk

$$z - c_1x_1 - c_2x_2 - \cdots - c_nx_n = 0$$

Bentuk fungsi objektif ini disebut sebagai baris 0.

1 Untuk memahami penyelesaian masalah linear programming menggunakan metode simplex,
2 akan digunakan sebuah contoh masalah linear programming sebagai berikut:

$$\begin{aligned}
 \max \quad & z = 60x_1 + 30x_2 + 20x_3 \\
 \text{s.t.} \quad & 8x_1 + 6x_2 + x_3 \leq 48 \\
 & 4x_1 + 2x_2 + 1.5x_3 \leq 20 \\
 & 2x_1 + 1.5x_2 + 0.5x_3 \leq 8 \\
 & x_2 \leq 5 \\
 & x_1 \geq 0 \\
 & x_2 \geq 0 \\
 & x_3 \geq 0
 \end{aligned}$$

3 Langkah pertama adalah mengubah masalah ke bentuk standar, sehingga masalah ini dalam
4 bentuk standar adalah sebagai berikut:

$$\begin{aligned}
 \max \quad & z = 60x_1 + 30x_2 + 20x_3 \\
 \text{s.t.} \quad & 8x_1 + 6x_2 + x_3 + s_1 = 48 \\
 & 4x_1 + 2x_2 + 1.5x_3 + s_2 = 20 \\
 & 2x_1 + 1.5x_2 + 0.5x_3 + s_3 = 8 \\
 & x_2 + s_4 = 5 \\
 & x_1, x_2, x_3, s_1, s_2, s_3, s_4 \geq 0
 \end{aligned}$$

5 Masalah dalam bentuk standar ini disajikan dalam bentuk tabel simplex seperti pada tabel 2.1.
6 Pada tabel ini, baris pertama (baris 0) merupakan baris bagi fungsi objektif dan baris lainnya
7 merupakan baris bagi variabel basis. Pada tabel simplex pertama, variabel basis terdiri dari variabel
8 *slack* dan variabel *excess*.

9 Tabel 2.1: Tabel simplex 0

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	rhs	$basic\ variable$
1	-60	-30	-20	0	0	0	0	0	$z = 0$
0	8	6	1	1	0	0	0	48	$s_1 = 48$
0	4	2	1.5	0	1	0	0	20	$s_2 = 20$
0	2	1.5	0.5	0	0	1	0	8	$s_3 = 8$
0	0	1	0	0	0	0	1	5	$s_4 = 5$

11 Kolom *right hand side* (rhs) menunjukkan nilai pada ruas kanan (nilai di sebelah kanan tanda
12 sama dengan). Kolom *basic variable* menunjukkan variabel basis beserta dengan nilainya. Kolom
13 *basic variable* juga menunjukkan solusi basis *feasible* (bfs) sehingga tabel 2.1 menunjukkan bfs
14 dengan nilai $z = 0, x_1 = 0, x_2 = 0, x_3 = 0, s_1 = 48, s_2 = 20, s_3 = 8$, dan $s_4 = 5$. Setiap variabel
15 yang tidak berada di dalam kolom *basic variable* menandakan bahwa variabel-variabel tersebut
16 merupakan variabel non-basis sehingga bernilai 0.

17 Langkah selanjutnya adalah memeriksa apakah bfs sudah optimal. Pada langkah ini akan
18 diperiksa apakah terdapat variabel negatif (untuk kasus maksimasi) pada baris 0 di setiap kolom
19 variabel. Pada tabel 2.1, masih terdapat variabel negatif, yaitu pada variabel x_1, x_2 , dan x_3 . Hal
20 ini menunjukkan bahwa bfs saat ini masih belum optimal.

Apabila *bfs* pada saat ini masih belum optimal, maka perlu dilakukan proses *pivotting*. Proses *pivotting* adalah proses penukaran variabel basis dengan variabel non-basis. Variabel yang akan menjadi variabel basis disebut dengan *entering variable* dan variabel basis yang akan digantikan disebut dengan *leaving variable*. *Entering variable* dipilih dengan cara memilih variabel yang bernilai paling negatif pada baris 0. Untuk mencari *leaving variable* perlu dilakukan tes rasio. Tes rasio membandingkan nilai pada nilai *rhs* dengan nilai pada kolom *entering variable*. Tes rasio hanya perlu dilakukan pada baris yang variabel pada kolom *entering variable* bernilai lebih besar dari 0. *Leaving variable* dipilih dengan cara memilih variabel pada baris yang menghasilkan rasio terkecil. Perpotongan antara kolom *entering variable* dengan baris *leaving variable* merupakan suatu elemen yang disebut *pivot element*.

Dalam contoh masalah, *bfs* pada saat ini masih belum optimal, sehingga perlu dilakukan proses *pivotting*. Variabel x_1 akan menjadi *entering variable* karena bernilai paling negatif pada baris 0. Setelah dilakukan tes rasio sesuai, variabel s_3 menjadi *leaving variable* karena memiliki hasil tes rasio terkecil. Tes rasio untuk *bfs* baru pada contoh masalah dapat dilihat pada tabel 2.2.

Tabel 2.2: Proses *pivotting* 1

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	rhs	<i>basic variable</i>	<i>ratio</i>
1	-60	-30	-20	0	0	0	0	0	$z = 0$	
0	8	6	1	1	0	0	0	48	$s_1 = 48$	$48/8 = 6$
0	4	2	1.5	0	1	0	0	20	$s_2 = 20$	$20/4 = 5$
0	2	1.5	0.5	0	0	1	0	8	$s_3 = 8$	$8/2 = 4$
0	0	1	0	0	0	0	1	5	$s_4 = 5$	

Setelah menentukan *entering variable* dan *leaving variable*, akan dilakukan Operasi Baris Elementer (OBE). OBE dilakukan untuk membuat *entering variable* menjadi variabel basis, sehingga *entering variable* pada baris *leaving variable* bernilai 1 dan variabel lain pada kolom *entering variable* bernilai 0. Setelah OBE dilakukan, baris *leaving variable* akan memiliki *entering variable* sebagai variabel basis.

Dalam contoh masalah, telah ditentukan *entering variable* dan *leaving variable*. Variabel yang menjadi *entering variable* adalah variabel x_1 yang berada pada kolom x_1 . Variabel yang menjadi *Leaving variable* adalah variabel s_3 yang berada pada baris 3. Dengan OBE, *entering variable* pada baris 3 akan dibuat bernilai 1 dan variabel lain pada kolom *entering variable* akan dibuat bernilai 0. Berikut ini urutan OBE yang dilakukan:

1. Untuk mendapatkan variabel x_1 bernilai 1 pada baris 3, maka baris 3' baru dapat dibentuk dari baris 3 lama yang dikalikan dengan $1/2$.

$$\frac{1}{2}R_3 \rightarrow R'_3$$

sehingga baris 3' baru menjadi

$$x_1 + 0.75x_2 + 0.25x_3 + 0.5s_3 = 4$$

2. Untuk mendapatkan variabel x_1 bernilai 0 pada baris 0, maka baris 0' baru dapat dibentuk dari baris 3' baru yang dikalikan 60 dan ditambahkan dengan baris 0 lama.

$$60R_3 + R_0 \rightarrow R'_0$$

sehingga baris 0' baru menjadi

$$z + 15x_2 - 5x_3 + 30s_3 = 240$$

3. Untuk mendapatkan variabel x_1 bernilai 0 pada baris 1, maka baris 1' baru dapat dibentuk dari baris 3' baru yang dikalikan -8 dan ditambahkan dengan baris 1 lama.

$$-8R_3 + R_1 \rightarrow R'_1$$

sehingga baris 1' baru menjadi

$$-x_3 + s_1 - 4s_3 = 16$$

4. Untuk mendapatkan variabel x_1 bernilai 0 pada baris 2, maka baris 2' baru dapat dibentuk dari baris 3' baru yang dikalikan -4 dan ditambahkan dengan baris 2 lama.

$$-4R_3 + R_2 \rightarrow R'_2$$

sehingga baris 2' baru menjadi

$$-x_2 + 0.5x_3 + s_2 - 2s_3 = 4$$

Karena nilai x_1 bernilai 0 pada baris 4, maka tidak perlu dilakukan OBE pada baris 4, sehingga baris 4' baru sama dengan baris 4 lama. *Entering variable* menggantikan *leaving variable* pada baris 3 sehingga x_1 menjadi variabel basis yang baru. Setelah melakukan OBE, maka didapatkan tabel simplex baru seperti pada tabel 2.3 yang menunjukkan *bfs* dengan nilai $z = 240, x_1 = 4, x_2 = 0, x_3 = 0, s_1 = 16, s_2 = 4, s_3 = 0$, dan $s_4 = 5$.

Tabel 2.3: Tabel simplex 1

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	rhs	<i>basic variable</i>
1	0	15	-5	0	0	30	0	240	$z = 240$
0	0	0	-1	1	0	-4	0	16	$s_1 = 16$
0	0	-1	0.5	0	1	-2	0	4	$s_2 = 4$
0	1	0.75	0.25	0	0	0.5	0	4	$x_1 = 4$
0	0	1	0	0	0	0	1	5	$s_4 = 5$

Setelah mendapatkan tabel simplex 2.3, langkah selanjutnya adalah memeriksa apakah *bfs* pada iterasi ini sudah optimal. Pada baris 0, terdapat nilai negatif, yaitu pada variabel x_3 , sehingga *bfs* ini belum optimal. Maka langkah selanjutnya adalah mencari *entering variable* dan *leaving variable*. *Entering variable* didapatkan dengan memilih variabel yang bernilai paling negatif pada baris 0, yaitu variabel x_3 . *Leaving variable* didapatkan dengan mencari rasio terkecil dari hasil tes rasio. Tes rasio dapat dilihat pada tabel 2.4. Variabel s_2 menghasilkan hasil tes rasio terkecil, sehingga variabel s_2 menjadi *leaving variable*.

Tabel 2.4: Proses *pivotting* 2

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	rhs	$basic\ variable$	$ratio$
1	0	15	-5	0	0	30	0	240	$z = 240$	
0	0	0	-1	1	0	-4	0	16	$s_1 = 16$	
0	0	-1	0.5	0	1	-2	0	4	$s_2 = 4$	$4/0.5 = 8$
0	1	0.75	0.25	0	0	0.5	0	4	$x_1 = 4$	$4/0.25 = 16$
0	0	1	0	0	0	0	1	5	$s_4 = 5$	

Selanjutnya dilakukan OBE untuk membuat *entering variable* x_3 menjadi variabel basis. Berikut ini urutan OBE yang dilakukan:

1. Untuk mendapatkan variabel x_3 bernilai 1 pada baris 2, maka baris 2'' baru dapat dibentuk dari baris 2' lama yang dikalikan dengan 2.

$$2R'_2 \rightarrow R''_2$$

sehingga baris 2'' baru menjadi

$$-2x_2 + x_3 + 2s_2 - 4s_3 = 8$$

2. Untuk mendapatkan variabel x_3 bernilai 0 pada baris 0, maka baris 0'' baru dapat dibentuk dari baris 2'' baru yang dikalikan 5 dan ditambahkan dengan baris 0' lama.

$$5R''_2 + R'_0 \rightarrow R''_0$$

sehingga baris 0'' baru menjadi

$$z + 5x_2 + 10s_2 + 10s_3 = 280$$

3. Untuk mendapatkan variabel x_3 bernilai 0 pada baris 1, maka baris 1'' baru dapat dibentuk dari baris 2'' baru yang ditambahkan dengan baris 1' lama.

$$R''_2 + R'_1 \rightarrow R''_1$$

sehingga baris 1" baru menjadi

$$-2x_2 + s_1 + 2s_2 - 8s_3 = 24$$

4. Untuk mendapatkan variabel x_3 bernilai 0 pada baris 3, maka baris 3" baru dapat dibentuk dari baris 2" baru yang dikalikan $-1/4$ dan ditambahkan dengan baris 3' lama.

$$-\frac{1}{4}R_2'' + R_3' \rightarrow R_3''$$

sehingga baris 3" baru menjadi

$$x_1 + 1.25x_2 - 0.5s_2 + 1.5s_3 = 2$$

Variabel x_3 sudah bernilai 0 pada baris 4, maka baris 4 yang baru sama dengan baris 4 yang lama. Variabel x_3 menjadi variabel basis baru yang menggantikan variabel s_2 . Setelah melakukan OBE, didapatkan tabel simplex 2.5 yang menunjukkan *bfs* dengan nilai $z = 280, x_1 = 2, x_2 = 0, x_3 = 8, s_1 = 24, s_2 = 0, s_3 = 0$, dan $s_4 = 5$.

Tabel 2.5: Tabel simplex 2

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	rhs	<i>basic variable</i>
1	0	5	0	0	10	10	0	280	$z = 280$
0	0	-2	0	1	2	-8	0	24	$s_1 = 24$
0	0	-2	1	0	2	-4	0	8	$x_3 = 8$
0	1	1.25	0	0	-0.5	1.5	0	2	$x_1 = 2$
0	0	1	0	0	0	0	1	5	$s_4 = 5$

Langkah selanjutnya adalah memeriksa apakah *bfs* saat ini sudah optimal. Pada baris 0, tidak ditemukan variabel bernilai negatif, sehingga *bfs* pada tahap ini sudah optimal. Dengan demikian masalah ini memiliki solusi optimal bernilai 280 dengan variabel $x_1 = 2, x_2 = 0, x_3 = 8$, dan $x_4 = 0$.

Contoh yang dibahas sebelumnya merupakan contoh masalah linear programming dalam kasus maksimisasi. Untuk menyelesaikan masalah linear programming pada kasus minimasi, dapat dilakukan dengan mengubah tujuan minimasi menjadi tujuan maksimasi. Perubahan tujuan ini dilakukan dengan mengalikan fungsi tujuan dengan -1. Contoh:

$$\min z = 2x_1 - 3x_2$$

$$s.t. x_1 + x_2 \leq 4$$

$$x_1 - x_2 \leq 6$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

Contoh tersebut merupakan contoh masalah linear programming dengan tujuan minimasi. Untuk menyelesaikannya, fungsi objektif dikalikan dengan -1, sehingga tujuan berubah menjadi maksimasi. Contoh masalah berubah menjadi:

$$\max -z = -2x_1 + 3x_2$$

$$s.t. x_1 + x_2 \leq 4$$

$$x_1 - x_2 \leq 6$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

Tabel 2.6 menunjukkan tabel simplex awal untuk contoh masalah linear programming kasus minimasi. Tabel 2.7 menunjukkan tabel simplex yang menghasilkan *bfs* optimal dengan nilai $-z = 12, x_1 = 0, x_2 = 4, s_1 = 0$, dan $s_2 = 10$. Dengan demikian, didapatkan solusi optimal sebesar -12 dengan variabel $x_1 = 0$ dan $x_2 = 4$.

Tabel 2.6: Tabel simplex 0 untuk kasus minimasi

$-z$	x_1	x_2	s_1	s_2	rhs	<i>basic variable</i>	<i>ratio</i>
1	2	-3	0	0	0	$-z = 0$	
0	1	1	1	0	4	$s_1 = 4$	$4/1 = 4$
0	1	-1	0	1	6	$s_2 = 6$	

Tabel 2.7: Tabel simplex 1 untuk kasus minimasi

$-z$	x_1	x_2	s_1	s_2	rhs	<i>basic variable</i>
1	5	0	3	0	12	$-z = 12$
0	1	1	1	0	4	$x_2 = 4$
0	2	0	1	1	10	$s_2 = 10$

2.2 Integer Programming [1]

Integer programming merupakan teknik lanjutan dari linear programming di mana setiap variabel pada solusi optimal harus berupa bilangan bulat. Terdapat beberapa metode yang dapat digunakan untuk menyelesaikan masalah integer programming, salah satunya adalah metode *branch and bound*.

2.2.1 Metode *Branch and Bound*

Branch and bound merupakan metode penyelesaian masalah integer programming dengan memecah masalah menjadi sub-sub masalah. Pemecahan masalah akan terus dilakukan hingga didapatkan solusi optimal yang terdiri dari variabel berupa bilangan bulat. Berikut langkah-langkah metode *branch and bound*:

1. Selesaikan masalah menggunakan metode simplex. Apabila solusi optimal yang didapatkan terdiri dari variabel berupa bilangan bulat, maka proses berhenti pada tahap ini. Solusi ini menjadi solusi optimal dalam permasalahan ini. Apabila sebaliknya, maka dilanjutkan ke tahap selanjutnya. Solusi pada tahap ini akan menjadi solusi batas atas yang akan digunakan sebagai pembanding di tahap-tahap selanjutnya.
2. Tentukan satu variabel yang nilainya pada solusi tidak berupa bilangan bulat. Variabel ini akan menjadi variabel terpilih yang akan digunakan di tahap selanjutnya.

3. Buat 2 sub masalah baru. Sub-sub masalah tersebut merupakan masalah yang sama dengan adanya sebuah tambahan batasan baru.
 - (a) Sub masalah pertama memiliki tambahan batasan baru yang menyatakan bahwa variabel terpilih harus bernilai lebih kecil atau sama dengan pembulatan ke bawah dari nilai variabel terpilih.
 - (b) Sub masalah kedua memiliki tambahan batasan baru yang menyatakan bahwa variabel terpilih harus bernilai lebih besar atau sama dengan pembulatan ke atas dari nilai variabel terpilih.
4. Selesaikan setiap sub masalah menggunakan metode simplex dan periksa solusinya.
 - (a) Apabila setiap variabel pada solusi berupa bilangan bulat, maka bandingkan solusi ini dengan solusi batas atas. Apabila solusi ini sama dengan batas atas, maka solusi ini merupakan solusi optimal dalam permasalahan ini dan proses berhenti pada tahap ini. Apabila solusi lebih buruk dari solusi batas atas, maka bandingkan dengan solusi batas bawah. Apabila solusi ini lebih baik daripada solusi batas bawah, maka jadikanlah solusi ini sebagai solusi batas bawah yang baru.
 - (b) Apabila setiap variabel pada solusi tidak berupa bilangan bulat, maka periksa apakah solusi ini lebih baik daripada solusi batas bawah. Apabila lebih baik, maka dilanjutkan pada tahap ke-2 untuk memecah sub masalah ini. Apabila tidak lebih baik, maka sub masalah ini tidak perlu dipecah lagi karena tidak akan menghasilkan solusi yang lebih baik.
 - (c) Apabila sub masalah tidak dapat diselesaikan (*infeasible*), maka sub masalah ini tidak perlu dilanjutkan lagi.

2.3 Kakas *lp_solve*

Kakas *lp_solve* merupakan kakas perangkat lunak yang berguna untuk menyelesaikan masalah linear programming¹. Kakas ini dapat menerima masalah linear programming atau masalah integer programming. Kakas *lp_solve* menggunakan metode *branch and bound* untuk menyelesaikan masalah integer programming. Kakas ini dibangun dengan bahasa pemrograman ANSI C yang dapat dikompilasi dalam lingkungan Unix dan Windows. Untuk menggunakan kakas *lp_solve* dalam lingkungan pemrograman Java, kakas *lp_solve* menyediakan kakas penghubung dalam bahasa pemrograman Java sehingga aplikasi Java dapat menggunakan kakas *lp_solve*. Berikut ini merupakan contoh penggunaan kakas *lp_solve* dalam pemrograman Java:

Listing 2.1: Demo.java

```
import lpsolve.*;

public class Demo {
```

¹<http://lpsolve.sourceforge.net/5.5/>

```
1  public static void main(String[] args) {
2      try {
3          // Create a problem with 4 variables and 0 constraints
4          LpSolve solver = LpSolve.makeLp(0, 4);
5
6          // add constraints
7          solver.strAddConstraint("3_2_2_1", LpSolve.LE, 4);
8          solver.strAddConstraint("0_4_3_1", LpSolve.GE, 3);
9
10         // set objective function
11         solver.strSetObjFn("2_3_-2_3");
12
13         // solve the problem
14         solver.solve();
15
16         // print solution
17         System.out.println("Value of objective function: "
18             + solver.getObjective());
19         double[] var = solver.getPtrVariables();
20         for (int i = 0; i < var.length; i++) {
21             System.out.println("Value of var[" + i + "] = " + var[i]);
22         }
23
24         // delete the problem and free memory
25         solver.deleteLp();
26     }
27     catch (LpSolveException e) {
28         e.printStackTrace();
29     }
30 }
31 }
```


BAB 3

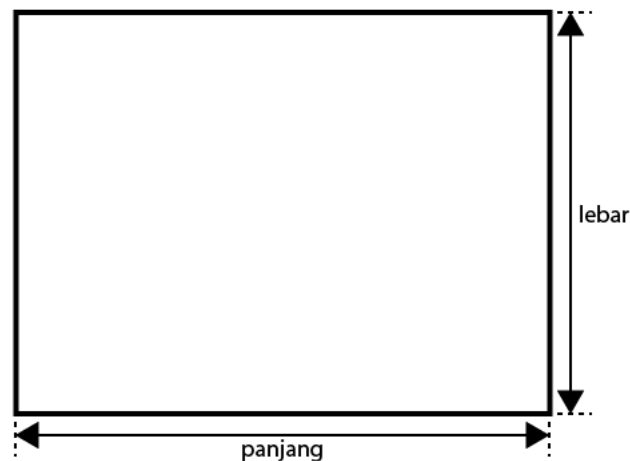
ANALISIS

3.1 Pemodelan Masalah

Masalah yang dibahas di skripsi ini perlu dirumuskan terlebih dahulu agar dapat diselesaikan. Masalah akan dipecah menjadi beberapa elemen sehingga fungsi dari setiap elemen dapat dipahami lebih mudah. Setiap elemen akan memiliki keterhubungan satu dengan yang lainnya sehingga apabila disatukan akan merepresentasikan masalah yang dibahas. Dengan merumuskan masalah, maka masalah dapat dimodelkan menjadi elemen-elemen yang dapat dipahami secara konkret baik bagi penulis maupun pembaca.

