

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**

LEMBAR PENGESAHAN

PENCARIAN JUMLAH KAMERA STATIS MINIMUM DALAM SUATU RUANGAN MENGGUNAKAN LINEAR PROGRAMMING

Prayogo Cendra

NPM: 2014730033

Bandung, 18 Mei 2018

Menyetujui,

Pembimbing

Claudio Franciscus, M.T.

Ketua Tim Penguji

Anggota Tim Penguji

«penguji 1»

«penguji 2»

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PENCARIAN JUMLAH KAMERA STATIS MINIMUM DALAM SUATU RUANGAN MENGGUNAKAN LINEAR PROGRAMMING

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 18 Mei 2018

Meterai Rp. 6000

Prayogo Cendra
NPM: 2014730033

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

«kepada siapa anda mempersembahkan skripsi ini...?»

KATA PENGANTAR

«Tuliskan kata pengantar dari anda di sini ...»

Bandung, Mei 2018

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xxi
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	1
1.3 Tujuan	2
1.4 Batasan Masalah	2
1.5 Metodologi	2
1.6 Sistematika Pembahasan	2
2 LANDASAN TEORI	5
2.1 <i>Linear Programming</i>	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	9
2.1.5 Metode <i>Simplex</i>	10
2.2 <i>Binary Integer Programming</i>	15
2.2.1 Algoritma <i>Balas's Additive</i>	15
2.3 Penelitian Terkait	17
3 ANALISIS	21
3.1 Pemodelan Masalah	21
3.1.1 Ruang	21
3.1.2 Kamera CCTV	23
3.1.3 Cakupan Kamera CCTV	23
3.2 Penyelesaian Masalah	26
3.2.1 Variabel	26
3.2.2 Fungsi Tujuan	27
3.2.3 Batasan	27
3.2.4 Model Masalah <i>Binary Integer Programming</i>	27
3.3 Analisis Kebutuhan Perangkat Lunak	28
3.3.1 Diagram <i>Use Case</i> dan Skenario	28
3.3.2 Kebutuhan Masukan Perangkat Lunak	28
3.3.3 Kebutuhan Keluaran Perangkat Lunak	29
3.3.4 Diagram Kelas Sederhana	29

4	PERANCANGAN	31
4.1	Perancangan Antarmuka	31
4.2	Perancangan Kelas	32
4.2.1	Kelas <i>Angle</i>	35
4.2.2	Kelas <i>Point</i>	36
4.2.3	Kelas <i>CameraSpecification</i>	37
4.2.4	Kelas <i>CameraPlacement</i>	37
4.2.5	Kelas <i>Dimension</i>	38
4.2.6	Kelas <i>Cell</i>	38
4.2.7	Kelas <i>CellMatrix</i>	39
4.2.8	Kelas <i>Room</i>	40
4.2.9	Kelas <i>MinimumCameraPlacementSolver</i>	41
4.2.10	Kelas <i>MinimumCameraPlacementSolverBalasAdditive</i>	42
4.2.11	Kelas <i>Variable</i>	42
4.2.12	Kelas <i>BIPFunction</i>	43
4.2.13	Kelas <i>BIPConstraint</i>	43
4.2.14	Kelas <i>BIPProblem</i>	44
4.2.15	Kelas <i>BIPFeasibilityCheckResult</i>	44
4.2.16	Kelas <i>Node</i>	45
4.2.17	Kelas <i>NodeStatus</i>	46
4.2.18	Kelas <i>BalasAdditiveBIPSolver</i>	46
4.2.19	Kelas <i>BalasAdditiveBIPSolverResult</i>	47
5	IMPLEMENTASI DAN PENGUJIAN	49
5.1	Lingkungan Implementasi Perangkat Keras	49
5.2	Lingkungan Implementasi Perangkat Keras	49
5.3	Implementasi Antarmuka	49
5.4	Pengujian Fungsional	51
5.5	Pengujian Eksperimental	55
5.5.1	Eksperimen Ukuran Cell	55
5.5.2	Eksperimen Rasio Sisi Terpendek Ruangan dengan Jarak Pandang Kamera CCTV	56
5.5.3	Eksperimen Besar Sudut Pandang Kamera CCTV	57
6	KESIMPULAN DAN SARAN	59
6.1	Kesimpulan	59
6.2	Saran	59
	DAFTAR REFERENSI	61
	A KODE PROGRAM	63
	B HASIL EKSPERIMEN	65

DAFTAR GAMBAR

2.1	Contoh masalah <i>linear programming</i> dengan daerah <i>feasible</i>	7
2.2	<i>Corner points</i> pada daerah <i>feasible</i>	7
2.3	Contoh masalah <i>linear programming</i> tanpa daerah <i>feasible</i>	8
2.4	Pemodelan daerah cakupan kamera	17
3.1	Pemodelan ruangan	22
3.2	Pemecahan ruangan menjadi matriks <i>cell</i>	22
3.3	Pemodelan kamera CCTV	23
3.4	Penempatan kamera CCTV dalam ruangan	24
3.5	Cakupan kamera CCTV yang terdiri dari kumpulan <i>cell</i>	24
3.6	<i>Overlap cell</i> dan <i>out of bound cell</i>	26
3.7	Diagram <i>use case</i>	28
3.8	Diagram kelas sederhana untuk <i>package</i> model	29
3.9	Diagram kelas sederhana untuk <i>package</i> bip dan <i>subpackage</i> <i>balasadditive</i>	30
4.1	Perancangan antarmuka penerima masukan	31
4.2	Perancangan antarmuka penempatan kamera CCTV	32
4.3	Diagram kelas rinci untuk <i>package</i> model	33
4.4	Diagram kelas rinci untuk <i>package</i> bip dan <i>subpackage</i> <i>balasadditive</i>	34
4.5	Diagram kelas <i>Angle</i>	35
4.6	Diagram kelas <i>Point</i>	36
4.7	Diagram kelas <i>CameraSpecification</i>	37
4.8	Diagram kelas <i>CameraPlacement</i>	37
4.9	Diagram kelas <i>Dimension</i>	38
4.10	Diagram kelas <i>Cell</i>	38
4.11	Diagram kelas <i>CellMatrix</i>	39
4.12	Diagram kelas <i>Room</i>	40
4.13	Diagram kelas <i>MinimumCameraPlacementSolver</i>	41
4.14	Diagram kelas <i>MinimumCameraPlacementSolverBalasAdditive</i>	42
4.15	Diagram kelas <i>Variable</i>	42
4.16	Diagram kelas <i>BIPFunction</i>	43
4.17	Diagram kelas <i>BIPConstraint</i>	43
4.18	Diagram kelas <i>BIPProblem</i>	44
4.19	Diagram kelas <i>BIPFeasibilityCheckResult</i>	44
4.20	Diagram kelas <i>Node</i>	45
4.21	Diagram kelas <i>NodeStatus</i>	46
4.22	Diagram kelas <i>BalasAdditiveBIPSolver</i>	46
4.23	Diagram kelas <i>BalasAdditiveBIPSolverResult</i>	47
5.1	Antarmuka penerima masukan	50
5.2	Antarmuka penempatan kamera CCTV	50
5.3	Tampilan pengisian masukan masalah	51
5.4	Tampilan setelah pengisian masukan masalah	52

5.5	Tampilan panel informasi setelah pengisian masukan masalah	52
5.6	Tampilan panel penambahan penempatan kamera CCTV	53
5.7	Tampilan panel informasi setelah menambah penempatan kamera CCTV	53
5.8	Tampilan panel visualisasi setelah menambah penempatan kamera CCTV	53
5.9	Tampilan panel informasi setelah membuang penempatan kamera CCTV	54
5.10	Tampilan panel visualisasi setelah membuang penempatan kamera CCTV	54

DAFTAR TABEL

2.1	Tabel <i>simplex</i> 0	10
2.2	Proses <i>pivotting</i> 1	11
2.3	Tabel <i>simplex</i> 1	12
2.4	Proses <i>pivotting</i> 2	13
2.5	Tabel <i>simplex</i> 2	14
2.6	Tabel <i>simplex</i> 0 untuk kasus minimasi	15
2.7	Tabel <i>simplex</i> 1 untuk kasus minimasi	15

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 ruangam 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 CCTV dapat dipahami dengan lebih baik.