3.1.1 Ruangan

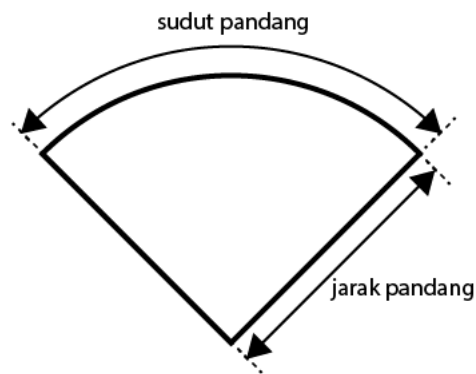
Dalam masalah, terdapat sebuah ruangan yang harus dicakup sepenuhnya oleh kamera-kamera CCTV. Ruangan dapat diartikan sebagai sebuah bidang 3 dimensi yang memiliki rongga di dalamnya. Ruangan ini pada umumnya memiliki bentuk yang beragam sesuai dengan arsitekturnya pada saat dibangun. Berbeda dengan ruangan tersebut, ruangan yang dibahas dalam masalah ini memiliki ukuran dimensi dan bentuk yang dibatasi. Ruangan tidak dimodelkan dalam bentuk 3 dimensi, tetapi dalam bidang 2 dimensi yang berbentuk persegi panjang. Dengan pemodelan ini, ruangan akan memiliki 2 parameter utama yang menentukan ukuran ruangan, yaitu ukuran panjang dan ukuran lebar. Ukuran panjang dan ukuran lebar ini memiliki satuan berupa sentimeter(cm). Pemodelan ruangan dapat dipahami lebih lanjut pada gambar 3.1.



Gambar 3.1: Pemodelan ruangan

3.1.2 Kamera CCTV

Kamera CCTV yang beredar di pasaran memiliki spesifikasi yang sangat beragam. Dalam skripsi ini, jenis kamera CCTV yang digunakan dibatasi sehingga tidak bervariasi, yaitu hanya menggunakan 1 jenis kamera CCTV saja. Kamera CCTV sendiri memiliki berbagai parameter seperti jarak pandang, lebar sudut pandang, tingkat resolusi, dan parameter-parameter lainnya. Dalam masalah ini, terdapat 2 parameter yang digunakan, yaitu jarak pandang efektif dan lebar sudut pandang. Jarak pandang efektif merupakan jarak pandang terjauh kamera CCTV untuk mengenali suatu objek yang akan dipantau. Jarak pandang efektif dinyatakan dalam ukuran bersatuan sentimeter(cm) dan lebar sudut pandang dinyatakan dalam ukuran derajat. Pemodelan kamera CCTV dapat dipahami lebih lanjut pada gambar 3.2.



Gambar 3.2: Pemodelan kamera CCTV

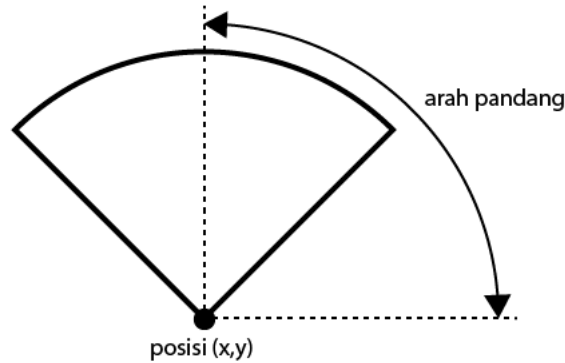
3.1.3 Penempatan Kamera CCTV

Setiap kamera CCTV dapat ditempatkan di mana saja selama berada di dalam ruangan. Penempatan kamera CCTV terdiri dari 2 komponen utama, yaitu posisi penempatan dan arah pandang. Posisi dan arah pandang akan mempengaruhi daerah cakupan kamera CCTV yang bersangkutan. Posisi penempatan dimodelkan dengan menggunakan sistem koordinat kartesius sehingga dapat dinyatakan dalam bentuk koordinat (x,y) . Sumbu y pada sistem koordinat yang digunakan akan dibalik agar sesuai dengan lingkungan grafis pada layar komputer. Arah pandang kamera CCTV dinyatakan sebagai besar sudut perpotongan antara garis tengah kamera CCTV dengan garis 0° yang dituliskan dalam satuan derajat. Dengan demikian, penempatan kamera CCTV terdiri atas posisi penempatan dan arah pandang yang dituju. Pemodelan penempatan kamera CCTV dapat dipahami lebih lanjut pada gambar 3.3.

3.1.4 Daerah Cakupan

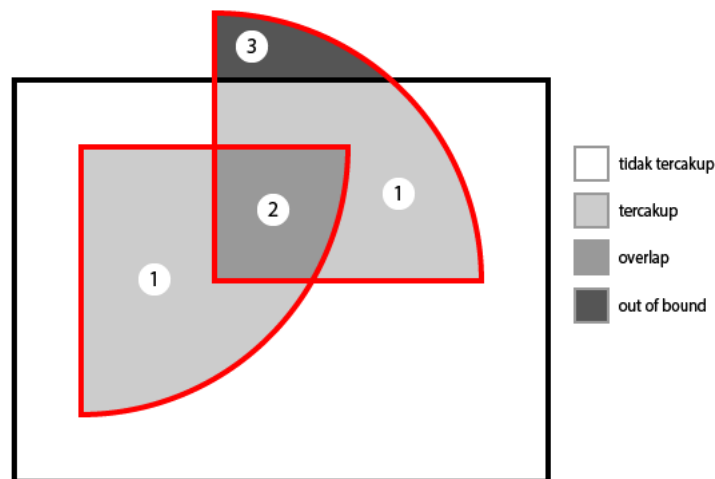
Daerah cakupan kamera CCTV memiliki bentuk yang tidak sederhana sehingga menjadi sulit ketika akan diolah. Terdapat 3 kasus yang menjelaskan daerah cakupan dengan bentuk yang tidak sederhana ini, yaitu:

1. Kasus ketika menghitung luas daerah yang tidak *overlap* dan tidak *out of bound*. Contoh kasus ini dapat dilihat pada gambar 3.4 pada daerah yang dilabeli nomor 1.



Gambar 3.3: Pemodelan penempatan kamera CCTV

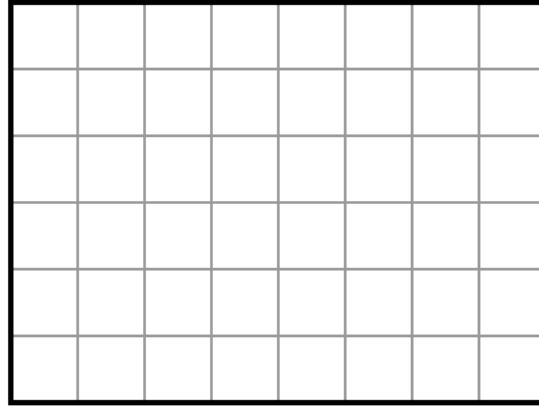
- 1 2. Kasus ketika menghitung luas daerah *overlap*. Contoh kasus ini dapat dilihat pada gambar 3.4
- 2 pada daerah yang dilabeli nomor 2.
- 3 3. Kasus ketika menghitung luas daerah *out of bound*. Contoh kasus ini dapat dilihat pada
- 4 gambar 3.4 pada daerah yang dilabeli nomor 3.



Gambar 3.4: Contoh masalah yang memiliki kasus bentuk daerah tidak sederhana

Daerah-daerah dalam ketiga kasus tersebut dapat berbentuk tidak sederhana, sehingga sulit untuk diolah. Dengan adanya ketiga kasus ini, maka daerah cakupan perlu didefinisikan dan dimodelkan lebih lanjut agar kasus tersebut dapat dihindari. Ruang dapat dimodelkan lebih lanjut sehingga berbentuk grid point seperti pada gambar 3.5. Grid point akan memecah ruangan ke dalam bagian-bagian yang lebih kecil yang disebut dengan cell.

Cell tidak selalu berbentuk persegi, namun dapat berbentuk persegi panjang. Hal ini dikarenakan susunan cell-cell harus menghasilkan ukuran yang sama dengan ukuran ruangan. Untuk menentukan ukuran cell, ditentukan sebuah ukuran yang menyatakan ukuran terbesar yang dapat dimiliki cell. Ukuran ini akan digunakan untuk menentukan ukuran cell yang apabila disusun akan menghasilkan ukuran ruangan. Berikut ini merupakan rumus yang digunakan untuk mencari ukuran cell:

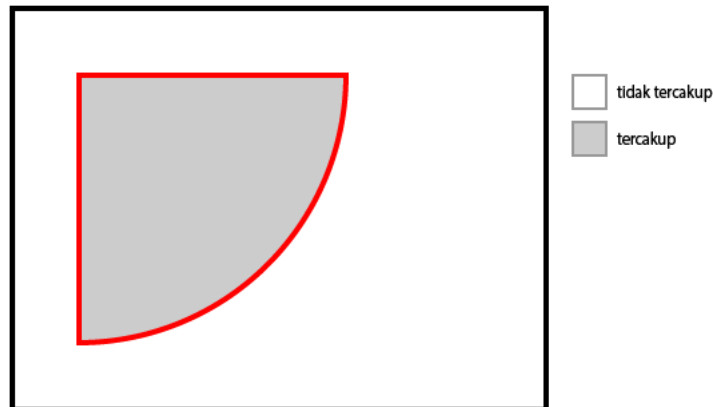


Gambar 3.5: Pemodelan ruangan dalam bentuk grid point

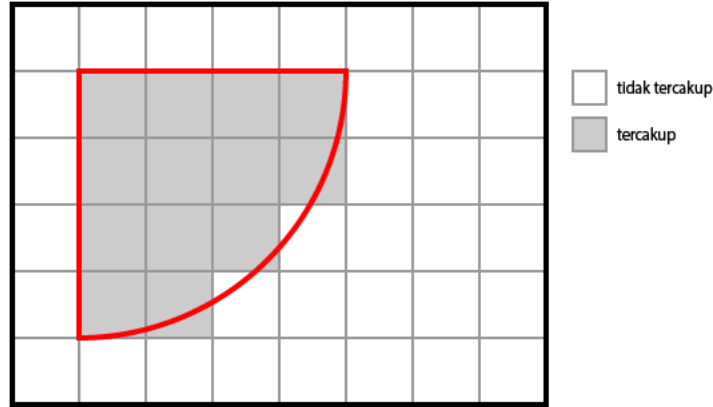
$$\begin{aligned} \text{columns} &= \left\lceil \frac{\text{room width}}{\text{max cell size}} \right\rceil \\ \text{rows} &= \left\lceil \frac{\text{room length}}{\text{max cell size}} \right\rceil \\ \text{cell width} &= \frac{\text{room width}}{\text{columns}} \\ \text{cell length} &= \frac{\text{room length}}{\text{rows}} \end{aligned}$$

1 Dengan rumus ini, akan didapatkan ukuran panjang dan ukuran lebar cell yang tidak melebihi
 2 ukuran terbesar cell. Apabila ukuran cell ini dikalikan dengan jumlah kolom dan jumlah baris grid
 3 point, maka akan didapatkan ukuran yang sama dengan ukuran ruangan.

4 Setiap cell berfungsi merepresentasikan sebagian daerah dalam ruangan yang berupa satu
 5 kesatuan sehingga kasus bentuk daerah yang tidak sederhana dapat dihindari. Dengan pemodelan
 6 menggunakan grid point, maka daerah cakupan dari suatu kamera CCTV dapat dinyatakan dalam
 7 bentuk kumpulan cell. Gambar 3.6 dan gambar 3.7 menunjukkan perbandingan daerah cakupan
 8 kamera CCTV ketika sebelum dan sesudah dimodelkannya ruangan dalam bentuk grid point.



Gambar 3.6: Daerah cakupan sebelum pemodelan grid point



Gambar 3.7: Daerah cakupan sesudah pemodelan grid point

1 Untuk mencari kumpulan cell yang dicakup suatu penempatan kamera CCTV, maka setiap cell
 2 harus diperiksa apakah dapat tercakup oleh penempatan tersebut. Pada setiap cell akan ditentukan
 3 sebuah titik tengah yang berada di tengah-tengah cell. Titik tengah ini akan digunakan dalam
 4 pemeriksaan ketercakup cell. Pemeriksaan terdiri dari pemeriksaan jarak cell dan pemeriksaan
 5 sudut rotasi cell. Jarak antara titik tengah cell dengan titik penempatan kamera CCTV harus
 6 lebih kecil daripada jarak pandang kamera CCTV. Sudut rotasi cell harus berada di antara sudut
 7 pandang kamera CCTV. Terdapat rumus yang digunakan untuk mendapatkan sudut rotasi ini,
 8 yaitu sebagai berikut:

$$\text{atan2}(x, y) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0 \end{cases}$$

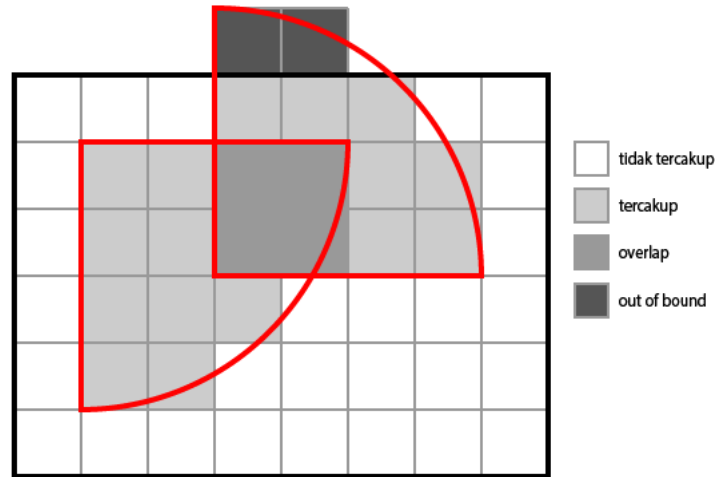
$$\theta = \text{atan2}(y_{cam} - y_{cell}, x_{cell} - x_{cam})$$

9 Pada rumus tersebut, titik (x_{cell}, y_{cell}) menunjukkan titik tengah cell dan titik (x_{cam}, y_{cam})
 10 menunjukkan titik penempatan kamera CCTV. Sudut rotasi cell (θ) akan dibandingkan dengan
 11 sudut mulai dan sudut akhir dari sudut pandang kamera CCTV. Sudut rotasi harus berada di antara
 12 kedua sudut tersebut. Apabila kedua pemeriksaan tersebut berhasil dilalui, maka cell dinyatakan
 13 tercakup oleh kamera CCTV. Apabila sebaliknya, maka cell dinyatakan tidak tercakup oleh kamera
 14 CCTV. Dengan pemeriksaan ini, maka cakupan kamera CCTV dapat dicari dan dinyatakan dalam
 15 bentuk kumpulan cell.

16 3.1.5 *Overlap dan Out of Bound*

17 Dengan pemodelan grid point, perhitungan tingkat *overlap* dan *out of bound* dapat dilakukan
 18 dengan membandingkan jumlah cell. *Overlap cell* adalah cell yang dicakup oleh lebih dari 1 kamera
 19 CCTV. *Out of bound cell* adalah cell yang tercakup oleh kamera CCTV, tetapi berada di luar
 20 ruangan. Gambar 3.8 menggambarkan *overlap cell* dan *out of bound cell*. Tingkat *overlap* dan

- 1 *out of bound* dapat dihitung dengan mengtotalkan jumlah cell yang dicakup dari setiap kamera
- 2 CCTV dan membaginya dengan jumlah cell yang berada di dalam ruangan. Perhitungan tingkat
- 3 *overlap* dan *out of bound* hanya bisa dilakukan apabila seluruh cell dalam ruangan telah tercakup
- 4 sepenuhnya.



Gambar 3.8: Daerah overlap dan out of bound

5 3.2 Penyelesaian Masalah

- 6 Dalam masalah ini, terdapat tujuan yang akan dicapai, yaitu mendapatkan penempatan-penempatan
- 7 kamera CCTV yang dapat mencakup seluruh daerah pada ruangan. Kamera-kamera CCTV
- 8 tentu dapat ditempatkan dimana saja hingga seluruh daerah pada ruangan tercakup sepenuhnya.
- 9 Penempatan dengan cara ini tentu tidak efektif karena jumlah kamera CCTV tidak selalu berjumlah
- 10 minimum sehingga diperlukan metode lainnya untuk menyelesaikan masalah ini. Salah satu
- 11 cara yang digunakan untuk menyelesaikan masalah ini adalah dengan menggunakan teknik linear
- 12 programming.

13 3.2.1 Variabel Keputusan

- 14 Pada awalnya ditentukan seluruh kemungkinan penempatan kamera CCTV sehingga setiap cell pada
- 15 ruangan dicakup oleh minimal 1 kamera CCTV. Setiap penempatan kamera CCTV memiliki posisi
- 16 dan arah pandang masing-masing. Dari seluruh kemungkinan ini akan dicari himpunan bagian yang
- 17 dimana penempatan-penempatannya dapat mencakup seluruh daerah pada ruangan. Setiap kemung-
- 18 kinan penempatan memiliki 2 kemungkinan, yaitu diterapkan sebagai bagian dari solusi atau tidak.
- 19 Dalam penyelesaian menggunakan teknik linear programming, setiap kemungkinan penempatan
- 20 akan menjadi variabel keputusan. Variabel keputusan $x_{ij\theta}$ akan merujuk pada penempatan kamera
- 21 CCTV pada posisi (i, j) dengan arah pandang θ . Karena setiap penempatan kamera CCTV memiliki
- 22 2 kemungkinan untuk diterapkan, maka setiap variabel keputusan dapat dinyatakan dengan nilai 0
- 23 atau 1. Apabila suatu variabel keputusan bernilai 1, maka penempatan kamera CCTV yang dirujuk
- 24 akan diterapkan sebagai bagian dari solusi. Apabila bernilai 0, maka penempatan kamera CCTV
- 25 yang dirujuk tidak akan diterapkan. Karena variabel keputusan hanya bisa bernilai 0 dan 1 saja,

1 masalah ini tidak diselesaikan menggunakan teknik linear programming biasa karena hasil dapat
 2 bernilai pecahan. Masalah ini perlu diselesaikan menggunakan teknik integer programming agar
 3 hasil dari setiap variabel keputusan hanya berupa bilangan bulat. Variabel keputusan dinyatakan
 4 dalam notasi berikut:

$$x_{ij\theta} = \begin{cases} 1 & \text{jika kamera CCTV ditempatkan pada posisi } (i, j) \\ & \text{dengan arah pandang } \theta \\ 0 & \text{jika kamera CCTV tidak ditempatkan} \end{cases}$$

5 3.2.2 Fungsi Tujuan

6 Solusi yang diharapkan dari masalah ini adalah mendapatkan penempatan-penempatan kamera
 7 CCTV yang paling minimum untuk mencakup seluruh daerah dalam ruangan. Dengan semakin
 8 sedikitnya penempatan-penempatan kamera CCTV yang digunakan, tingkat overlap dan out of
 9 bound juga akan semakin kecil. Sebelumnya telah diketahui bahwa setiap variabel keputusan hanya
 10 bisa bernilai 0 atau 1 saja. Jumlah penempatan kamera CCTV pun dapat dicari dengan menjum-
 11 lahkan seluruh variabel keputusan. Fungsi tujuan dalam linear programming dapat dinyatakan
 12 sebagai jumlah penempatan kamera CCTV akan diminimumkan. Fungsi tujuan dituliskan dalam
 13 notasi berikut:

$$\min z = \sum_{\theta=0}^{s_{\theta}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{ij\theta}$$

14 3.2.3 Batasan

15 Selain mencari jumlah kamera CCTV yang minimal, seluruh daerah pada ruangan harus tercakup
 16 sepenuhnya. Berdasarkan pemodelan, ruangan dimodelkan dalam bentuk grid point sehingga
 17 daerah pada ruangan dimodelkan dalam bentuk cell. Agar seluruh daerah pada ruangan tercakup
 18 sepenuhnya, maka setiap cell harus dicakup oleh minimal 1 kamera CCTV. Terdapat sebuah fungsi
 19 biner yang digunakan untuk mengetahui apakah suatu penempatan kamera CCTV dapat mencakup
 20 suatu cell. Fungsi biner ini akan menghasilkan nilai 1 apabila suatu penempatan kamera CCTV
 21 dapat mencakup suatu cell. Apabila sebaliknya, maka fungsi biner ini akan menghasilkan nilai 0.
 22 Fungsi tersebut ditulis sebagai berikut:

$$cov(i, j, \theta, p, q) = \begin{cases} 1 & \text{jika kamera CCTV pada posisi } (i, j) \text{ dengan arah pandang } \theta \\ & \text{dapat mencakup cell } (p, q) \\ 0 & \text{jika sebaliknya} \end{cases}$$

23 Dalam bentuk linear programming akan ditambahkan batasan yang menyatakan bahwa setiap
 24 cell harus dicakup oleh minimal 1 penempatan kamera CCTV. Fungsi biner sebelumnya akan
 25 digunakan untuk menyatakan hubungan ketercakup cell dengan penempatan kamera CCTV.
 26 Batasan tersebut ditulis sebagai berikut:

$$\sum_{\theta=0}^{s_{\theta}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{i,j,\theta} \times cov(i, j, \theta, p, q) \geq 1$$

$$0 \leq p \leq (s_p - 1), 0 \leq q \leq (s_q - 1)$$

3.2.4 Bentuk Masalah Linear Programming

Masalah yang dibahas di dalam skripsi ini dapat diubah ke dalam bentuk yang dapat diselesaikan dengan teknik linear programming. Setiap penempatan kamera CCTV pada ruangan menjadi variabel-variabel keputusan yang menunjukkan apakah akan diterapkan sebagai solusi atau tidak. Fungsi tujuan berfungsi untuk menyatakan bahwa penempatan-penempatan kamera CCTV harus berjumlah minimum sehingga sesuai dengan solusi yang diharapkan. Selain mendapatkan jumlah penempatan kamera CCTV yang minimum, terdapat batasan-batasan yang menyatakan bahwa seluruh daerah pada ruangan juga harus tercakup sepenuhnya. Apabila digabungkan, maka bentuk masalah ini dalam bentuk linear programming adalah seperti berikut:

$$\begin{aligned} \min z &= \sum_{\theta=0}^{s_{\theta}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{ij\theta} \\ \text{s.t.} \quad &\sum_{\theta=0}^{s_{\theta}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{i,j,\theta} \times cov(i, j, \theta, p, q) \geq 1 \\ &0 \leq p \leq (s_p - 1), 0 \leq q \leq (s_q - 1) \\ &x_{ij\theta} \in \{0, 1\} \end{aligned}$$

3.3 Analisis Kebutuhan Perangkat Lunak

Pada subbab ini, akan dijelaskan aksi-aksi yang dapat dilakukan pengguna terhadap perangkat lunak melalui diagram *use case* dan skenario-skenario. Selain penjelasan aksi-aksi, terdapat juga diagram kelas sederhana yang akan dikembangkan lebih lanjut pada tahap perancangan perangkat lunak.

3.3.1 Diagram Use Case

Dalam perangkat lunak yang dibangun, hanya terdapat 1 jenis pengguna. Diagram *use case* pada gambar 3.9 menunjukkan aktor dan aksi-aksi yang dapat dilakukannya.

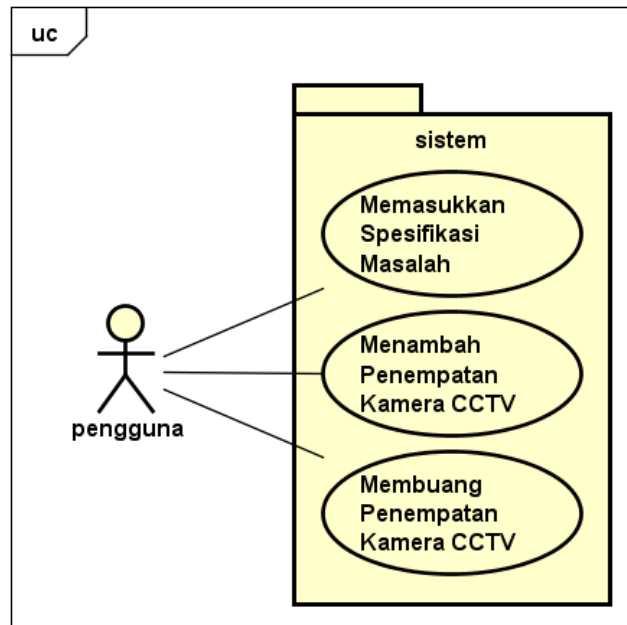
Berikut ini skenario dari setiap aksi pada diagram *use case*:

- Skenario: Memasukkan Spesifikasi Masalah

- Aktor: Pengguna

- Langkah:

1. Aktor memasukkan spesifikasi masalah yang terdiri dari ukuran ruangan dan spesifikasi kamera CCTV dan dilanjutkan dengan menekan tombol "submit".

Gambar 3.9: Diagram *use case*

2. Sistem menampilkan tampilan simulasi penempatan kamera CCTV.

• Skenario: **Menambah Penempatan Kamera CCTV**

– Aktor: Pengguna

– Langkah:

1. Aktor memilih posisi penempatan kamera CCTV dan mengisi sudut arah pandang penempatan kamera CCTV dan dilanjutkan dengan menekan tombol "add camera".
2. Sistem menambahkan penempatan ke dalam simulasi dan memperbaharui tampilan simulasi.

• Skenario: **Membuang Penempatan Kamera CCTV**

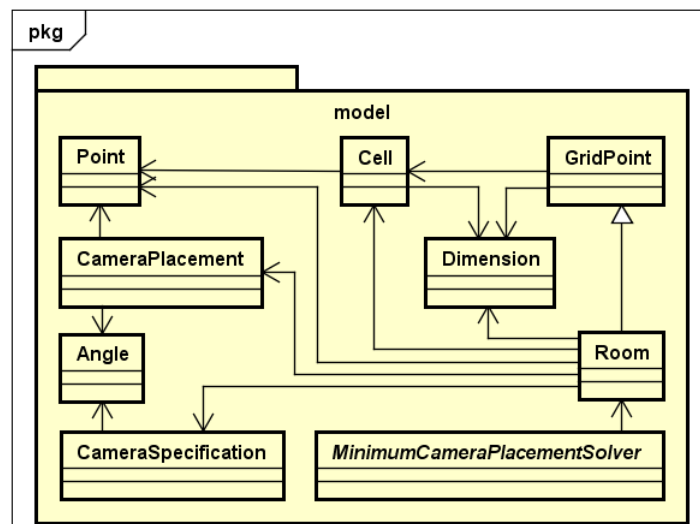
– Aktor: Pengguna

– Langkah:

1. Aktor memilih penempatan kamera CCTV yang akan dibuang dan menekan tombol "remove".
2. Sistem membuang penempatan dari simulasi dan memperbaharui tampilan simulasi.

3.3.2 Diagram Kelas

Pada bagian ini terdapat diagram kelas sederhana yang menunjukkan kelas-kelas yang akan digunakan untuk merancang perangkat lunak. Diagram kelas sederhana dapat dilihat pada gambar 3.10.



Gambar 3.10: Diagram kelas sederhana

BAB 4

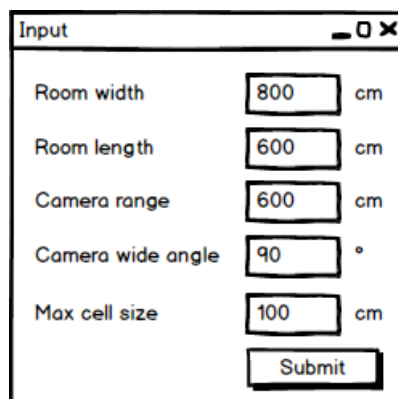
PERANCANGAN

4.1 Perancangan Antarmuka

Dalam perangkat lunak yang dibangun, tampilan yang digunakan adalah tampilan antarmuka grafis. Tampilan antarmuka ini berguna untuk mempermudah interaksi pengguna dengan perangkat lunak. Selain mempermudah, tampilan ini digunakan agar dapat menghasilkan visualisasi penempatan kamera CCTV sehingga pengguna dapat memahami penempatan-penempatan tersebut. Pada bagian ini akan dijelaskan bentuk dari setiap antarmuka. Berikut bentuk antarmuka-antarmuka tersebut:

- Antarmuka: **Penerima Masukan**

Antarmuka ini berfungsi untuk menerima masukan dari pengguna. Antarmuka ini dapat dilihat pada gambar 4.1. Pada antarmuka ini terdapat kolom-kolom masukan yang dapat diisi oleh pengguna. Pengguna dapat mengisi ukuran ruangan, spesifikasi kamera CCTV, dan ukuran terbesar cell pada kolom-kolom tersebut. Apabila pengguna sudah yakin dengan masukannya, maka pengguna dapat menekan tombol "*submit*" yang akan mengarahkan pengguna pada antarmuka simulasi penempatan kamera CCTV.



The image shows a window titled 'Input' with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are five rows of input fields, each with a label on the left and a unit on the right. The first row is 'Room width' with a value of '800' and unit 'cm'. The second row is 'Room length' with a value of '600' and unit 'cm'. The third row is 'Camera range' with a value of '600' and unit 'cm'. The fourth row is 'Camera wide angle' with a value of '90' and unit '°'. The fifth row is 'Max cell size' with a value of '100' and unit 'cm'. Below these input fields is a 'Submit' button.

Label	Value	Unit
Room width	800	cm
Room length	600	cm
Camera range	600	cm
Camera wide angle	90	°
Max cell size	100	cm

Submit

Gambar 4.1: Perancangan antarmuka penerima masukan

- Antarmuka: **Simulasi Penempatan Kamera CCTV**

Antarmuka ini berfungsi untuk melakukan kegiatan simulasi penempatan kamera CCTV. Antarmuka ini dapat dilihat pada gambar 4.2. Di dalam antarmuka ini terdapat 3 bagian, yaitu:

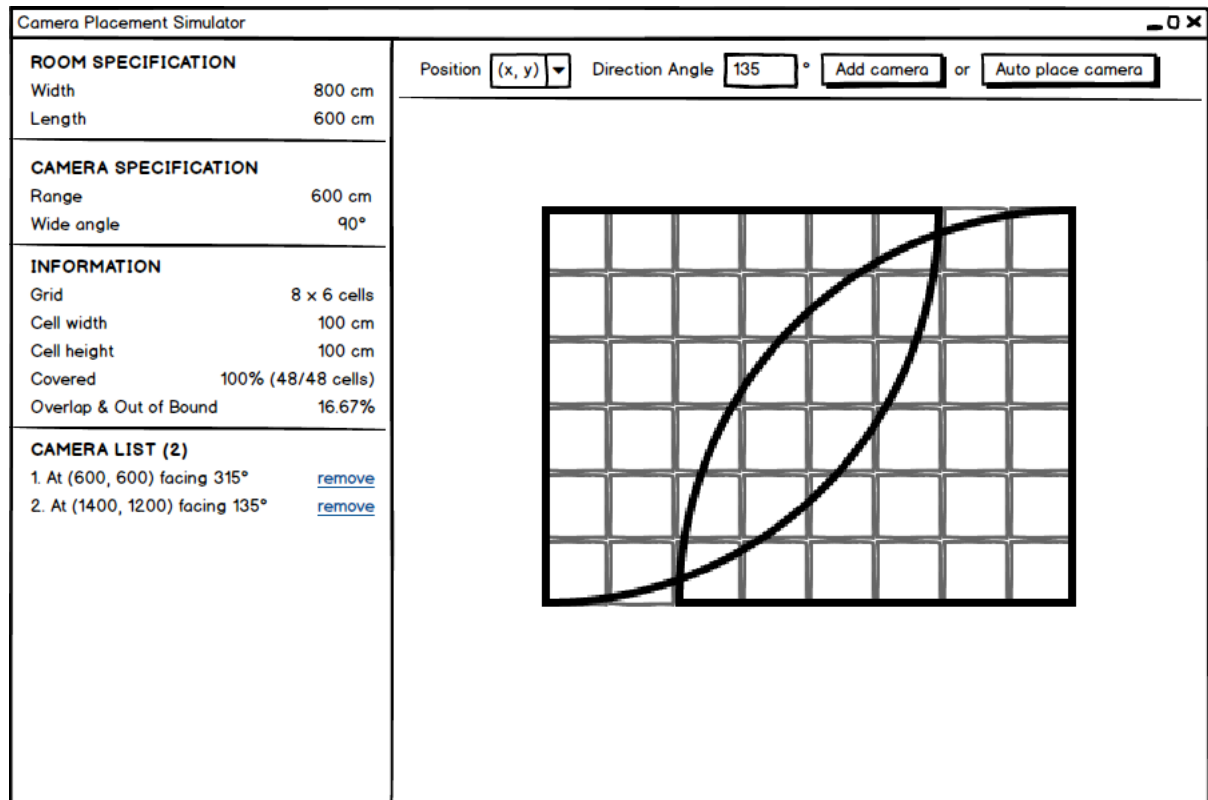
- Panel informasi yang berada di bagian kiri antarmuka.

Panel ini berfungsi untuk memberikan informasi-informasi yang terdiri dari ukuran ruangan, spesifikasi kamera CCTV, informasi simulasi, dan daftar penempatan kamera CCTV. Pada bagian informasi simulasi terdapat informasi persentase ketercakupannya dan persentase tingkat *overlap* dan *out of bound*. Pada bagian daftar penempatan kamera CCTV, terdapat penempatan-penempatan yang sedang diterapkan dalam simulasi. Pada setiap penempatan terdapat tombol "*remove*" yang apabila ditekan akan membuang penempatan tersebut.

- Panel visualisasi penempatan kamera CCTV yang berada di bagian kanan antarmuka. Panel ini berfungsi untuk menampilkan visualisasi penempatan kamera CCTV sesuai dengan penempatan-penempatan yang sedang diterapkan pada simulasi.
- Panel penambah kamera CCTV yang berada di atas panel visualisasi. Panel ini berfungsi untuk melakukan penempatan kamera CCTV baru. Pengguna dapat melakukan 2 jenis penambahan penempatan, yaitu penambahan sesuai dengan keinginan pengguna dan penambahan secara otomatis. Apabila pengguna ingin melakukan penambahan sesuai dengan keinginan, maka pengguna dapat memilih posisi dan mengisi sudut arah pandang dan dilanjutkan dengan menekan tombol "*add camera*". Apabila pengguna ingin melakukan penambahan secara otomatis, maka pengguna dapat menekan tombol "*auto place camera*". Dengan melakukan penambahan secara otomatis, sistem akan mencari penempatan-penempatan tersedikit yang dapat mencakup seluruh cell yang belum tercakup. Selama proses penambahan otomatis berjalan, tombol "*add camera*" dan tombol "*auto place camera*" akan dinon-aktifkan dan diaktifkan kembali apabila proses telah selesai.

4.2 Perancangan Kelas

Pada bagian ini akan dijelaskan kelas-kelas yang digunakan dalam membangun perangkat lunak. Diagram kelas rinci dapat dilihat pada gambar 4.3.



Gambar 4.2: Perancangan antarmuka penempatan kamera CCTV

1 4.2.1 Kelas *Angle*

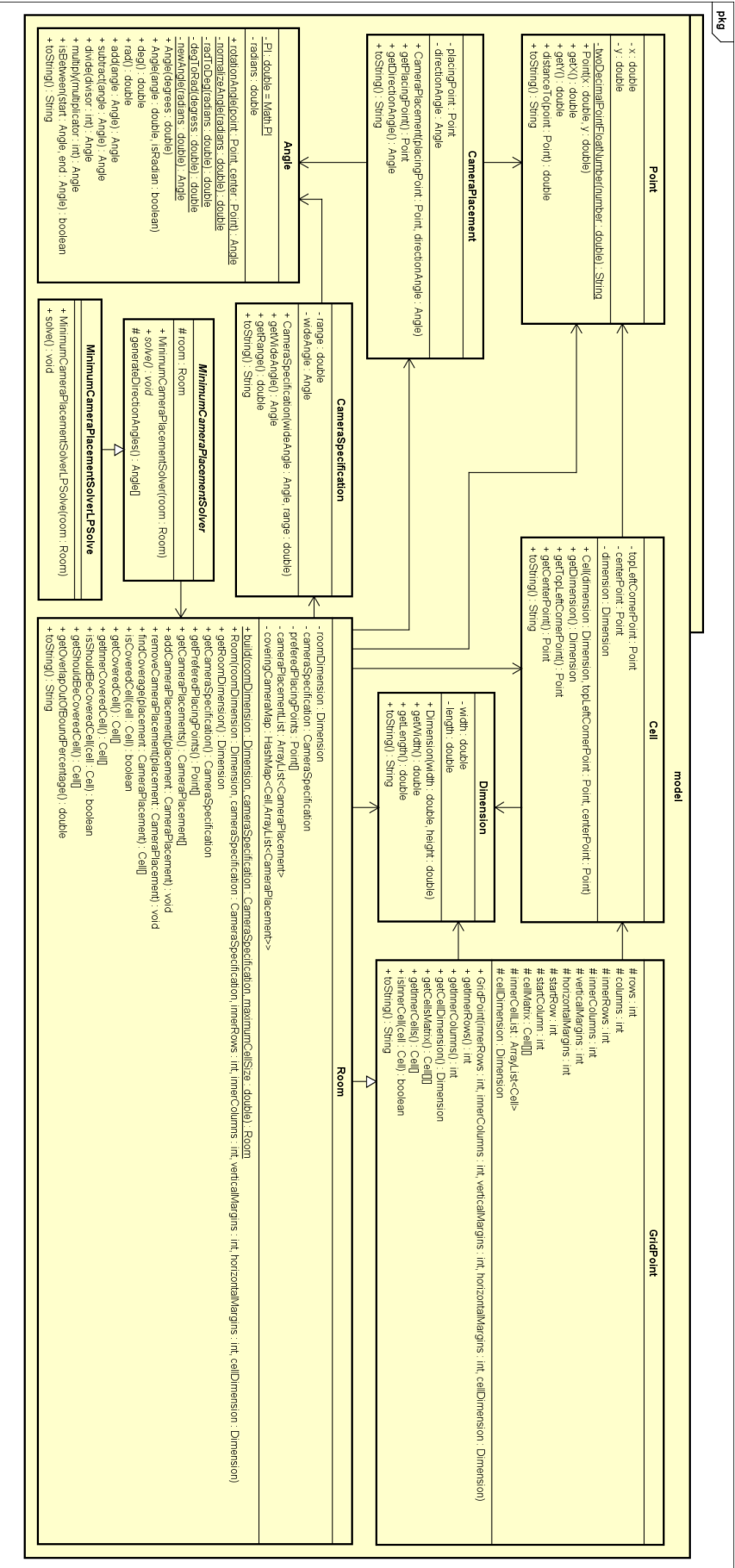
Angle
- PI : double = Math.PI
- radians : double
+ rotationAngle(point : Point, center : Point) : Angle
- normalizeAngle(radians : double) : double
- radToDeg(radians : double) : double
- degToRad(degress : double) : double
- newAngle(radians : double) : Angle
+ Angle(degrees : double)
+ Angle(angle : double, isRadian : boolean)
+ deg() : double
+ rad() : double
+ add(angle : Angle) : Angle
+ subtract(angle : Angle) : Angle
+ divide(divisor : int) : Angle
+ multiply(multiplier : int) : Angle
+ isBetween(start : Angle, end : Angle) : boolean
+ toString() : String

Gambar 4.4: Diagram kelas *Angle*

- 2 Kelas ini merepresentasikan sudut dan menangani fungsi-fungsi yang berhubungan dengan sudut.
 3 Diagram kelas *Angle* dapat dilihat pada gambar 4.4. Berikut ini merupakan atribut-atribut yang
 4 terdapat pada kelas *Angle*:

- 5 • **PI** → *double*

- 6 Atribut ini merupakan atribut statis bernilai π yang dapat digunakan oleh setiap objek dari



Gambar 4.3: Diagram kelas rinci

kelas *Angle*.

- ***radians* → *double***

Atribut ini berguna untuk menampung sudut dalam bentuk radian.

Berikut ini merupakan fungsi-fungsi yang terdapat pada kelas *Angle*:

- ***rotationAngle(point* → *Point*, *center* → *Point*) → *Angle***

Fungsi ini merupakan fungsi statis yang berguna untuk mendapatkan sudut rotasi dari titik *point* terhadap titik *center*.

- ***normalizeAngle(radians* → *double*) → *double***

Fungsi ini merupakan fungsi statis yang berguna untuk melakukan normalisasi sudut *radians* sehingga berada dalam rentang $0 \leq radians < 2\pi$.

- ***radToDeg(radians* → *double*) → *double***

Fungsi ini merupakan fungsi statis yang berguna untuk mengubah sudut dalam bentuk radian menjadi sudut dalam bentuk derajat.

- ***degToRad(degrees* → *double*) → *double***

Fungsi ini merupakan fungsi statis yang berguna untuk mengubah sudut dalam bentuk derajat menjadi sudut dalam bentuk radian.

- ***newAngle(radians* → *double*) → *Angle***

Fungsi ini merupakan fungsi statis yang berguna untuk membuat objek *Angle* baru.

- ***deg()* → *double***

Fungsi ini berguna untuk mendapatkan sudut dalam bentuk derajat.

- ***rad()* → *double***

Fungsi ini berguna untuk mendapatkan sudut dalam bentuk radian.

- ***add(angle* → *Angle*) → *Angle***

Fungsi ini berguna untuk menghasilkan objek sudut baru yang merupakan hasil penjumlahan antara sudut objek ini dengan sudut objek *angle*.

- ***subtract(angle* → *Angle*) → *Angle***

Fungsi ini berguna untuk menghasilkan objek sudut baru yang merupakan hasil pengurangan antara sudut objek ini dengan sudut objek *angle*.

- ***divide(divisor* → *int*) → *Angle***

Fungsi ini berguna untuk menghasilkan objek sudut baru yang merupakan hasil pembagian antara sudut objek ini dengan nilai *divisor*.