1.2 Rumusan Masalah

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

- Bagaimana cara mencari jumlah minimum penempatan kamera CCTV dalam suatu ruangan

yang dapat mencakup seluruh ruangan?

- Bagaimana cara memvisualisasikan penempatan-penempatan kamera CCTV dalam suatu ruangan?

1.3 Tujuan

Berdasarkan rumusan masalah, maka tujuan dalam skripsi ini adalah:

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

1.4 Batasan Masalah

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

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

1.5 Metodologi

- Mempelajari metode linear programming
- Mempelajari metode integer programming
- Mempelajari penggunaan kakas *lp_solve*
- Melakukan pemodelan masalah
- Menerapkan metode linear programming dalam penyelesaian masalah
- Merancang perangkat lunak
- Membangun perangkat lunak
- Menguji perangkat lunak
- 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*

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*:

$$\begin{aligned} \max \quad & z = 3x_1 + 2x_2 \\ \text{s.t.} \quad & 2x_1 + x_2 \leq 100 \\ & x_1 + x_2 \leq 80 \\ & x_1 \leq 40 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned}$$

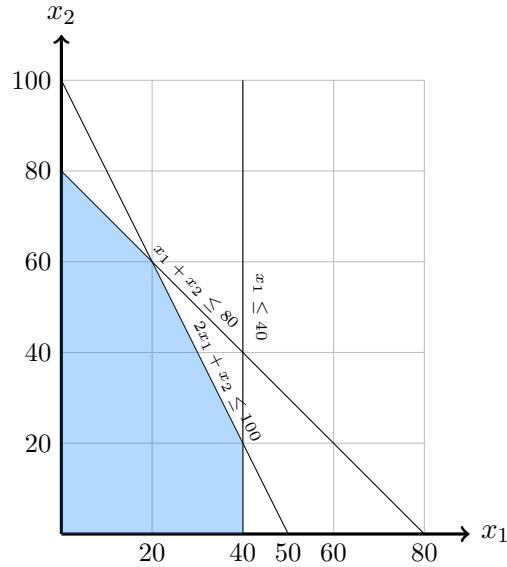
Tulisan "*s.t.*" atau "*subject to*" menandakan bahwa nilai dari setiap variabel keputusan harus memenuhi setiap batasan dan batasan non-negatif.

2.1.2 Daerah *Feasible* dan Solusi Optimal

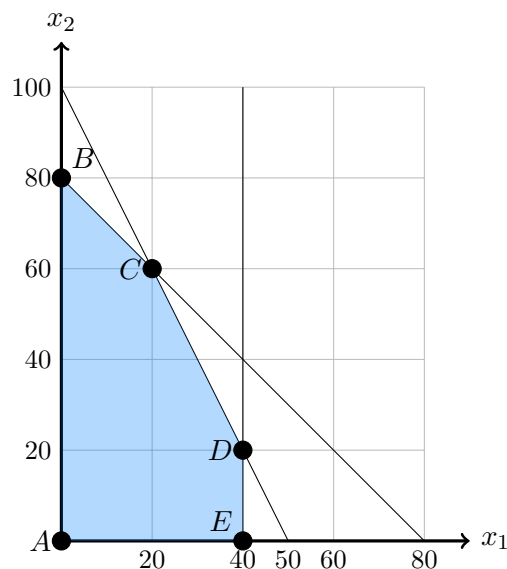
Dalam masalah *linear programming*, terdapat daerah yang bernama daerah *feasible*. Daerah *feasible* dalam suatu masalah *linear programming* merupakan himpunan yang terdiri dari seluruh titik yang memenuhi setiap batasan dan batasan non-negatif [1]. Dengan demikian, setiap titik yang berada di dalam 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:

$$\begin{aligned} 2x_1 + x_2 &\leq 100 \\ x_1 + x_2 &\leq 80 \\ x_1 &\leq 40 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