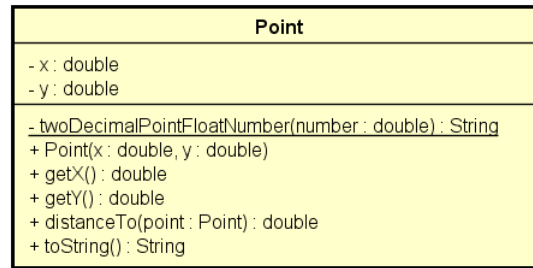
- ***multiply(multiplier* → *int*) → *Angle***

Fungsi ini berguna untuk menghasilkan objek sudut baru yang merupakan hasil pengalian antara sudut objek ini dengan nilai *multiplier*.

- ***isBetween(start* → *Angle*, *end* → *Angle*) → *boolean***

Fungsi ini berguna untuk mengetahui apakah sudut objek ini berada di antara sudut objek *start* dan sudut objek *end*.

4.2.2 Kelas *Point*



Gambar 4.5: Diagram kelas *Point*

Kelas ini merepresentasikan titik koordinat 2D. Diagram kelas *Point* dapat dilihat pada gambar 4.5.

Berikut ini merupakan atribut-atribut yang terdapat pada kelas *Point*:

- $x \rightarrow \text{double}$

Atribut ini berguna untuk menampung nilai titik pada sumbu x.

- $y \rightarrow \text{double}$

Atribut ini berguna untuk menampung nilai titik pada sumbu y.

Berikut ini merupakan fungsi-fungsi yang terdapat pada kelas *Point*:

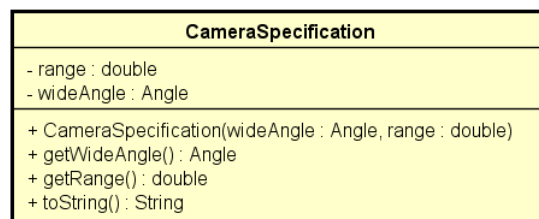
- $\text{twoDecimalPointFloatNumber}(number \rightarrow \text{double}) \rightarrow \text{String}$

Fungsi ini merupakan fungsi statis yang berguna untuk mengubah bilangan *number* ke dalam bentuk *String* dengan maksimal bilangan di belakang koma berjumlah 2 buah.

- $\text{distanceTo}(point \rightarrow \text{Point}) \rightarrow \text{double}$

Fungsi ini berguna untuk mendapatkan jarak antara titik objek ini dengan titik objek *point*.

4.2.3 Kelas *CameraSpecification*



Gambar 4.6: Diagram kelas *CameraSpecification*

Kelas ini merepresentasikan spesifikasi kamera CCTV yang terdiri dari jarak pandang efektif dan besar sudut pandang. Diagram kelas *CameraSpecification* dapat dilihat pada gambar 4.6. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *CameraSpecification*:

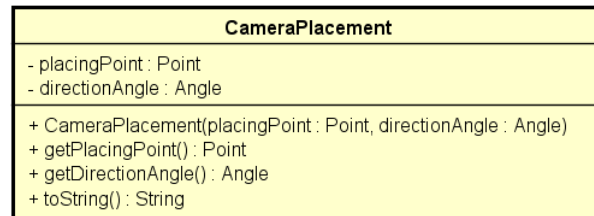
- $range \rightarrow \text{double}$

Atribut ini berguna untuk menampung jarak pandang efektif kamera CCTV.

- *wideAngle* → *Angle*

Atribut ini berguna untuk menampung besar sudut pandang kamera CCTV.

4.2.4 Kelas *CameraPlacement*



Gambar 4.7: Diagram kelas *CameraPlacement*

Kelas ini merepresentasikan penempatan kamera CCTV yang terdiri dari posisi dan sudut arah pandang. Diagram kelas *CameraPlacement* dapat dilihat pada gambar 4.7. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *CameraPlacement*:

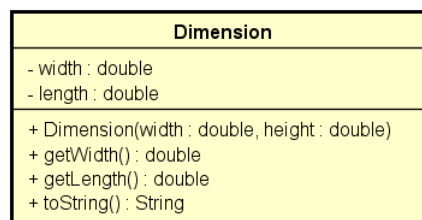
- *placingPoint* → *Point*

Atribut ini berguna untuk menampung posisi penempatan kamera CCTV.

- *directionAngle* → *Angle*

Atribut ini berguna untuk menampung sudut arah pandang kamera CCTV.

4.2.5 Kelas *Dimension*



Gambar 4.8: Diagram kelas *Dimension*

Kelas ini merepresentasikan dimensi yang terdiri dari panjang dan lebar. Diagram kelas *Dimension* dapat dilihat pada gambar 4.8. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *Dimension*:

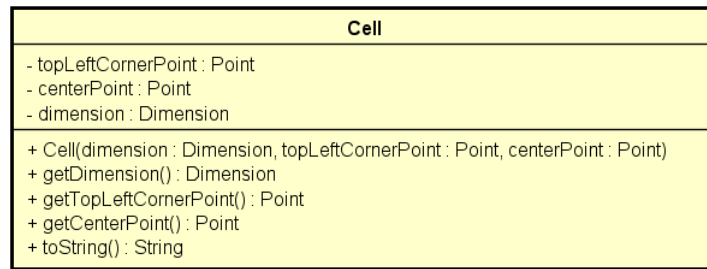
- *width* → *double*

Atribut ini berguna untuk menampung ukuran lebar.

- *length* → *double*

Atribut ini berguna untuk menampung ukuran panjang.

4.2.6 Kelas *Cell*

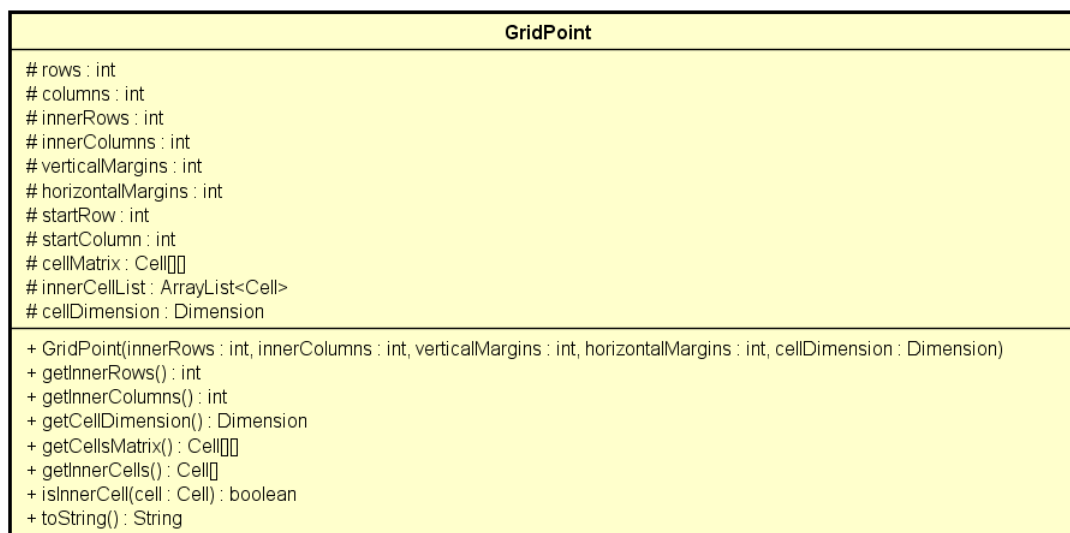


Gambar 4.9: Diagram kelas *Cell*

Kelas ini merepresentasikan cell. Diagram kelas *Cell* dapat dilihat pada gambar 4.9. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *Cell*:

- ***topLeftCornerPoint* → *Point***
Atribut ini berguna untuk menampung titik ujung kiri atas cell.
- ***centerPoint* → *Point***
Atribut ini berguna untuk menampung titik tengah cell.
- ***dimension* → *Dimension***
Atribut ini berguna untuk menampung dimensi cell.

4.2.7 Kelas *GridPoint*



Gambar 4.10: Diagram kelas *GridPoint*

Kelas ini merepresentasikan grid point yang terdiri dari matriks cell. Diagram kelas *GridPoint* dapat dilihat pada gambar 4.10. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *GridPoint*:

- ***rows*** → ***int***

Atribut ini berguna untuk menampung jumlah baris grid point secara keseluruhan.

- ***columns*** → ***int***

Atribut ini berguna untuk menampung jumlah kolom grid point secara keseluruhan.

- ***innerRows*** → ***int***

Atribut ini berguna untuk menampung jumlah baris grid point bagian dalam.

- ***innerColumns*** → ***int***

Atribut ini berguna untuk menampung jumlah kolom grid point bagian dalam.

- ***verticalMargins*** → ***int***

Atribut ini berguna untuk menampung jumlah margin vertikal grid point.

- ***horizontalMargins*** → ***int***

Atribut ini berguna untuk menampung jumlah margin horizontal grid point.

- ***startRow*** → ***int***

Atribut ini berguna untuk menampung indeks baris pertama dalam grid point bagian dalam.

- ***startColumn*** → ***int***

Atribut ini berguna untuk menampung indeks kolom pertama dalam grid point bagian dalam.

- ***cellMatrix*** → ***Cell***[[]]

Atribut ini berguna untuk menampung matriks cell 2 dimensi.

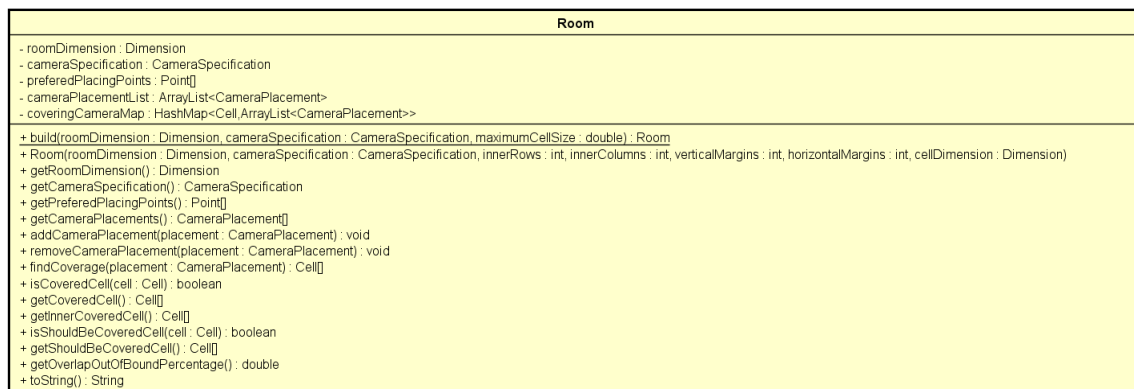
- ***innerCellList*** → ***ArrayList***<***Cell***>

Atribut ini berguna untuk menampung cell-cell yang berada pada grid point bagian dalam.

- ***cellDimension*** → ***Dimension***

Atribut ini berguna untuk menampung dimensi cell.

4.2.8 Kelas *Room*



Gambar 4.11: Diagram kelas *Room*

1 Kelas ini merepresentasikan ruangan yang dapat diisi oleh kamera-kamera CCTV. Kelas ini meru-
 2 pakan turunan dari kelas *GridPoint*. Diagram kelas *Room* dapat dilihat pada gambar 4.11. Berikut
 3 ini merupakan atribut-atribut yang terdapat pada kelas *Room*:

- 4 • ***roomDimension* → *Dimension***
 5 Atribut ini berguna untuk menampung dimensi ruangan.
- 6 • ***cameraSpecification* → *CameraSpecification***
 7 Atribut ini berguna untuk menampung spesifikasi kamera CCTV yang akan ditempatkan
 8 dalam ruangan.
- 9 • ***preferedPlacingPoints* → *Point*[]**
 10 Atribut ini berguna untuk menampung posisi-posisi yang dapat ditempati oleh kamera CCTV.
- 11 • ***cameraPlacementList* → *ArrayList*<*CameraPlacement*>**
 12 Atribut ini berguna untuk menampung daftar penempatan kamera CCTV.
- 13 • ***coveringCameraMap* → *HashMap*<*Cell*, *ArrayList*<*CameraPlacement*>>**
 14 Atribut ini berguna untuk menampung pemetaan cell dengan penempatan-penempatan kamera
 15 CCTV yang dapat mencakup cell tersebut.

16 Berikut ini merupakan fungsi-fungsi yang terdapat pada kelas *Room*:

- 17 • ***build*(*roomDimension* → *Dimension*, *cameraSpecification* → *CameraSpecification*,
 18 *maximumCellSize* → *double*) → *Room***
 19 Fungsi ini merupakan fungsi statis yang berguna untuk membuat objek *Room* yang dimana
 20 jumlah baris, jumlah kolom, jumlah margin vertikal, jumlah margin horizontal, dan ukur-
 21 an cell akan ditentukan berdasarkan ukuran ruangan *roomDimension*, spesifikasiKamera
 22 *cameraSpecification*, dan ukuran terbesar cell *maximumCellSize*.
- 23 • ***addCameraPlacement*(*placement* → *CameraPlacement*) → *void***
 24 Fungsi ini berguna untuk menambahkan penempatan kamera CCTV ke dalam daftar penem-
 25 patan kamera CCTV dan memperbaharui pemetaan cell pada atribut *coveringCameraMap*.
- 26 • ***removeCameraPlacement*(*placement* → *CameraPlacement*) → *void***
 27 Fungsi ini berguna untuk membuang penempatan kamera CCTV dari daftar penempatan
 28 kamera CCTV dan memperbaharui pemetaan cell pada atribut *coveringCameraMap*.
- 29 • ***findCoverage*(*placement* → *CameraPlacement*) → *Cell*[]**
 30 Fungsi ini berguna untuk mendapatkan cell-cell yang tercakup oleh penempatan *placement*.
- 31 • ***isCoveredCell*(*cell* → *Cell*) → *boolean***
 32 Fungsi ini berguna untuk mengetahui apakah cell *cell* telah tercakup oleh setidaknya 1
 33 penempatan kamera CCTV.
- 34 • ***getCoveredCell*() → *Cell*[]**
 35 Fungsi ini berguna untuk mendapatkan cell-cell yang telah tercakup oleh setidaknya 1 penem-
 36 patan kamera CCTV.

1 • ***getInnerCoveredCell()* → *Cell* []**

2 Fungsi ini berguna untuk mendapatkan cell-cell yang berada pada grid point bagian dalam
3 dan telah tercakup oleh setidaknya 1 penempatan kamera CCTV.

4 • ***isShouldBeCoveredCell(cell → *Cell*) → *boolean****

5 Fungsi ini berguna untuk mengetahui apakah cell *cell* berada pada grid point bagian dalam
6 dan belum tercakup oleh setidaknya 1 penempatan kamera CCTV.

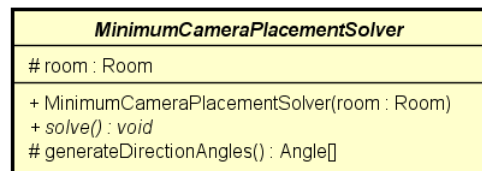
7 • ***getShouldBeCoveredCell()* → *Cell* []**

8 Fungsi ini berguna untuk mendapatkan cell-cell yang berada pada grid point bagian dalam
9 dan belum tercakup oleh setidaknya 1 penempatan kamera CCTV.

10 • ***getOverlapAndOutOfBoundPercentage()* → *double***

11 Fungsi ini berguna untuk mendapatkan persentase *overlap* dan *out of bound*.

12 **4.2.9 Kelas *MinimumCameraPlacementSolver***



Gambar 4.12: Diagram kelas *MinimumCameraPlacementSolver*

13 Kelas ini merepresentasikan pemecah masalah penempatan kamera CCTV dalam ruangan agar
14 berjumlah minimum. Kelas ini merupakan kelas abstrak. Diagram kelas *MinimumCameraPlace-*
15 *mentSolver* dapat dilihat pada gambar 4.12. Berikut ini merupakan atribut-atribut yang terdapat
16 pada kelas *MinimumCameraPlacementSolver*:

17 • ***room* → *Room***

18 Atribut ini berguna untuk menampung ruangan yang akan diselesaikan masalahnya.

19 Berikut ini merupakan fungsi-fungsi yang terdapat pada kelas *Room*:

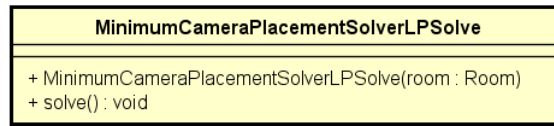
20 • ***solve()* → *void***

21 Fungsi ini merupakan fungsi abstrak yang bertujuan untuk menyelesaikan masalah penempatan
22 kamera CCTV dalam ruangan agar berjumlah minimum.

23 • ***generateDirectionAngles()* → *Angle* []**

24 Fungsi ini berguna untuk menghasilkan sudut-sudut arah pandang yang akan digunakan dalam
25 penyelesaian masalah.

1 4.2.10 Kelas *MinimumCameraPlacementSolverLPSolve*



Gambar 4.13: Diagram kelas *MinimumCameraPlacementSolverLPSolve*

2 Kelas ini merepresentasikan pemecah masalah penempatan kamera CCTV dalam ruangan agar
 3 berjumlah minimum dengan menggunakan kakas *lp_solve*. Kelas ini merupakan turunan dari
 4 kelas *MinimumCameraPlacementSolver*. Diagram kelas *MinimumCameraPlacementSolverLPSolve*
 5 dapat dilihat pada gambar 4.13. Berikut ini merupakan fungsi-fungsi yang terdapat pada kelas
 6 *MinimumCameraPlacementSolverLPSolve*:

7 • *solve()* → *void*

8 Fungsi ini berguna untuk menyelesaikan masalah penempatan kamera CCTV dalam ruangan
 9 agar berjumlah minimum dengan menggunakan kakas *lp_solve*.

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Pada bab ini, akan dibahas hasil dari implementasi dan pengujian terhadap perangkat lunak yang dibangun. Pada saat tahap implementasi, terdapat spesifikasi lingkungan yang digunakan. Spesifikasi yang sama juga digunakan pada tahap pengujian. Pada tahap pengujian, terdapat 2 jenis pengujian, yaitu pengujian fungsional dan pengujian eksperimental.

5.1 Lingkungan Implementasi Perangkat Keras

Berikut ini merupakan spesifikasi perangkat keras yang digunakan pada tahap implementasi:

- CPU: Intel® Core™ i5-7200U Processor, 3M Cache, up to 3.10 Ghz
- GPU: NVIDIA GeForce 930MX
- RAM: 8GB

5.2 Lingkungan Implementasi Perangkat Lunak

Berikut ini merupakan spesifikasi perangkat lunak yang digunakan pada tahap implementasi:

- OS: Windows 10 Pro, 64-bit
- Pemrograman: Java 8 Update 152 (64-bit)

5.3 Implementasi Antarmuka

Pada bagian ini, akan dibahas hasil implementasi antarmuka sesuai dengan perancangan antarmuka yang dilakukan pada bab sebelumnya. Berikut ini adalah hasil dari implementasi antarmuka:

- Antarmuka: **Penerima Masukan**

Gambar 5.1 menunjukkan tampilan antarmuka penerima masukan. Pada antarmuka ini terdapat kolom-kolom masukan yang dapat diisi oleh pengguna. Apabila pengguna telah selesai mengisi kolom-kolom tersebut, pengguna dapat menekan tombol "*submit*" untuk diarahkan ke antarmuka simulasi penempatan kamera CCTV.

Gambar 5.1: Antarmuka penerima masukan

• Antarmuka: Simulasi Penempatan Kamera CCTV

Gambar 5.2 menunjukkan tampilan antarmuka simulasi penempatan kamera CCTV. Pada antarmuka ini, pengguna dapat melakukan simulasi penempatan kamera CCTV. Pengguna dapat melihat penempatan-penempatan beserta dengan cakupannya melalui panel visualisasi yang berada di bagian kanan antarmuka. Pada bagian kiri terdapat panel informasi yang menunjukkan informasi dari simulasi yang sedang dijalankan. Pada panel ini, pengguna dapat melihat penempatan-penempatan yang sedang diterapkan dalam ruangan. Pada bagian kanan atas antarmuka terdapat panel penambah kamera CCTV yang digunakan untuk menambah penempatan kamera CCTV. Pada panel ini, pengguna dapat menambah penempatan dengan menentukan koordinat dan sudut arah pandang dari kamera CCTV. Pengguna juga dapat memerintahkan simulasi untuk mencari penempatan-penempatan kamera CCTV dengan menggunakan metode yang dibahas pada bagian penyelesaian masalah. Hal ini dapat dilakukan pengguna dengan menekan tombol "*auto place camera*" yang berada pada panel penambah kamera CCTV.

Gambar 5.2: Antarmuka penempatan kamera CCTV

5.4 Pengujian Fungsional

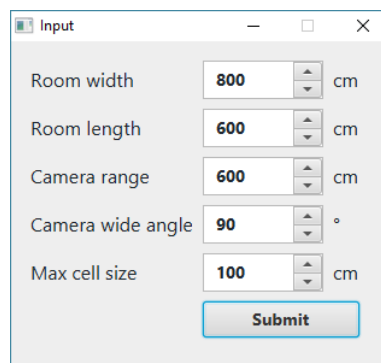
Pada bagian ini, perangkat lunak akan diuji untuk memastikan bahwa setiap fungsi-fungsi dalam perangkat lunak dapat bekerja sesuai tujuannya. Pengujian ini dilakukan sesuai dengan skenario-skenario yang terdapat pada use case. Berikut ini pengujian-pengujian fungsional yang dilakukan:

- **Pengujian: Memasukkan Spesifikasi Masalah**

Pada pengujian ini, akan diuji apakah perangkat lunak dapat membangun simulasi masalah yang sesuai dengan masukan yang diberikan oleh pengguna. Pada pengujian ini, akan digunakan masukan sebagai berikut:

- Lebar ruangan: 800 cm
- Panjang ruangan: 600 cm
- Jarak pandang kamera CCTV: 600 cm
- Besar sudut pandang kamera CCTV: 90°
- Ukuran terbesar cell: 100 cm

Masukan-masukan tersebut dimasukkan melalui antarmuka penerima masukan seperti pada gambar 5.3.

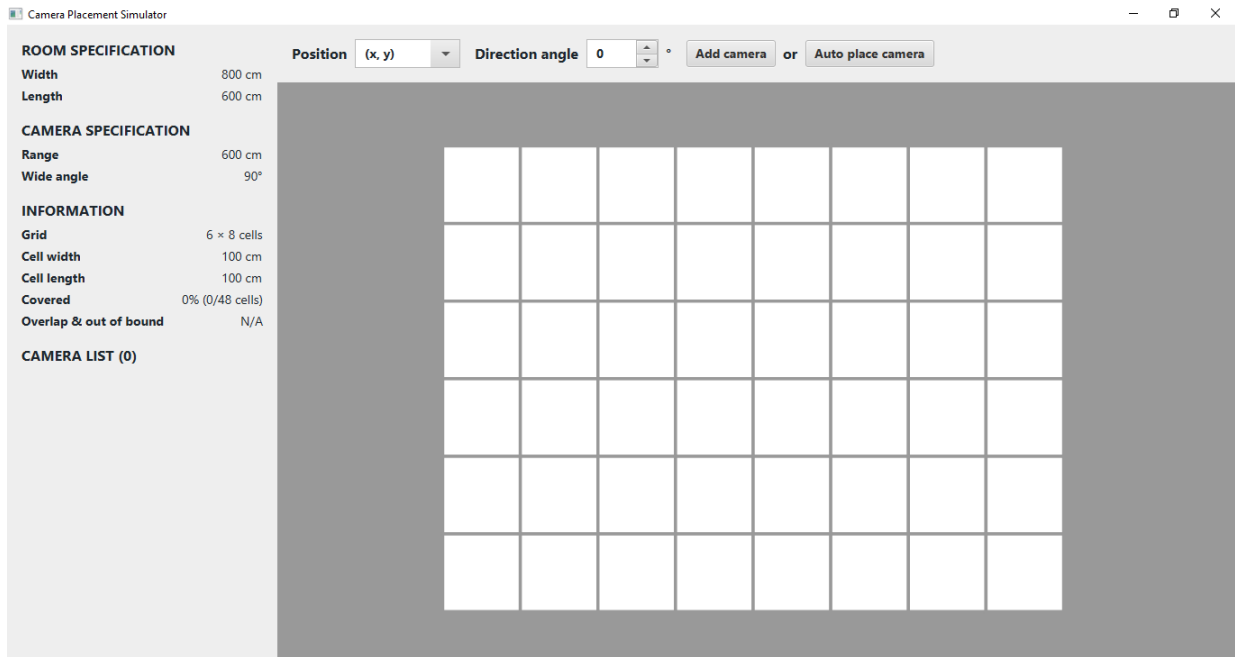


The image shows a software window titled "Input" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are five input fields, each with a label, a numeric input box, and a unit indicator. The inputs are: "Room width" with value 800 and unit "cm"; "Room length" with value 600 and unit "cm"; "Camera range" with value 600 and unit "cm"; "Camera wide angle" with value 90 and unit "°"; and "Max cell size" with value 100 and unit "cm". Each input box has small up and down arrow buttons on its right side. At the bottom of the window, there is a "Submit" button.

Field	Value	Unit
Room width	800	cm
Room length	600	cm
Camera range	600	cm
Camera wide angle	90	°
Max cell size	100	cm

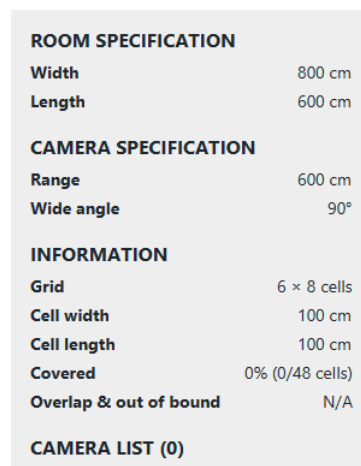
Gambar 5.3: Tampilan pengisian masukan masalah

Setelah masukan-masukan tersebut dimasukkan, tombol "submit" ditekan. Kemudian, perangkat lunak menampilkan antarmuka simulasi penempatan kamera CCTV seperti pada gambar 5.4.



Gambar 5.4: Tampilan setelah pengisian masukan masalah

- 1 Pada panel informasi, terdapat informasi masalah yang sesuai dengan masukan yang diberikan.
- 2 Panel informasi dapat dilihat pada gambar 5.5.

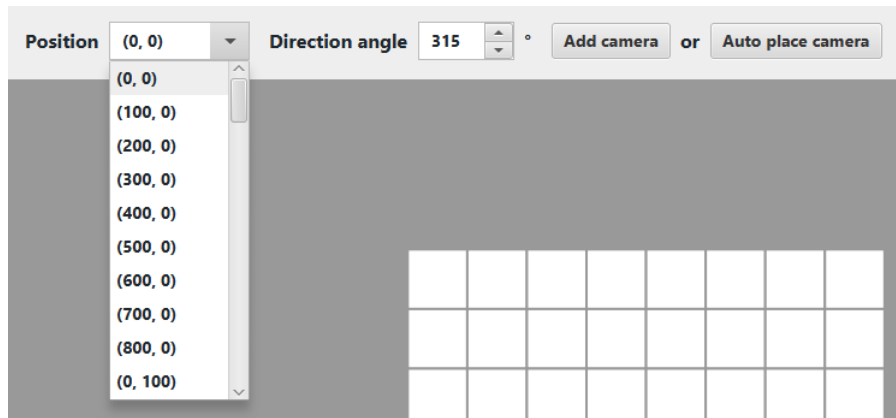


Gambar 5.5: Tampilan panel informasi setelah pengisian masukan masalah

- 3 Hal ini menunjukkan bahwa proses memasukkan spesifikasi masalah telah berjalan sesuai
- 4 dengan fungsinya. Dengan demikian, pengujian memasukkan spesifikasi masalah telah berhasil
- 5 dilakukan.

6 • Pengujian: Menambahkan Penempatan Kamera CCTV

- 7 Pada pengujian ini, akan diuji apakah perangkat lunak dapat merespon penambahan penem-
- 8 patan kamera CCTV dengan memperbaharui panel informasi dan panel visualisasi. Pengujian
- 9 dilakukan dengan memilih titik (0, 0) dan mengisi sudut 315° sebagai posisi dan sudut arah
- 10 pandang kamera CCTV. Gambar 5.6 menunjukkan tampilan pada saat mengisi penempatan
- 11 kamera CCTV.

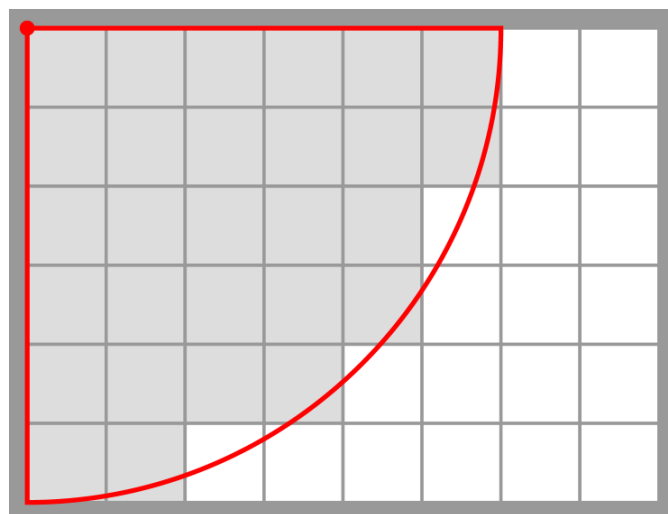


Gambar 5.6: Tampilan panel penambahan penempatan kamera CCTV

- 1 Setelah mengisi penempatan tersebut, tombol "add camera" ditekan. Perangkat lunak
- 2 merespon penambahan penempatan tersebut dengan memperbaharui panel informasi dan
- 3 panel visualisasi seperti pada gambar 5.7 dan 5.8.

ROOM SPECIFICATION	
Width	800 cm
Length	600 cm
CAMERA SPECIFICATION	
Range	600 cm
Wide angle	90°
INFORMATION	
Grid	6 × 8 cells
Cell width	100 cm
Cell length	100 cm
Covered	58.33% (28/48 cells)
Overlap & out of bound	N/A
CAMERA LIST (1)	
1. At (0, 0) facing 315°	remove

Gambar 5.7: Tampilan panel informasi setelah menambah penempatan kamera CCTV



Gambar 5.8: Tampilan panel visualisasi setelah menambah penempatan kamera CCTV

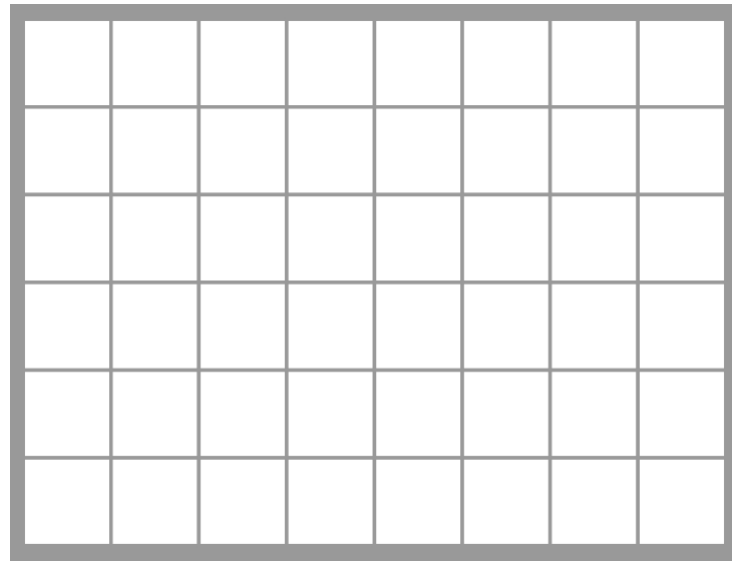
Pada panel informasi bagian daftar penempatan kamera CCTV, terdapat baris baru bertuliskan "At (0, 0) facing 315°" yang menunjukkan penempatan pada titik (0, 0) dengan sudut arah pandang 315°. Selain itu, informasi-informasi lainnya yang terdapat pada panel informasi juga telah diperbaharui. Pada panel visualisasi terdapat visualisasi objek kamera CCTV pada titik (0, 0) dengan sudut arah pandang 315°. Hal ini menunjukkan bahwa penambahan penempatan kamera CCTV telah berhasil dilakukan sehingga proses menambahkan penempatan kamera CCTV dinyatakan berjalan sesuai dengan fungsinya. Dengan demikian, pengujian menambahkan penempatan kamera CCTV telah berhasil dilakukan.

• Pengujian: Membuang Penempatan Kamera CCTV

Pada pengujian ini akan diuji apakah perangkat lunak dapat merespon pembuangan penempatan kamera CCTV dengan memperbaharui panel informasi dan panel visualisasi penempatan kamera CCTV. Pengujian dilakukan dengan memilih penempatan pada titik (0, 0) dengan sudut arah pandang (315°) sebagai penempatan yang akan dibuang. Pada penempatan tersebut, tombol "remove" ditekan. Perangkat lunak merespon pembuangan penempatan tersebut dengan memperbaharui panel informasi dan panel visualisasi seperti pada gambar 5.9 dan 5.10.

ROOM SPECIFICATION	
Width	800 cm
Length	600 cm
CAMERA SPECIFICATION	
Range	600 cm
Wide angle	90°
INFORMATION	
Grid	6 × 8 cells
Cell width	100 cm
Cell length	100 cm
Covered	0% (0/48 cells)
Overlap & out of bound	N/A
CAMERA LIST (0)	

Gambar 5.9: Tampilan panel informasi setelah membuang penempatan kamera CCTV



Gambar 5.10: Tampilan panel visualisasi setelah membuang penempatan kamera CCTV

Pada panel informasi bagian daftar penempatan, tidak lagi ditemukan penempatan pada titik (0, 0) dengan sudut arah pandang 315° . Pada panel visualisasi juga tidak lagi ditemukan visualisasi objek kamera CCTV pada titik (0, 0) dengan sudut arah pandang 315° . Hal ini menunjukkan bahwa pembuangan penempatan kamera CCTV telah berhasil dilakukan sehingga proses membuang penempatan kamera CCTV dinyatakan berjalan sesuai dengan fungsinya. Dengan demikian, pengujian membuang penempatan kamera CCTV telah berhasil dilakukan.

5.5 Pengujian Eksperimental

Pada bagian ini akan dibahas pengujian yang dilakukan dengan eksperimen. Eksperimen yang dilakukan bertujuan untuk mendapatkan informasi mengenai korelasi masukan spesifikasi masalah dengan tingkat *overlap* dan *out of bound*. Pada setiap eksperimen, akan dijalankan proses pencarian minimum kamera CCTV untuk mendapatkan hasil yang dapat dianalisa. Gambar tampilan perangkat lunak untuk setiap eksperimen dapat dilihat pada bagian lampiran.

5.5.1 Eksperimen Ukuran Cell

Pada eksperimen ini akan dianalisa tingkat *overlap* dan *out of bound* terhadap ukuran cell. Eksperimen ini dilakukan sebanyak 3 kali dengan ukuran cell yang berbeda. Berikut ini merupakan spesifikasi masalah yang digunakan dalam eksperimen ini:

1. Eksperimen pertama

- Lebar ruangan: 800 cm
- Panjang ruangan: 600 cm
- Jarak pandang kamera CCTV: 275 cm
- Besar sudut pandang kamera CCTV: 90°

- Ukuran cell: 100 cm

2. Eksperimen kedua

- Lebar ruangan: 800 cm
- Panjang ruangan: 600 cm
- Jarak pandang kamera CCTV: 275 cm
- Besar sudut pandang kamera CCTV: 90°
- Ukuran cell: 50 cm

3. Eksperimen ketiga

- Lebar ruangan: 800 cm
- Panjang ruangan: 600 cm
- Jarak pandang kamera CCTV: 275 cm
- Besar sudut pandang kamera CCTV: 90°
- Ukuran cell: 25 cm

Berdasarkan eksperimen yang dilakukan, didapatkan hasil sebagai berikut:

1. Eksperimen pertama

- Tingkat *overlap* dan *out of bound*: 0%

2. Eksperimen kedua

- Tingkat *overlap* dan *out of bound*: 14.58%

3. Eksperimen ketiga

- Tingkat *overlap* dan *out of bound*: 25%

Berdasarkan hasil tersebut, didapatkan bahwa pemodelan menggunakan cell yang berukuran lebih kecil akan menghasilkan efektivitas yang lebih baik, yakni tingkat *overlap* dan *out of bound* yang lebih kecil.

5.5.2 Eksperimen Rasio Sisi Terpendek Ruangan dengan Jarak Pandang Kamera CCTV

Pada eksperimen ini akan dianalisa tingkat *overlap* dan *out of bound* terhadap rasio antara sisi terpendek ruangan dengan jarak pandang kamera CCTV. Eksperimen ini dilakukan sebanyak 4 kali dengan jarak pandang kamera CCTV yang berbeda sehingga didapatkan rasio yang berbeda. Berikut ini merupakan spesifikasi masalah yang digunakan dalam eksperimen ini:

1. Eksperimen pertama

- Lebar ruangan: 400 cm
- Panjang ruangan: 300 cm

- Jarak pandang kamera CCTV: 300 cm
- Besar sudut pandang kamera CCTV: 90°
- Ukuran cell: 25 cm
- Rasio $\frac{\text{sisi terpendek ruangan}}{\text{jarak pandang kamera CCTV}} = 1$

2. Eksperimen kedua

- Lebar ruangan: 400 cm
- Panjang ruangan: 300 cm
- Jarak pandang kamera CCTV: 150 cm
- Besar sudut pandang kamera CCTV: 90°
- Ukuran cell: 25 cm
- Rasio $\frac{\text{sisi terpendek ruangan}}{\text{jarak pandang kamera CCTV}} = 2$

3. Eksperimen ketiga

- Lebar ruangan: 400 cm
- Panjang ruangan: 300 cm
- Jarak pandang kamera CCTV: 100 cm
- Besar sudut pandang kamera CCTV: 90°
- Ukuran cell: 25 cm
- Rasio $\frac{\text{sisi terpendek ruangan}}{\text{jarak pandang kamera CCTV}} = 3$

4. Eksperimen keempat

- Lebar ruangan: 400 cm
- Panjang ruangan: 300 cm
- Jarak pandang kamera CCTV: 75 cm
- Besar sudut pandang kamera CCTV: 90°
- Ukuran cell: 25 cm
- Rasio $\frac{\text{sisi terpendek ruangan}}{\text{jarak pandang kamera CCTV}} = 4$

Berdasarkan eksperimen yang dilakukan, didapatkan hasil sebagai berikut:

1. Eksperimen pertama

- Tingkat *overlap* dan *out of bound*: 75%

2. Eksperimen kedua

- Tingkat *overlap* dan *out of bound*: 16.67%

3. Eksperimen ketiga

- Tingkat *overlap* dan *out of bound*: 8.33%

4. Eksperimen keempat

- Tingkat *overlap* dan *out of bound*: 0%

Berdasarkan hasil tersebut, didapatkan bahwa semakin besar rasio sisi terpendek ruangan dengan jarak pandang kamera CCTV, maka tingkat *overlap* dan *out of bound* akan semakin kecil.

5.5.3 Eksperimen Besar Sudut Pandang Kamera CCTV

Pada eksperimen ini akan dianalisa tingkat *overlap* dan *out of bound* terhadap besar sudut pandang kamera CCTV. Eksperimen ini dilakukan sebanyak 4 kali dengan besar sudut pandang kamera CCTV yang berbeda. Berikut ini merupakan spesifikasi masalah yang digunakan dalam eksperimen ini:

1. Eksperimen pertama

- Lebar ruangan: 400 cm
- Panjang ruangan: 300 cm
- Jarak pandang kamera CCTV: 135 cm
- Besar sudut pandang kamera CCTV: 90°
- Ukuran cell: 25 cm

2. Eksperimen kedua

- Lebar ruangan: 400 cm
- Panjang ruangan: 300 cm
- Jarak pandang kamera CCTV: 135 cm
- Besar sudut pandang kamera CCTV: 75°
- Ukuran cell: 25 cm

3. Eksperimen ketiga

- Lebar ruangan: 400 cm
- Panjang ruangan: 300 cm
- Jarak pandang kamera CCTV: 135 cm
- Besar sudut pandang kamera CCTV: 60°
- Ukuran cell: 25 cm

4. Eksperimen keempat

- Lebar ruangan: 400 cm
- Panjang ruangan: 300 cm
- Jarak pandang kamera CCTV: 135 cm
- Besar sudut pandang kamera CCTV: 45°

- Ukuran cell: 25 cm

Berdasarkan eksperimen yang dilakukan, didapatkan hasil sebagai berikut:

1. Eksperimen pertama

- Tingkat *overlap* dan *out of bound*: 14.58%

2. Eksperimen kedua

- Tingkat *overlap* dan *out of bound*: 18.75%

3. Eksperimen ketiga

- Tingkat *overlap* dan *out of bound*: 16.15%

4. Eksperimen keempat

- Tingkat *overlap* dan *out of bound*: 8.33%

Berdasarkan hasil tersebut, tidak ditemukan adanya korelasi besar sudut pandang kamera CCTV terhadap tingkat *overlap* dan *out of bound*.

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan penelitian yang telah dilakukan di dalam skripsi ini, disimpulkan bahwa:

1. Masalah pencarian penempatan kamera CCTV berjumlah minimum yang mencakup seluruh ruangan dapat diselesaikan menggunakan metode linear programming. Masalah ini dapat memiliki banyak solusi, namun solusi-solusi tersebut belum tentu menghasilkan solusi kamera CCTV yang berjumlah minimum. Dengan memodelkan masalah ini dalam bentuk masalah linear programming, maka solusi optimal dari masalah ini dapat ditemukan.
2. Perangkat lunak untuk menyelesaikan masalah ini telah berhasil dibangun. Perangkat lunak ini dapat menerima spesifikasi masalah dan menerapkan metode linear programming untuk menyelesaikannya. Hasil dari penyelesaian ini disajikan dalam bentuk visualisasi sehingga mudah untuk dipahami.
3. Pengujian yang dilakukan menghasilkan informasi bahwa ukuran cell dan rasio antara sisi terpendek ruangan dengan jarak pandang kamera CCTV memiliki korelasi dengan tingkat *overlap* dan *out of bound*. Apabila ukuran cell semakin kecil, maka tingkat *overlap* dan *out of bound* akan semakin besar. Apabila rasio antara sisi terpendek ruangan dengan jarak pandang kamera CCTV semakin besar, maka tingkat *overlap* dan *out of bound* akan semakin kecil. Sedangkan untuk besar sudut pandang kamera CCTV, tidak ditemukan adanya korelasi dengan tingkat *overlap* dan *out of bound*.