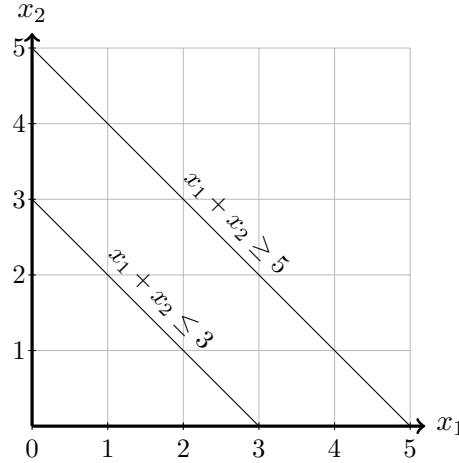
Pada contoh masalah tersebut, hanya terdapat 2 variabel keputusan sehingga dapat digambarkan dalam diagram kartesius. Pada gambar 2.1 terlihat bahwa batasan-batasan membentuk daerah *feasible*.

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

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

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

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



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

2.1.3 Bentuk Standar

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

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

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

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

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

$$\begin{aligned} x_1 + 2x_2 + s_1 &= 40 \\ s_1 &\geq 0 \end{aligned}$$

- **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

$$\begin{aligned} x_1 + 2x_2 - e_1 &= 40 \\ e_1 &\geq 0 \end{aligned}$$

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.

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

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

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

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

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

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	$s_4 = 5$

Kolom *right hand side* (rhs) menunjukkan nilai pada ruas kanan (nilai di sebelah kanan tanda sama dengan). Kolom *basic variable* menunjukkan variabel basis beserta dengan nilainya. Kolom *basic variable* juga menunjukkan solusi basis *feasible* (bfs) sehingga tabel 2.1 menunjukkan bfs 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 yang tidak berada di dalam kolom *basic variable* menandakan bahwa variabel-variabel tersebut merupakan variabel non-basis sehingga bernilai 0.

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

Apabila bfs pada saat ini masih belum optimal, maka perlu dilakukan proses *pivoting*. Proses *pivoting* 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 *pivoting*. 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 *pivoting* 1

z	x_1	x_2	x_3	s_1	s_2	s_3	s_4	rhs	$basic\ variable$	$ratio$
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	$basic\ variable$
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 *pivoting* 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:

$$\begin{aligned} \min \quad & z = 2x_1 - 3x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq 4 \\ & x_1 - x_2 \leq 6 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned}$$

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:

$$\begin{aligned} \max \quad & -z = -2x_1 + 3x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leq 4 \\ & x_1 - x_2 \leq 6 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned}$$

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 Binary Integer Programming

Binary integer programming adalah lanjutan dari *linear programming* (2.1) di mana setiap variabel keputusan pada solusi optimal harus bernilai 0 atau 1. Bentuk umum masalah *binary integer programming* adalah seperti berikut:

$$\begin{aligned}
 \min \quad & z = C^T x \\
 \text{s.t.} \quad & Ax \geq b \\
 & x \in \{0, 1\}
 \end{aligned} \tag{2.1}$$

Terdapat sebuah algoritma yang diciptakan oleh Egon Balas [2] bernama *Balas's additive* yang digunakan untuk menyelesaikan masalah *binary integer programming*. Algoritma *Balas's additive* dibahas lebih lanjut pada 2.2.1.

2.2.1 Algoritma Balas's Additive

Setiap masalah *binary integer programming* memiliki kemungkinan solusi berjumlah 2^n karena setiap variabel hanya dapat bernilai 0 atau 1 saja. Apabila menggunakan algoritma *Balas's additive*, maka solusi optimal dari masalah *binary integer programming* dapat ditemukan tanpa perlu memeriksa seluruh 2^n kemungkinan. Dengan demikian, algoritma *Balas's additive* dapat digunakan untuk menemukan solusi optimal bagi masalah *binary integer programming* secara efisien.