Dengan demikian, setiap tujuan dalam skripsi ini telah tercapai.

6.2 Saran

Berikut ini adalah saran-saran dari penulis bagi pembaca/peneliti yang ingin melanjutkan penelitian ini:

1. Lanjutan pemodelan masalah dalam bidang 3 dimensi.
2. Penerapan algoritma *Heuristic* pada metode linear programming agar masalah linear programming dapat diselesaikan dengan lebih cepat.

DAFTAR REFERENSI

- [1] Winston, W. L. dan Goldberg, J. B. (2004) *Operations research: applications and algorithms*. Thomson/Brooks/Cole Belmont^ eCalif Calif.

LAMPIRAN A

KODE PROGRAM

Listing A.1: Angle.java

```
1 package model;
2
3 /**
4  *
5  * @author Prayogo Cendra
6  */
7 public class Angle {
8
9     private static final double PI = Math.PI;
10    private final double radians;
11
12    //coordinate system (flipped y)
13    //  -y
14    //   ^
15    //   |
16    //-x <- + -> +x
17    //   |
18    //   v
19    //   +y
20    public static Angle rotationAngle(Point point, Point center) {
21        // double radians = Math.atan2( //if normal y
22        //     point.getY() - center.getY(),
23        //     point.getX() - center.getX()
24        // );
25        double radians = Math.atan2( //if flipped y
26            center.getY() - point.getY(),
27            point.getX() - center.getX()
28        );
29        Angle angle = new Angle(radians, true);
30        return angle;
31    }
32
33    private static double normalizeAngle(double radians) {
34        radians %= 2 * PI;
35        if (radians < 0) {
36            radians += 2 * PI;
37        }
38        return radians;
39    }
40
41    private static double radToDeg(double radians) {
42        int decimalPrecision = 3;
43        return (double) Math.round(
44            radians * 180 / PI * Math.pow(10, decimalPrecision)
45        ) / Math.pow(10, decimalPrecision);
46    }
47
48    private static double degToRad(double degress) {
49        return degress / 180 * PI;
50    }
51
52    private static Angle newAngle(double radians) {
53        return new Angle(radians, true);
54    }
55
56    public Angle(double degrees) {
57        this(degrees, false);
58    }
59
60    public Angle(double angle, boolean isRadian) {
61        if (!isRadian) {
62            angle = degToRad(angle);
63        }
64        this.radians = normalizeAngle(angle);
65    }
66
67    public double deg() {
68        return radToDeg(radians);
69    }
70
71    public double rad() {
72        return radians;
73    }
74
75    public Angle add(Angle angle) {
```

```

76         return newAngle(radians + angle.radians);
77     }
78
79     public Angle subtract(Angle angle) {
80         return newAngle(radians - angle.radians);
81     }
82
83     public Angle divide(int divisor) {
84         return newAngle(radians / divisor);
85     }
86
87     public Angle multiply(int multiplicator) {
88         return newAngle(radians * multiplicator);
89     }
90
91     public boolean isBetween(Angle start, Angle end) {
92         if (start.rad() < end.rad()) {
93             return start.rad() < radians && radians < end.rad();
94         } else {
95             return start.rad() < radians || radians < end.rad();
96         }
97     }
98
99     @Override
100     public int hashCode() {
101         int hash = 3;
102         hash = 53 * hash + (int) (Double.doubleToLongBits(this.radians) ^ (Double.doubleToLongBits(this.radians) >>> 32));
103         return hash;
104     }
105
106     @Override
107     public boolean equals(Object obj) {
108         if (this == obj) {
109             return true;
110         }
111         if (obj == null) {
112             return false;
113         }
114         if (getClass() != obj.getClass()) {
115             return false;
116         }
117         final Angle other = (Angle) obj;
118         if (Math.abs(this.radians - other.radians) > 0.000001) {
119             return false;
120         }
121         return true;
122     }
123
124     @Override
125     public String toString() {
126         StringBuilder sb = new StringBuilder();
127         sb.append(deg());
128         sb.append("°");
129         return sb.toString();
130     }
131 }

```

Listing A.2: Point.java

```

1 package model;
2
3 import java.text.DecimalFormat;
4
5 /**
6  *
7  * @author Prayogo Cendra
8  */
9 public class Point {
10
11     private final double x, y;
12
13     private static String twoDecimalPointFloatNumber(double number) {
14         DecimalFormat df = new DecimalFormat("#.##");
15         return df.format(number);
16     }
17
18     public Point(double x, double y) {
19         this.x = x;
20         this.y = y;
21     }
22
23     public double getX() {
24         return x;
25     }
26
27     public double getY() {
28         return y;
29     }
30
31     public double distanceTo(Point point) {
32         double horizontalDifference = point.getX() - x;
33         double verticalDifference = point.getY() - y;
34         return Math.sqrt(
35             Math.pow(horizontalDifference, 2)
36             + Math.pow(verticalDifference, 2)
37         );
38     }
39 }

```



```

40 | @Override
41 | public int hashCode() {
42 |     int hash = 7;
43 |     hash = 47 * hash + (int) (Double.doubleToLongBits(this.x) ^ (Double.doubleToLongBits(this.x) >>> 32));
44 |     hash = 47 * hash + (int) (Double.doubleToLongBits(this.y) ^ (Double.doubleToLongBits(this.y) >>> 32));
45 |     return hash;
46 | }
47 |
48 | @Override
49 | public boolean equals(Object obj) {
50 |     if (this == obj) {
51 |         return true;
52 |     }
53 |     if (obj == null) {
54 |         return false;
55 |     }
56 |     if (getClass() != obj.getClass()) {
57 |         return false;
58 |     }
59 |     final Point other = (Point) obj;
60 |     if (Math.abs(this.x - other.x) > 0.000001) {
61 |         return false;
62 |     }
63 |     if (Math.abs(this.y - other.y) > 0.000001) {
64 |         return false;
65 |     }
66 |     return true;
67 | }
68 |
69 | @Override
70 | public String toString() {
71 |     return "("
72 |         + twoDecimalPointFloatNumber(x)
73 |         + ", "
74 |         + twoDecimalPointFloatNumber(y)
75 |         + ")";
76 | }
77 | }

```

Listing A.3: CameraSpecification.java

```

1 | package model;
2 |
3 | /**
4 |  *
5 |  * @author Prayogo Cendra
6 |  */
7 | public class CameraSpecification {
8 |
9 |     private final Angle wideAngle;
10 |    private final double range;
11 |
12 |    public CameraSpecification(Angle wideAngle, double range) {
13 |        this.wideAngle = wideAngle;
14 |        this.range = range;
15 |    }
16 |
17 |    public Angle getWideAngle() {
18 |        return wideAngle;
19 |    }
20 |
21 |    public double getRange() {
22 |        return range;
23 |    }
24 |
25 |    @Override
26 |    public String toString() {
27 |        return "("
28 |            + range
29 |            + ", "
30 |            + wideAngle
31 |            + ")";
32 |    }
33 | }

```

Listing A.4: CameraPlacement.java

```

1 | package model;
2 |
3 | import java.util.Objects;
4 |
5 | /**
6 |  *
7 |  * @author Prayogo Cendra
8 |  */
9 | public class CameraPlacement {
10 |
11 |     private final Point placingPoint;
12 |     private final Angle directionAngle;
13 |
14 |     public CameraPlacement(Point placingPoint, Angle directionAngle) {
15 |         this.placingPoint = placingPoint;
16 |         this.directionAngle = directionAngle;
17 |     }
18 |
19 |     public Point getPlacingPoint() {

```

```

20     return placingPoint;
21 }
22
23 public Angle getDirectionAngle() {
24     return directionAngle;
25 }
26
27 @Override
28 public int hashCode() {
29     int hash = 3;
30     hash = 11 * hash + Objects.hashCode(this.placingPoint);
31     hash = 11 * hash + Objects.hashCode(this.directionAngle);
32     return hash;
33 }
34
35 @Override
36 public boolean equals(Object obj) {
37     if (this == obj) {
38         return true;
39     }
40     if (obj == null) {
41         return false;
42     }
43     if (getClass() != obj.getClass()) {
44         return false;
45     }
46     final CameraPlacement other = (CameraPlacement) obj;
47     if (!Objects.equals(this.placingPoint, other.placingPoint)) {
48         return false;
49     }
50     if (!Objects.equals(this.directionAngle, other.directionAngle)) {
51         return false;
52     }
53     return true;
54 }
55
56 @Override
57 public String toString() {
58     return "("
59         + placingPoint
60         + ", "
61         + directionAngle
62         + ")";
63 }
64 }

```

Listing A.5: Dimension.java

```

1 package model;
2
3 /**
4  *
5  * @author Prayogo Cendra
6  */
7 public class Dimension {
8
9     private final double width, length;
10
11     public Dimension(double width, double length) {
12         this.width = width;
13         this.length = length;
14     }
15
16     public double getWidth() {
17         return width;
18     }
19
20     public double getLength() {
21         return length;
22     }
23
24     @Override
25     public int hashCode() {
26         int hash = 5;
27         hash = 29 * hash + (int) (Double.doubleToLongBits(this.width) ^ (Double.doubleToLongBits(this.width) >>> 32));
28         hash = 29 * hash + (int) (Double.doubleToLongBits(this.length) ^ (Double.doubleToLongBits(this.length) >>> 32));
29         return hash;
30     }
31
32     @Override
33     public boolean equals(Object obj) {
34         if (this == obj) {
35             return true;
36         }
37         if (obj == null) {
38             return false;
39         }
40         if (getClass() != obj.getClass()) {
41             return false;
42         }
43         final Dimension other = (Dimension) obj;
44         if (Math.abs(this.width - other.width) > 0.000001) {
45             return false;
46         }
47         if (Math.abs(this.length - other.length) > 0.000001) {
48             return false;
49         }
50         return true;

```

```

51 |     }
52 |
53 |     @Override
54 |     public String toString() {
55 |         return width + "x" + length;
56 |     }
57 | }

```

Listing A.6: Cell.java

```

1 | package model;
2 |
3 | import java.util.Objects;
4 |
5 | /**
6 |  *
7 |  * @author Prayogo Cendra
8 |  */
9 | public class Cell {
10 |
11 |     private final Dimension dimension;
12 |     private final Point topLeftCornerPoint, centerPoint;
13 |
14 |     public Cell(
15 |         Dimension dimension,
16 |         Point topLeftCornerPoint,
17 |         Point centerPoint
18 |     ) {
19 |         this.dimension = dimension;
20 |         this.topLeftCornerPoint = topLeftCornerPoint;
21 |         this.centerPoint = centerPoint;
22 |     }
23 |
24 |     public Dimension getDimension() {
25 |         return dimension;
26 |     }
27 |
28 |     public Point getTopLeftCornerPoint() {
29 |         return topLeftCornerPoint;
30 |     }
31 |
32 |     public Point getCenterPoint() {
33 |         return centerPoint;
34 |     }
35 |
36 |     @Override
37 |     public int hashCode() {
38 |         int hash = 7;
39 |         hash = 47 * hash + Objects.hashCode(this.dimension);
40 |         hash = 47 * hash + Objects.hashCode(this.topLeftCornerPoint);
41 |         hash = 47 * hash + Objects.hashCode(this.centerPoint);
42 |         return hash;
43 |     }
44 |
45 |     @Override
46 |     public boolean equals(Object obj) {
47 |         if (this == obj) {
48 |             return true;
49 |         }
50 |         if (obj == null) {
51 |             return false;
52 |         }
53 |         if (getClass() != obj.getClass()) {
54 |             return false;
55 |         }
56 |         final Cell other = (Cell) obj;
57 |         if (!Objects.equals(this.dimension, other.dimension)) {
58 |             return false;
59 |         }
60 |         if (!Objects.equals(this.topLeftCornerPoint, other.topLeftCornerPoint)) {
61 |             return false;
62 |         }
63 |         if (!Objects.equals(this.centerPoint, other.centerPoint)) {
64 |             return false;
65 |         }
66 |         return true;
67 |     }
68 |
69 |     @Override
70 |     public String toString() {
71 |         return "Cell_" + centerPoint + ")";
72 |     }
73 | }

```

Listing A.7: GridPoint.java

```

1 | package model;
2 |
3 | import java.util.ArrayList;
4 |
5 | /**
6 |  *
7 |  * @author Prayogo Cendra
8 |  */
9 | public class GridPoint {
10 |

```

```

11     protected final int rows,
12         columns,
13         innerRows,
14         innerColumns,
15         verticalMargins,
16         horizontalMargins,
17         startRow,
18         startColumn;
19     protected final Dimension cellDimension;
20     protected final Cell[][] cellsMatrix;
21     protected final ArrayList<Cell> innerCellList;
22
23     public GridPoint(
24         int innerRows,
25         int innerColumns,
26         int verticalMargins,
27         int horizontalMargins,
28         Dimension cellDimension
29     ) {
30         this.innerRows = innerRows;
31         this.innerColumns = innerColumns;
32         this.verticalMargins = verticalMargins;
33         this.horizontalMargins = horizontalMargins;
34         this.rows = innerRows + 2 * verticalMargins;
35         this.columns = innerColumns + 2 * horizontalMargins;
36         this.startRow = verticalMargins;
37         this.startColumn = horizontalMargins;
38         this.cellDimension = cellDimension;
39         this.cellsMatrix = new Cell[rows][columns];
40         this.innerCellList = new ArrayList();
41         for (int i = 0; i < rows; i++) {
42             for (int j = 0; j < columns; j++) {
43                 double cellWidth = cellDimension.getWidth();
44                 double cellLength = cellDimension.getLength();
45
46                 Point topLeftCornerPoint = new Point(
47                     (j - startColumn) * cellWidth,
48                     (i - startRow) * cellLength
49                 );
50                 Point centerPoint = new Point(
51                     (j - startColumn + 0.5) * cellWidth,
52                     (i - startRow + 0.5) * cellLength
53                 );
54                 boolean innerCell
55                     = i >= startRow
56                     && i < startRow + innerRows
57                     && j >= startColumn
58                     && j < startColumn + innerColumns;
59
60                 cellsMatrix[i][j] = new Cell(
61                     cellDimension,
62                     topLeftCornerPoint,
63                     centerPoint
64                 );
65
66                 if (innerCell) {
67                     innerCellList.add(cellsMatrix[i][j]);
68                 }
69             }
70         }
71     }
72
73     public int getInnerRows() {
74         return innerRows;
75     }
76
77     public int getInnerColumns() {
78         return innerColumns;
79     }
80
81     public Dimension getCellDimension() {
82         return cellDimension;
83     }
84
85     public Cell[][] getCellsMatrix() {
86         return cellsMatrix;
87     }
88
89     public Cell[] getInnerCells() {
90         Cell[] innerCells = new Cell[innerCellList.size()];
91         innerCellList.toArray(innerCells);
92         return innerCells;
93     }
94
95     public boolean isInnerCell(Cell cell) {
96         return innerCellList.contains(cell);
97     }
98
99     @Override
100     public String toString() {
101         StringBuilder sb = new StringBuilder();
102         sb.append("GRID_POINT\nTotal_cells:");
103         sb.append(rows);
104         sb.append("x");
105         sb.append(columns);
106         sb.append("\nInner_cells:");
107         sb.append(innerRows);
108         sb.append("x");
109         sb.append(innerColumns);

```

```

110 |         sb.append("\nCell_dimension:");
111 |         sb.append(cellDimension);
112 |         sb.append("\n\nCELLS\n");
113 |         for (int i = 0; i < rows; i++) {
114 |             for (int j = 0; j < columns; j++) {
115 |                 sb.append(cellsMatrix[i][j]);
116 |                 sb.append("\n");
117 |             }
118 |         }
119 |         return sb.toString();
120 |     }
121 | }

```

Listing A.8: Room.java

```

1 | package model;
2 |
3 | import java.util.ArrayList;
4 | import java.util.HashMap;
5 | import java.util.Iterator;
6 |
7 | /**
8 |  *
9 |  * @author Prayogo Cendra
10 |  */
11 | public class Room extends GridPoint {
12 |
13 |     private final Dimension roomDimension;
14 |     private final CameraSpecification cameraSpecification;
15 |     private final Point[] preferredPlacingPoints;
16 |     private final ArrayList<CameraPlacement> cameraPlacementList;
17 |     private final HashMap<Cell, ArrayList<CameraPlacement>> coveringCameraMap;
18 |
19 |     public static Room build(
20 |         Dimension roomDimension,
21 |         CameraSpecification cameraSpecification,
22 |         double maximumCellSize
23 |     ) {
24 |         double roomWidth = roomDimension.getWidth();
25 |         double roomLength = roomDimension.getLength();
26 |         double minimumMargin = cameraSpecification.getRange();
27 |
28 |         int innerRows = (int) Math.ceil(roomLength / maximumCellSize);
29 |         int innerColumns = (int) Math.ceil(roomWidth / maximumCellSize);
30 |         double cellWidth = roomWidth / innerColumns;
31 |         double cellLength = roomLength / innerRows;
32 |         int verticalMargins = (int) Math.ceil(minimumMargin / cellLength);
33 |         int horizontalMargins = (int) Math.ceil(minimumMargin / cellWidth);
34 |         Dimension cellDimension = new Dimension(cellWidth, cellLength);
35 |
36 |         return new Room(
37 |             roomDimension,
38 |             cameraSpecification,
39 |             innerRows,
40 |             innerColumns,
41 |             verticalMargins,
42 |             horizontalMargins,
43 |             cellDimension
44 |         );
45 |     }
46 |
47 |     public Room(
48 |         Dimension roomDimension,
49 |         CameraSpecification cameraSpecification,
50 |         int innerRows,
51 |         int innerColumns,
52 |         int verticalMargins,
53 |         int horizontalMargins,
54 |         Dimension cellDimension
55 |     ) {
56 |         super(
57 |             innerRows,
58 |             innerColumns,
59 |             verticalMargins,
60 |             horizontalMargins,
61 |             cellDimension
62 |         );
63 |
64 |         this.roomDimension = roomDimension;
65 |         this.cameraSpecification = cameraSpecification;
66 |         this.cameraPlacementList = new ArrayList<>();
67 |
68 |         double cellWidth = cellDimension.getWidth();
69 |         double cellLength = cellDimension.getLength();
70 |
71 |         this.preferredPlacingPoints
72 |             = new Point[(innerRows + 1) * (innerColumns + 1)];
73 |         for (int i = 0; i < innerRows + 1; i++) {
74 |             for (int j = 0; j < innerColumns + 1; j++) {
75 |                 preferredPlacingPoints[i * (innerColumns + 1) + j]
76 |                     = new Point(
77 |                         j * cellWidth,
78 |                         i * cellLength
79 |                     );
80 |             }
81 |         }
82 |
83 |         this.coveringCameraMap = new HashMap<>();

```

```

84     for (Cell[] cells : this.cellsMatrix) {
85         for (Cell cell : cells) {
86             this.coveringCameraMap.put(cell, new ArrayList<>());
87         }
88     }
89 }
90
91 public Dimension getRoomDimension() {
92     return roomDimension;
93 }
94
95 public CameraSpecification getCameraSpecification() {
96     return cameraSpecification;
97 }
98
99 public Point[] getPreferedPlacingPoints() {
100     return preferedPlacingPoints;
101 }
102
103 public CameraPlacement[] getCameraPlacements() {
104     CameraPlacement[] cameraPlacements
105         = new CameraPlacement[cameraPlacementList.size()];
106     cameraPlacementList.toArray(cameraPlacements);
107     return cameraPlacements;
108 }
109
110 public void addCameraPlacement(CameraPlacement placement) {
111     cameraPlacementList.add(placement);
112     for (Cell coveredCell : findCoverage(placement)) {
113         coveringCameraMap.get(coveredCell).add(placement);
114     }
115 }
116
117 public void removeCameraPlacement(CameraPlacement placement) {
118     cameraPlacementList.remove(placement);
119     for (Cell coveredCell : findCoverage(placement)) {
120         coveringCameraMap.get(coveredCell).remove(placement);
121     }
122 }
123
124 public Cell[] findCoverage(CameraPlacement placement) {
125     ArrayList<Cell> coverageCellList = new ArrayList<>();
126
127     Angle directionAngle = placement.getDirectionAngle();
128     Angle halfAngle = cameraSpecification.getWideAngle().divide(2);
129     Angle startAngle = directionAngle.subtract(halfAngle);
130     Angle endAngle = directionAngle.add(halfAngle);
131
132     for (int i = 0; i < rows; i++) {
133         for (int j = 0; j < columns; j++) {
134             Cell cell = cellsMatrix[i][j];
135             Point centerPoint = cell.getCenterPoint();
136             Angle rotationAngle = Angle.rotationAngle(centerPoint,
137                 placement.getPlacingPoint());
138
139             boolean isInsideRange
140                 = placement.getPlacingPoint()
141                     .distanceTo(centerPoint)
142                     < cameraSpecification.getRange();
143             boolean isInsideViewAngle = rotationAngle
144                 .isBetween(startAngle, endAngle);
145
146             if (isInsideRange && isInsideViewAngle) {
147                 coverageCellList.add(cell);
148             }
149         }
150     }
151
152     Cell[] coverageCells = new Cell[coverageCellList.size()];
153     coverageCellList.toArray(coverageCells);
154     return coverageCells;
155 }
156
157 public boolean isCoveredCell(Cell cell) {
158     return !coveringCameraMap.get(cell).isEmpty();
159 }
160
161 public Cell[] getCoveredCell() {
162     ArrayList<Cell> coveredCellList = new ArrayList<>();
163     for (Cell[] cells : cellsMatrix) {
164         for (Cell cell : cells) {
165             if (isCoveredCell(cell)) {
166                 coveredCellList.add(cell);
167             }
168         }
169     }
170     Cell[] coveredCells = new Cell[coveredCellList.size()];
171     coveredCellList.toArray(coveredCells);
172     return coveredCells;
173 }
174
175 public Cell[] getInnerCoveredCell() {
176     ArrayList<Cell> innerCoveredCellList = new ArrayList<>();
177     for (Cell[] cells : cellsMatrix) {
178         for (Cell cell : cells) {
179             if (isInnerCell(cell) && isCoveredCell(cell)) {
180                 innerCoveredCellList.add(cell);
181             }
182         }
183     }

```

```

183     }
184     Cell[] coveredCells = new Cell[innerCoveredCellList.size()];
185     innerCoveredCellList.toArray(coveredCells);
186     return coveredCells;
187 }
188
189 public boolean isShouldBeCoveredCell(Cell cell) {
190     return isInnerCell(cell) && !isCoveredCell(cell);
191 }
192
193 public Cell[] getShouldBeCoveredCell() {
194     ArrayList<Cell> shouldBeCoveredCellList = new ArrayList<>();
195     for (Cell[] cells : cellsMatrix) {
196         for (Cell cell : cells) {
197             if (isShouldBeCoveredCell(cell)) {
198                 shouldBeCoveredCellList.add(cell);
199             }
200         }
201     }
202     Cell[] shouldBeCoveredCells = new Cell[shouldBeCoveredCellList.size()];
203     shouldBeCoveredCellList.toArray(shouldBeCoveredCells);
204     return shouldBeCoveredCells;
205 }
206
207 public boolean isOverlapCell(Cell cell) {
208     return coveringCameraMap.get(cell).size() > 1;
209 }
210
211 public double getOverlapOutOfBoundPercentage() {
212     for (Cell innerCell : innerCellList) {
213         if (!isCoveredCell(innerCell)) {
214             return Double.NaN;
215         }
216     }
217     int totalCoverageCell = 0;
218     for (Cell[] cells : cellsMatrix) {
219         for (Cell cell : cells) {
220             totalCoverageCell += coveringCameraMap.get(cell).size();
221         }
222     }
223     int totalInnerCell = innerCellList.size();
224     return (1.0 * totalCoverageCell / totalInnerCell) - 1.0;
225 }
226
227 @Override
228 public String toString() {
229     StringBuilder sb = new StringBuilder();
230     sb.append(super.toString());
231     sb.append("\n\nCAMERA_LIST\n");
232     Iterator cameraListIterator = cameraPlacementList.iterator();
233     while (cameraListIterator.hasNext()) {
234         sb.append(cameraListIterator.next());
235         if (cameraListIterator.hasNext()) {
236             sb.append("\n\n");
237         }
238     }
239     return sb.toString();
240 }
241 }

```

Listing A.9: MinimumCameraPlacementSolver.java

```

1 package model;
2
3 /**
4  *
5  * @author Prayogo Cendra
6  */
7 public abstract class MinimumCameraPlacementSolver {
8
9     protected final Room room;
10
11     public MinimumCameraPlacementSolver(Room room) {
12         this.room = room;
13     }
14
15     public abstract void solve();
16
17     protected Angle[] generateDirectionAngles() {
18         // type unique coverage
19         // int testAngleCount = 360; // every 1 deg
20         // Point testPlacingPoint = room.getPreferedPlacingPoints()[0];
21         // Angle[] testAngles = new Angle[testAngleCount];
22         // Cell[][] testCoveredCellss = new Cell[testAngleCount][];
23         // for (int i = 0; i < testAngleCount; i++) {
24         //     testAngles[i] = new Angle(i * 360 / testAngleCount);
25         //     testCoveredCellss[i] = room.findCoverage(
26         //         new CameraPlacement(testPlacingPoint, testAngles[i])
27         //     );
28         // }
29         // boolean isFirstFound = false;
30         // int startIdx = 1;
31         // int endIdx = 0;
32         // int currIdx = 0;
33         // Cell[] startCells = testCoveredCellss[0];
34         // ArrayList<Integer> idxList = new ArrayList<>();
35         // while (true) {
36         //     boolean check = true;

```

```

37 //      Cell[] currCells = testCoveredCellss[currIdx];
38 //      if (startCells.length != currCells.length) {
39 //          check = false;
40 //      } else if (!new HashSet<>(Arrays.asList(startCells))
41 //          .equals(new HashSet<>(Arrays.asList(currCells)))) {
42 //          check = false;
43 //      }
44 //      if (check) {
45 //          endIdx = currIdx;
46 //      } else {
47 //          if (isFirstFound) {
48 //              int midIdx = (startIdx + endIdx) / 2;
49 //              if (!idxList.contains(midIdx)) {
50 //                  System.out.println(startIdx + " " + midIdx + " " + endIdx);
51 //                  idxList.add(midIdx);
52 //              } else {
53 //                  break;
54 //              }
55 //          } else {
56 //              isFirstFound = true;
57 //          }
58 //          startIdx = currIdx;
59 //          endIdx = currIdx;
60 //          startCells = testCoveredCellss[startIdx];
61 //      }
62 //      currIdx = (currIdx + 1) % testAngleCount;
63 //  }
64 //  int anglesCount = idxList.size();
65 //  Angle[] directionAngles = new Angle[anglesCount];
66 //  for (int i = 0; i < idxList.size(); i++) {
67 //      directionAngles[i] = testAngles[idxList.get(i)];
68 //      System.out.println(directionAngles[i]);
69 //  }
70 //
71 //  type-1
72 //  double cameraWideAngle = specification.getWideAngle().deg();
73 //  int angleVariations = (int) Math.ceil(360.0 / cameraWideAngle);
74 //  double angleRotation = 360.0 / angleVariations;
75 //  double angleOffset = specification.getWideAngle().deg() / 2;
76 //  Angle[] directionAngles = new Angle[angleVariations];
77 //
78 //  for (int i = 0; i < angleVariations; i++) {
79 //      directionAngles[i] = new Angle(i * angleRotation + angleOffset);
80 //  }
81 //  type-2
82 //  int anglesCount = 72;
83 //  Angle[] directionAngles = new Angle[anglesCount];
84 //  for (int i = 0; i < anglesCount; i++) {
85 //      directionAngles[i] = new Angle(i * (360 / anglesCount));
86 //  }
87 //  type-3
88 //  Angle[] directionAngles = new Angle[16];
89 //  double cameraWideAngle = cameraSpecification.getWideAngle().deg();
90 //  for (int i = 0; i < 4; i++) {
91 //      directionAngles[i * 4] = new Angle(i * 90);
92 //      directionAngles[(i * 4) + 1]
93 //          = new Angle((cameraWideAngle / 2) + (i * 90));
94 //      directionAngles[(i * 4) + 2] = new Angle((i * 90) + 45);
95 //      directionAngles[(i * 4) + 3]
96 //          = new Angle(((i + 1) * 90) - (cameraWideAngle / 2));
97 //  }
98 //  return directionAngles;
99 //  }
100 }

```

Listing A.10: MinimumCameraPlacementSolverLPSolve.java

```

1 package model;
2
3 import java.util.ArrayList;
4 import java.util.HashMap;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;
7 import lp_solve.LpSolve;
8 import lp_solve.LpSolveException;
9
10 /**
11  *
12  * @author Prayogo Cendra
13  */
14 public class MinimumCameraPlacementSolverLPSolve extends MinimumCameraPlacementSolver {
15
16     public MinimumCameraPlacementSolverLPSolve(Room room) {
17         super(room);
18     }
19
20     @Override
21     public void solve() {
22         Cell[] shouldBeCoveredCells = room.getShouldBeCoveredCell();
23         Point[] preferredPlacingPoints = room.getPreferredPlacingPoints();
24         Angle[] directionAngles = generateDirectionAngles();
25
26         int preferredPlacingPointsCount = preferredPlacingPoints.length;
27         int shouldBeCoveredCellsCount = shouldBeCoveredCells.length;
28         int directionAnglesCount = directionAngles.length;
29
30         HashMap<Cell, Integer> shouldBeCoveredCellIndexMap = new HashMap<>(shouldBeCoveredCellsCount);
31         for (int i = 0; i < shouldBeCoveredCellsCount; i++) {

```



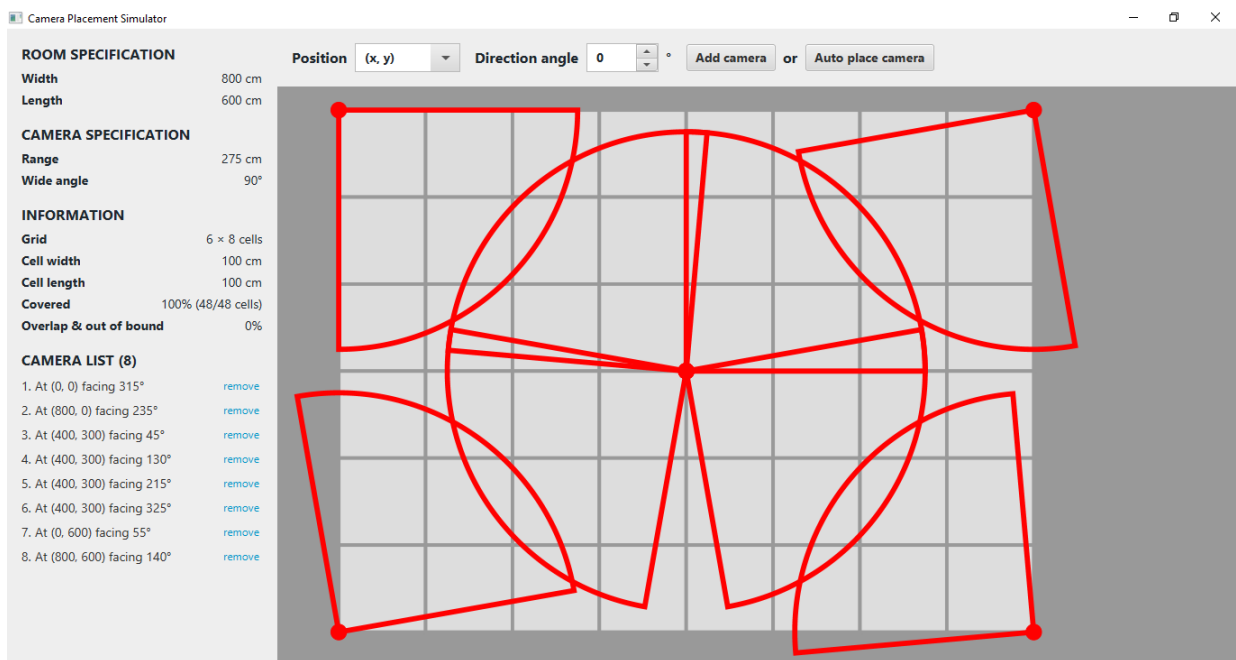
```

32 |         shouldBeCoveredCellIndexMap.put(shouldBeCoveredCells[i], i);
33 |     }
34 |
35 |     HashMap<CameraPlacement, Cell[]> possiblePlacementsCoverageMap = new HashMap<>();
36 |     ArrayList<CameraPlacement> possiblePlacementList = new ArrayList<>();
37 |
38 |     System.out.println("Finding_coverage...");
39 |     for (int i = 0; i < preferredPlacingPointsCount; i++) {
40 |         for (int j = 0; j < directionAnglesCount; j++) {
41 |             Point placingPoint = preferredPlacingPoints[i];
42 |             Angle directionAngle = directionAngles[j];
43 |             CameraPlacement placement
44 |                 = new CameraPlacement(placingPoint, directionAngle);
45 |             Cell[] coveredCells = room.findCoverage(placement);
46 |             ArrayList<Cell> correlatedCellList = new ArrayList<>();
47 |             for (Cell cell : coveredCells) {
48 |                 if (shouldBeCoveredCellIndexMap.containsKey(cell)) {
49 |                     correlatedCellList.add(cell);
50 |                 }
51 |             }
52 |             if (correlatedCellList.size() > 0) {
53 |                 Cell[] correlatedCells = new Cell[correlatedCellList.size()];
54 |                 correlatedCellList.toArray(correlatedCells);
55 |                 possiblePlacementsCoverageMap.put(placement, correlatedCells);
56 |                 possiblePlacementList.add(placement);
57 |             }
58 |         }
59 |     }
60 |
61 |     int possiblePlacementsCount = possiblePlacementList.size();
62 |     System.out.println(possiblePlacementsCount + "_possible_placements");
63 |
64 |     /**
65 |      * *****
66 |      */
67 |     double[][] lpConstraints = new double[shouldBeCoveredCellsCount][possiblePlacementsCount];
68 |     double[] lpConstraintsRhs = new double[shouldBeCoveredCellsCount];
69 |     double[] lpObjective = new double[possiblePlacementsCount];
70 |
71 |     for (int i = 0; i < possiblePlacementsCount; i++) {
72 |         lpObjective[i] = 1;
73 |         CameraPlacement placement = possiblePlacementList.get(i);
74 |         for (Cell cell : possiblePlacementsCoverageMap.get(placement)) {
75 |             lpConstraints[shouldBeCoveredCellIndexMap.get(cell)][i] = 1;
76 |         }
77 |     }
78 |
79 |     for (int i = 0; i < shouldBeCoveredCellsCount; i++) {
80 |         lpConstraintsRhs[i] = 1;
81 |     }
82 |
83 |     try {
84 |         LpSolve lp = LpSolve.makeLp(shouldBeCoveredCellsCount, possiblePlacementsCount);
85 |
86 |         System.out.println("Building_linear_programming...");
87 |
88 |         for (int i = 0; i < possiblePlacementsCount; i++) {
89 |             lp.setObj(i + 1, lpObjective[i]);
90 |             lp.setBinary(i + 1, true);
91 |         }
92 |         for (int i = 0; i < shouldBeCoveredCellsCount; i++) {
93 |             lp.setRowName(i + 1, shouldBeCoveredCells[i].toString());
94 |             for (int j = 0; j < possiblePlacementsCount; j++) {
95 |                 lp.setMat(i + 1, j + 1, lpConstraints[i][j]);
96 |             }
97 |             lp.setConstrType(i + 1, LpSolve.GE);
98 |             lp.setRh(i + 1, lpConstraintsRhs[i]);
99 |         }
100 |
101 |         // lp.printLp();
102 |         // lp.setTimeout(120); //seconds
103 |         System.out.println("Solving_linear_programming...");
104 |         lp.solve();
105 |         System.out.println("Done!");
106 |
107 |         double[] results = lp.getPtrVariables();
108 |         for (int i = 0; i < results.length; i++) {
109 |             if ((int) results[i] == 1) {
110 |                 room.addCameraPlacement(possiblePlacementList.get(i));
111 |             }
112 |         }
113 |
114 |         lp.deleteLp();
115 |     } catch (LpSolveException ex) {
116 |         Logger.getLogger(Room.class.getName()).log(Level.SEVERE, null, ex);
117 |     }
118 | }
119 |
120 | }

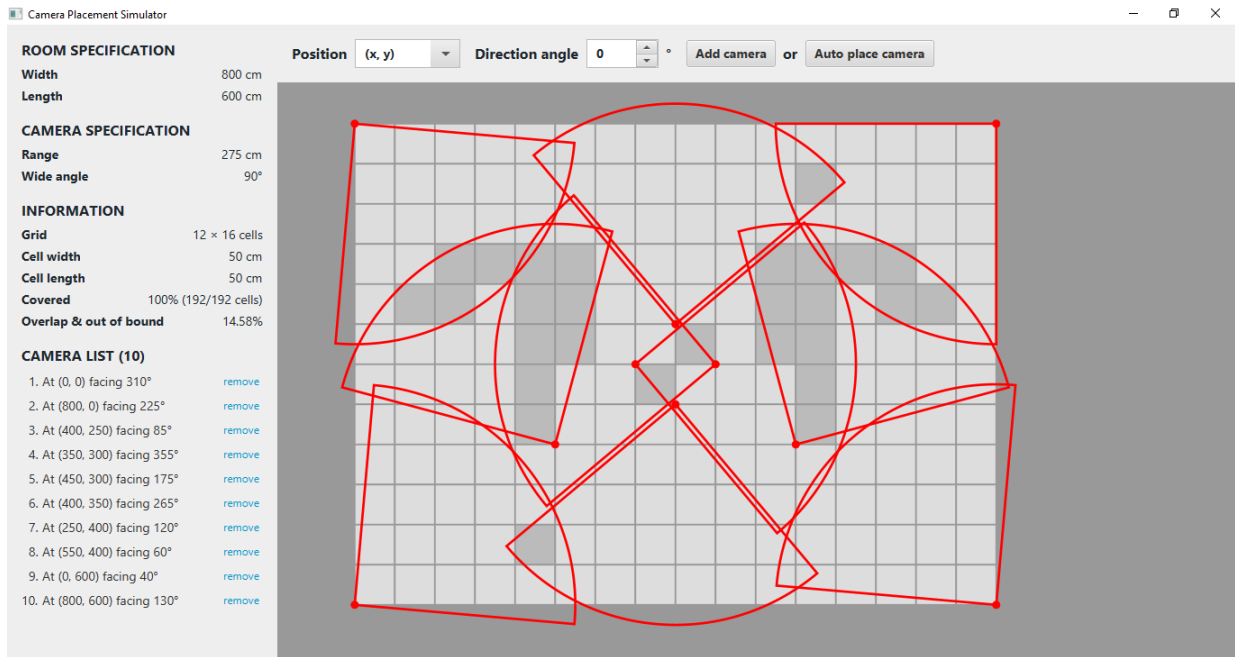
```


LAMPIRAN B

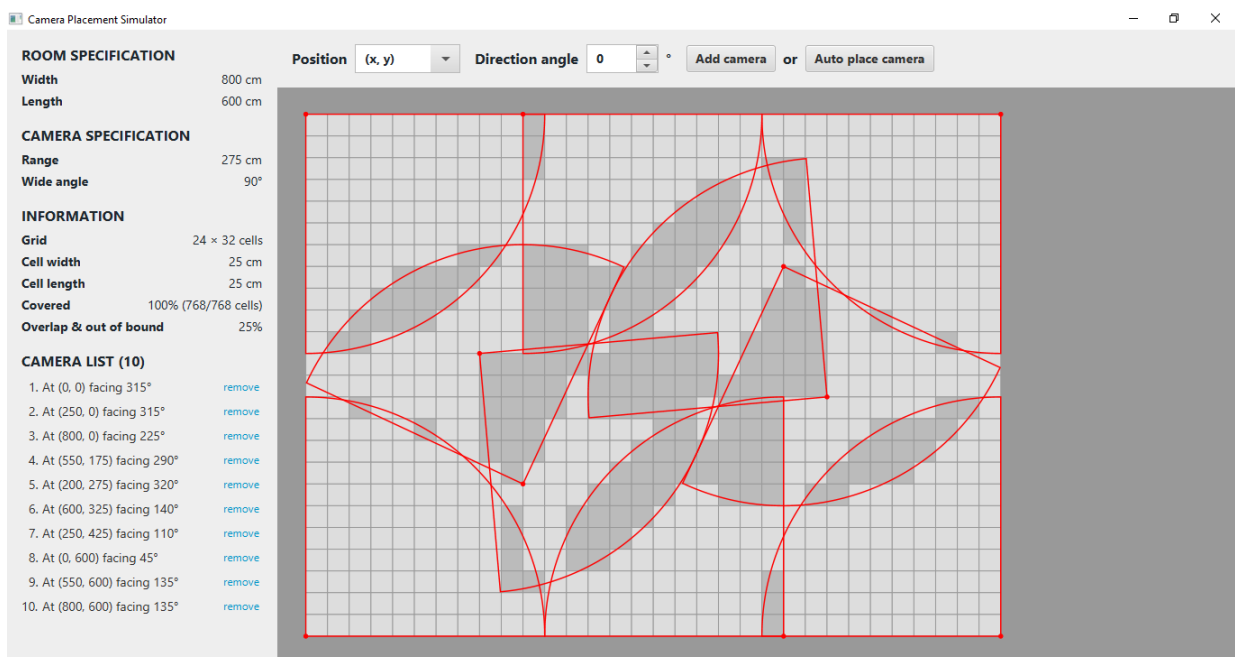
HASIL EKSPERIMEN



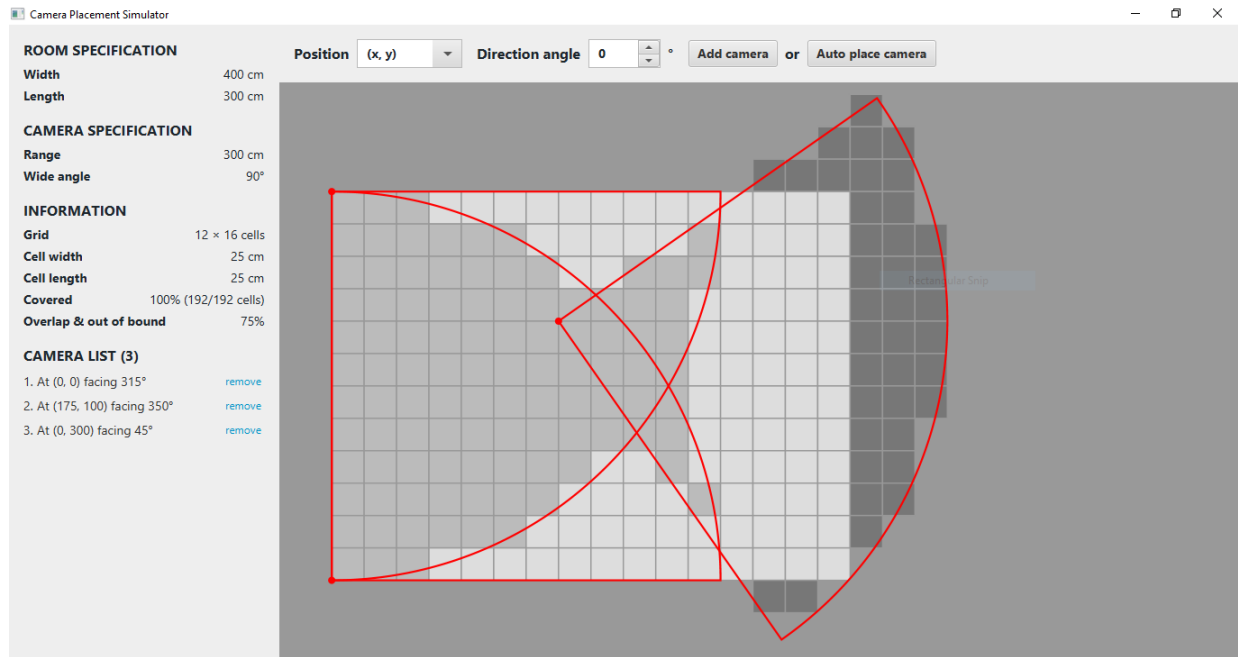
Gambar B.1: Hasil eksperimen ukuran cell, pertama



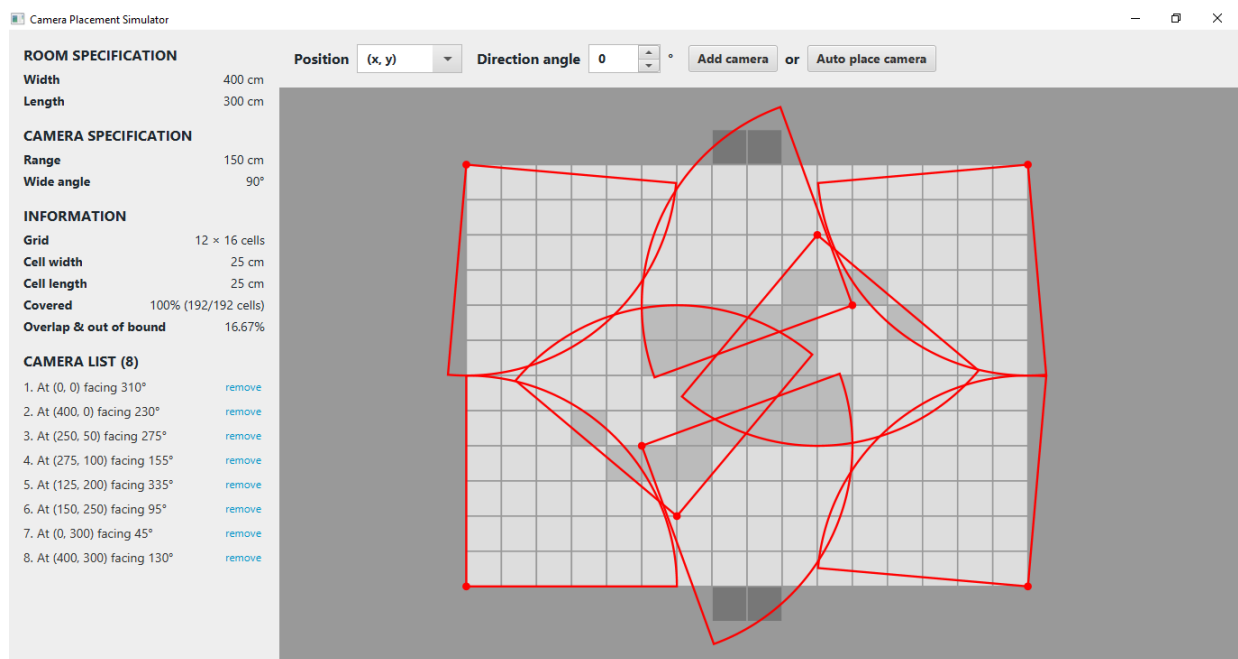
Gambar B.2: Hasil eksperimen ukuran cell, kedua



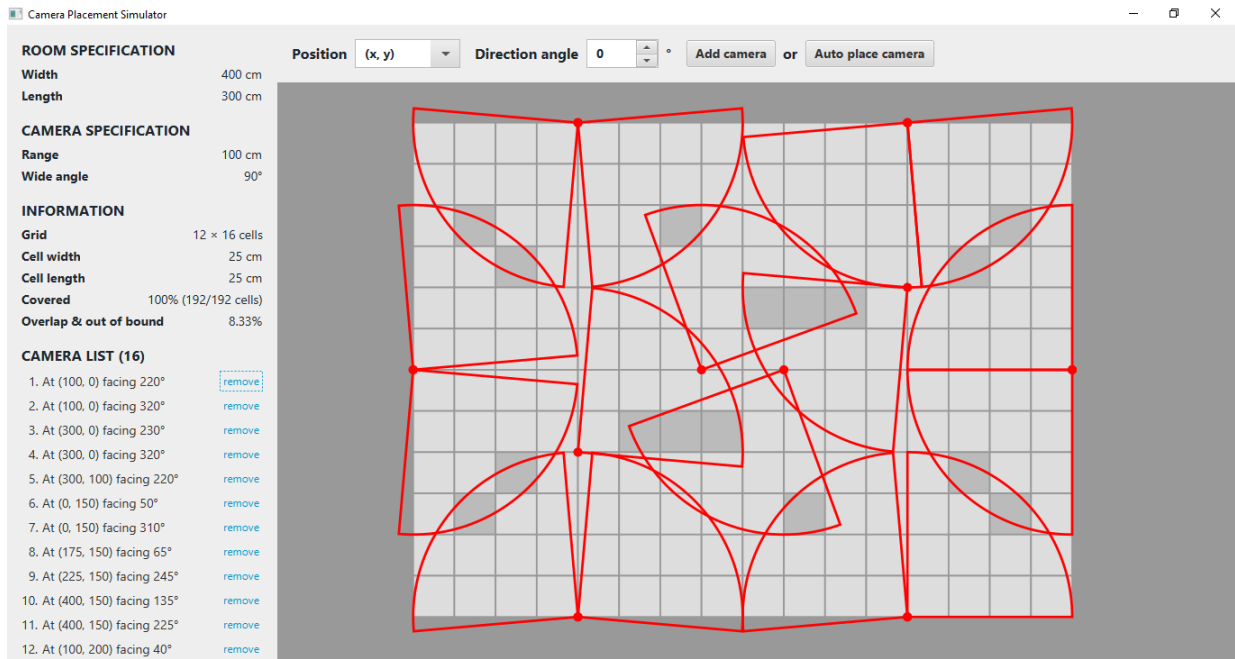
Gambar B.3: Hasil eksperimen ukuran cell, ketiga



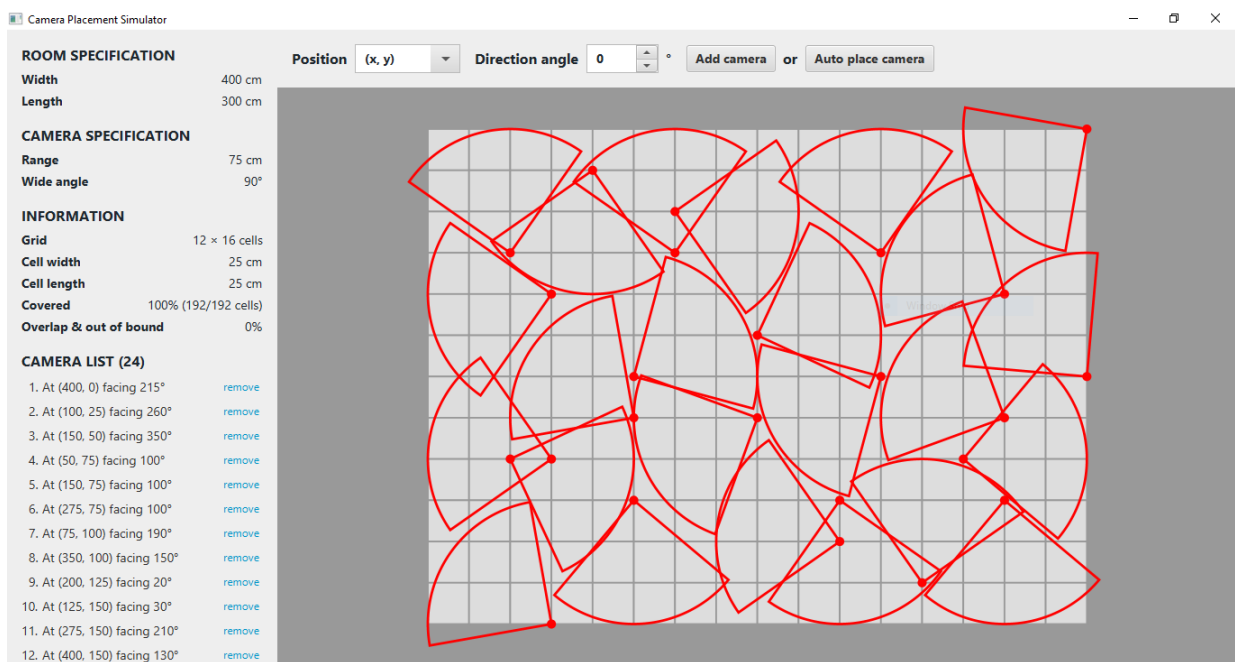
Gambar B.4: Hasil eksperimen rasio, pertama



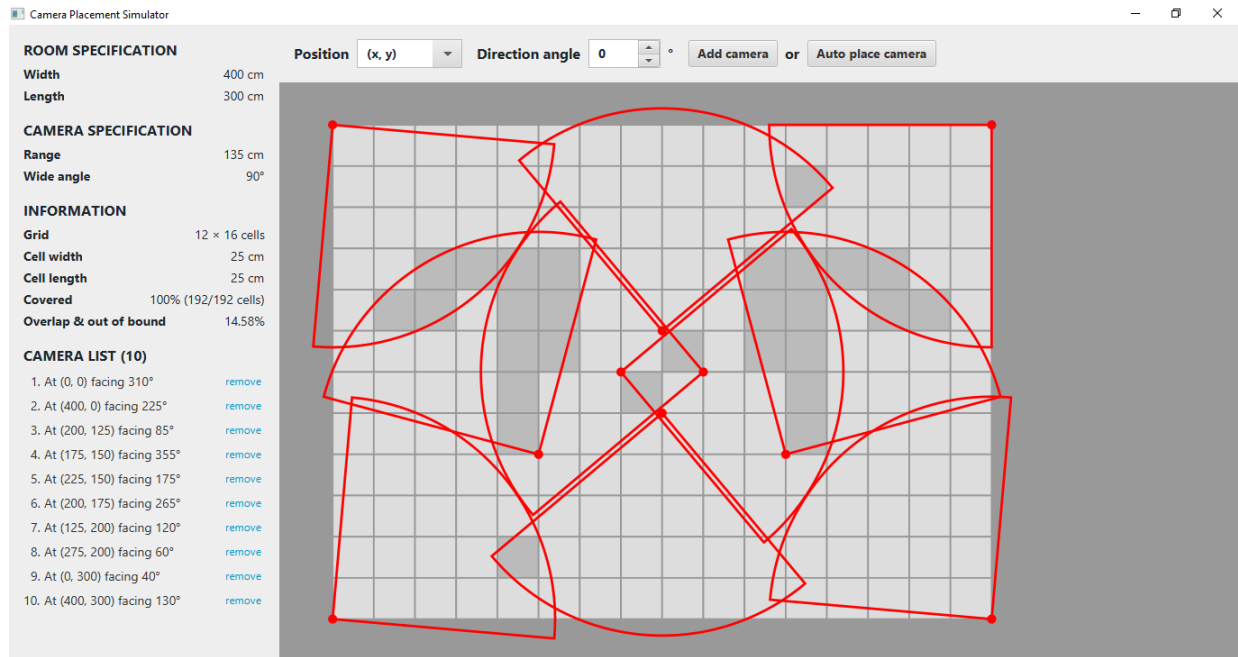
Gambar B.5: Hasil eksperimen rasio, kedua



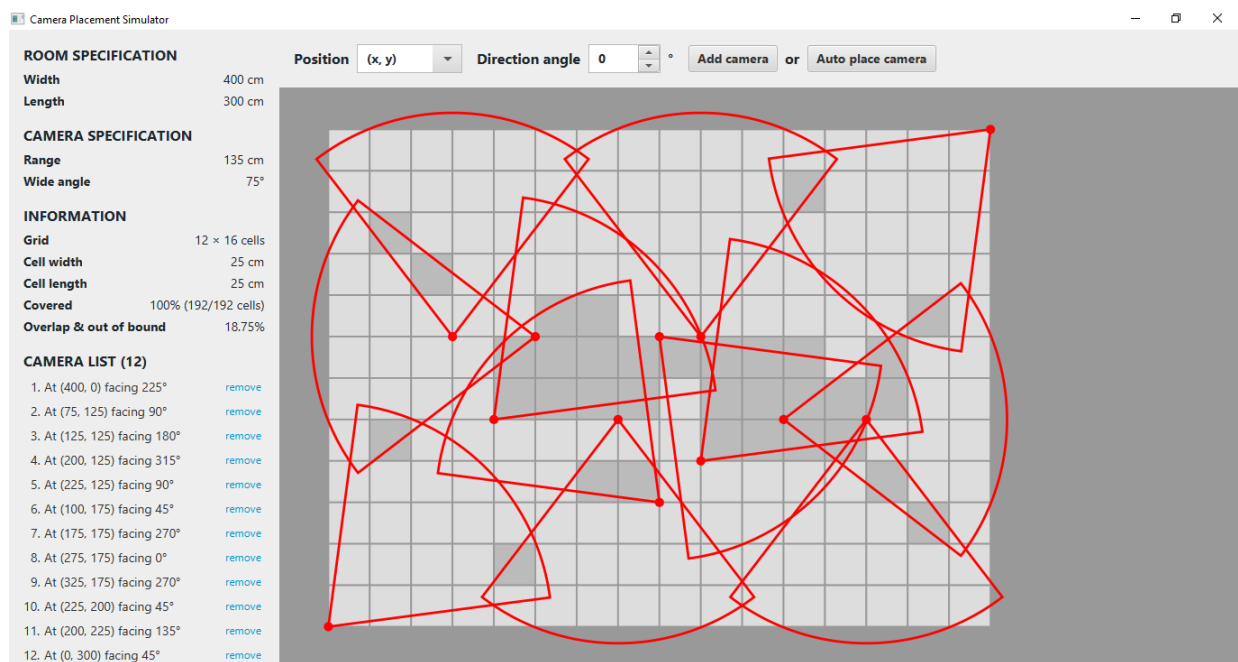
Gambar B.6: Hasil eksperimen rasio, ketiga



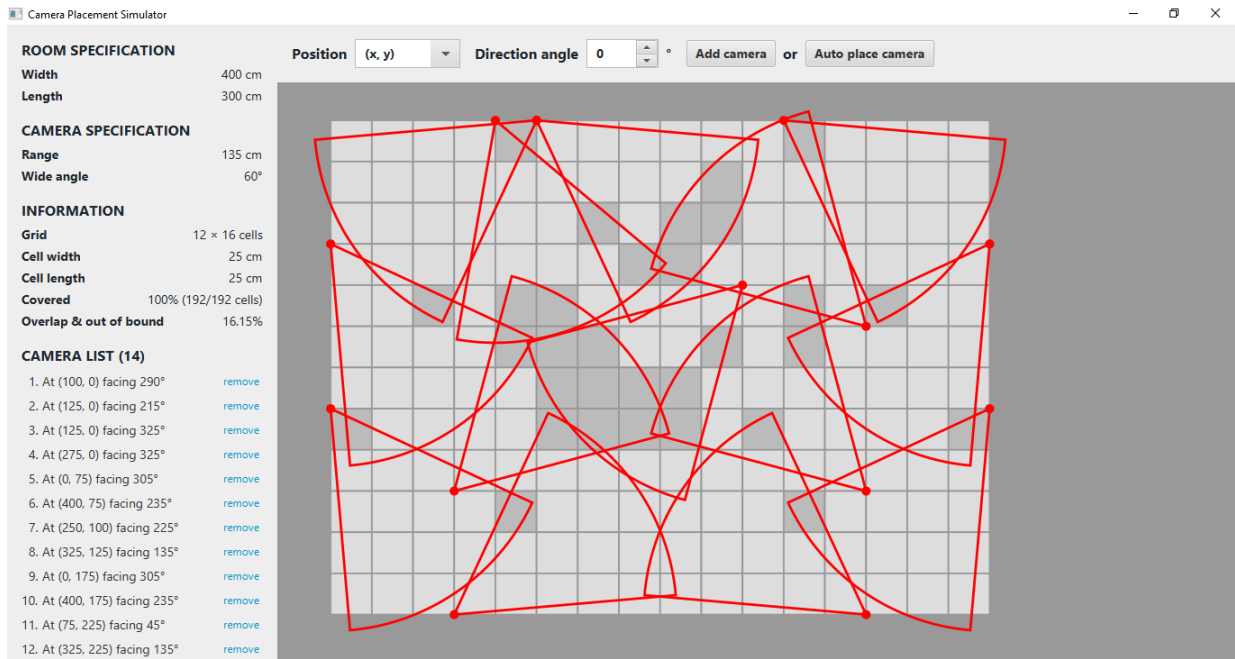
Gambar B.7: Hasil eksperimen rasio, keempat



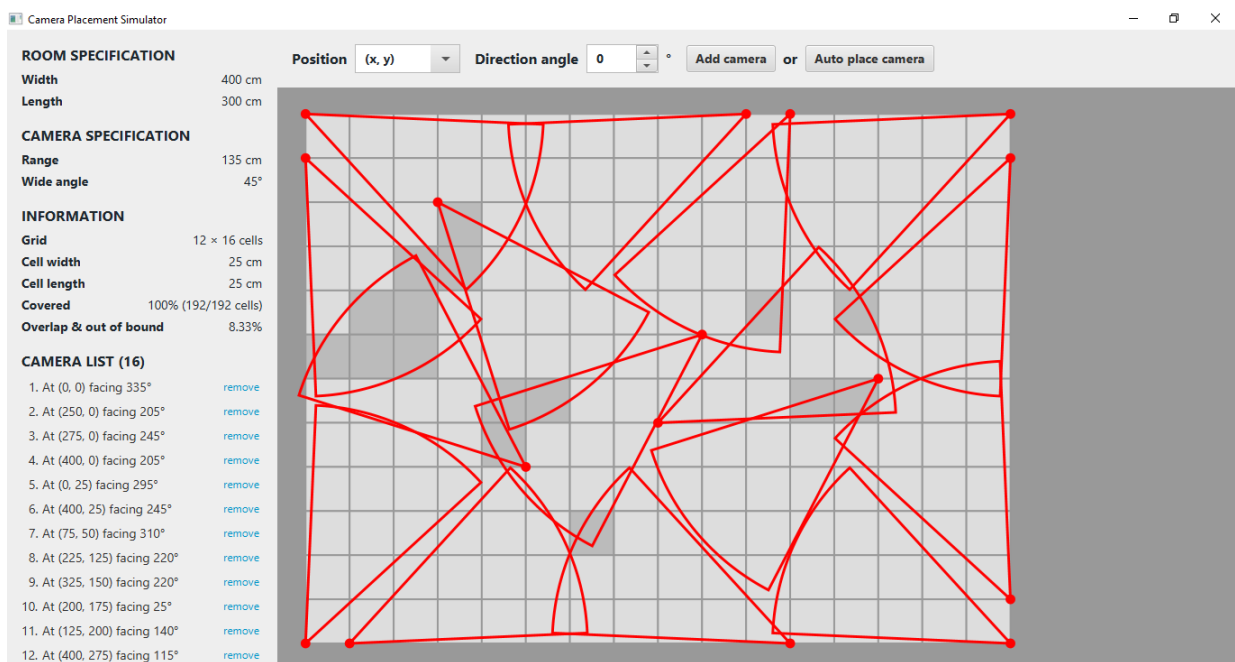
Gambar B.8: Hasil eksperimen besar sudut pandang, pertama



Gambar B.9: Hasil eksperimen besar sudut pandang, kedua



Gambar B.10: Hasil eksperimen besar sudut pandang, ketiga



Gambar B.11: Hasil eksperimen besar sudut pandang, keempat