Untuk menyelesaikan masalah *binary integer programming* menggunakan algoritma *Balas's additive*, maka masalah perlu diubah terlebih dahulu berdasarkan ketentuan-ketentuan berikut ini:

- Fungsi tujuan dinyatakan dalam bentuk $\min z = \sum_{j=1}^n c_j x_j$.
- Sebanyak m batasan harus dinyatakan dalam bentuk pertidaksamaan $\sum a_{ij} x_j \geq b_i$ untuk $i = 1, 2, \dots, m$.
- Setiap variabel x_j dimana $j = 1, 2, \dots, n$ harus merupakan variabel biner (variabel yang hanya dapat bernilai 0 atau 1).
- Koefisien dari setiap variabel pada fungsi tujuan tidak bernilai negatif.
- Setiap variabel pada fungsi tujuan diurutkan menaik berdasarkan koefisiennya sehingga $0 \leq c_1 \leq c_2 \leq \dots \leq c_n$.

Pengurutan variabel secara menaik berdasarkan koefisiennya bertujuan agar nilai z meningkat seminimum mungkin pada setiap iterasinya. Mulanya algoritma *Balas's additive* akan membuat seluruh variabel bernilai 0 agar mendapatkan nilai z yang paling minimum. Apabila solusi bersifat *infeasible*, maka algoritma secara beriterasi akan membuat variabel-variabel terdekat dengan indeks 1 untuk bernilai 1. Dengan demikian, nilai z akan meningkat dengan nilai seminimum mungkin pada setiap iterasinya karena variabel telah terurut menaik berdasarkan koefisiennya.

Algoritma *Balas's additive* menggunakan paradigma algoritma *branch and bound* [3] untuk mendapatkan solusi optimal dari seluruh kemungkinan solusi yang ada. *Branch and bound* merupakan algoritma yang dapat mencari solusi terbaik dari suatu himpunan solusi tanpa melibatkan *exhaustive search*. *Branch and bound* bekerja dengan cara membagi dan memeriksa subhimpunan solusi secara rekursi hingga mendapatkan subhimpunan yang memiliki solusi yang paling baik. Setiap subhimpunan memiliki nilai *bound* yang menyatakan kualitas dari solusi yang dimiliki oleh masing-masing subhimpunan. Nilai *bound* ini menjadi pembanding sebelum melakukan pembagian subhimpunan. Apabila nilai *bound* suatu subhimpunan lebih buruk daripada solusi yang telah ditemukan sebelumnya, maka pembagian subhimpunan tidak dilakukan karena telah dipastikan bahwa subhimpunannya tidak akan dapat menghasilkan solusi yang lebih baik. *Branch and bound* menggunakan struktur data *tree* di mana *root* menunjukkan himpunan keseluruhan solusi dan cabang menunjukkan subhimpunannya. Terdapat 2 proses utama dalam *branch and bound*, yaitu *branching* dan *bounding*. *Branching* berfungsi untuk membagi himpunan solusi menjadi beberapa subhimpunan yang lebih kecil. *Bounding* berfungsi untuk menetapkan nilai *bound* dari suatu subhimpunan.

Branch and bound pada algoritma *Balas's additive* menggunakan struktur data *binary tree* karena setiap percabangan menghasilkan 2 *node* di mana salah satu *node* menunjukkan variabel bernilai 0 dan *node* lainnya menunjukkan variabel bernilai 1. Setiap *node* menunjukkan nilai dari variabel yang ditujunya sesuai dengan tingkat kedalaman *node* pada pohon biner. *Branching* terhadap suatu *node* akan menghasilkan 2 *node* baru, yaitu *node* $x_N = 0$ dan *node* $x_N = 1$ di mana x_N merupakan variabel x yang diacu oleh *node* pada saat ini yang berbeda dengan variabel terakhir (x_n) dalam kumpulan x . *Bounding* pada algoritma *Balas's additive* dibedakan menjadi 2 berdasarkan nilai x_N , yaitu:

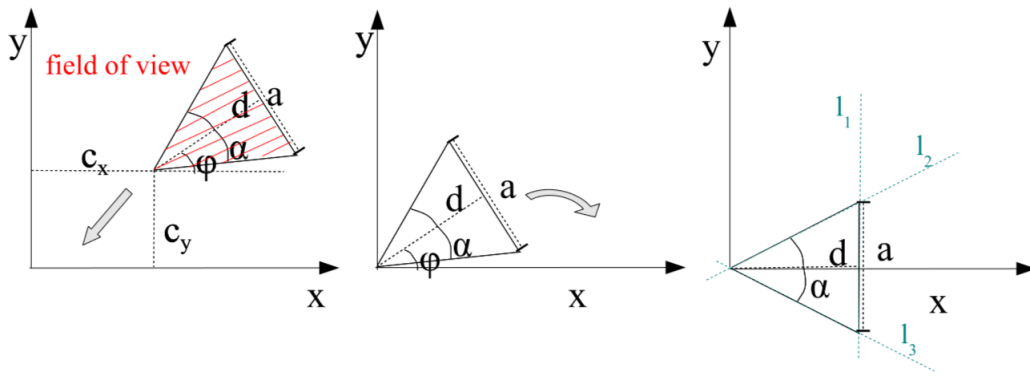
- Apabila $x_N = 1$, maka algoritma berasumsi bahwa *node* pada saat ini dapat menghasilkan solusi *feasible* sehingga nilai *bound* untuk *node* ini adalah $\sum_{j=1}^N c_j x_j$. Karena variabel telah diurutkan sebelumnya, maka nilai *bound* ini dipastikan menjadi yang paling rendah untuk saat ini.
- Apabila $x_N = 0$, maka nilai *bound* dari *node* ini adalah $\sum_{j=1}^N c_j x_j + c_{N+1}$. *Node* ini terbentuk karena *node* sebelumnya memiliki solusi yang bersifat *infeasible*. Hal tersebut disebabkan oleh adanya batasan \geq yang tidak terpenuhi karena total ruas kiri yang lebih kecil daripada ruas kanan. Apabila variabel saat ini bernilai 0, maka batasan tersebut tetap tidak akan terpenuhi sehingga setidaknya harus terdapat 1 variabel lainnya yang bernilai 1, yaitu x_{N+1} . Variabel tersebut dipilih untuk memastikan bahwa nilai *bound* pada saat ini merupakan yang paling rendah.

Setiap *node* menunjukkan solusi yang belum tentu bersifat *feasible* sehingga perlu dilakukannya pengecekan. Untuk menentukan apakah solusi bersifat *feasible*, maka akan diasumsikan bahwa variabel setelah x_N (saat $x_N = 1$) atau x_{N+1} (saat $x_N = 0$) bernilai 0. Selanjutnya, solusi ini akan diuji terhadap setiap batasan. Apabila tidak terdapat batasan yang dilanggar, maka solusi bersifat *feasible* dan *node* dinyatakan *fathomed* sehingga tidak perlu dilakukan *branching*. *Branching* tidak perlu dilakukan karena turunannya tidak akan menghasilkan solusi yang lebih baik. Hal ini telah dipastikan ketika melakukan proses *bounding*. Solusi *feasible* ini kemudian dibandingkan dengan *incumbent* yang telah ditemukan sebelumnya. Apabila lebih baik, maka solusi ini menjadi *incumbent* yang baru. Apabila *node* memiliki solusi yang bersifat *infeasible*, maka akan dilakukan *branching*.

Namun sebelum melakukan *branching*, akan dilakukan pengecekan untuk mengetahui apakah turunan dari *node* dapat menghasilkan solusi *feasible*. Pengecekan dilakukan dengan memastikan bahwa setiap batasan memiliki kemungkinan untuk dipenuhi, yaitu dengan memastikan bahwa ruas kiri dapat menghasilkan total yang tidak lebih kecil daripada ruas kanan. Total ruas kiri terbesar dapat dicari dengan menjumlahkan ruas kiri untuk variabel yang telah didapatkan hingga saat ini dengan total penjumlahan nilai koefisien positif pada variabel tersisa. Apabila terdapat 1 atau lebih batasan di mana total ruas kiri terbesar bernilai lebih kecil dibandingkan dengan nilai ruas kanan, maka *node* tersebut dinyatakan *fathomed* sehingga tidak perlu dilakukan *branching*. *Branching* tidak perlu dilakukan karena batasan tersebut tidak akan dapat dipenuhi dengan berapa pun nilai dari variabel yang tersisa. Sebaliknya, apabila setiap batasan memiliki kemungkinan untuk dipenuhi, maka *branching* akan dilakukan pada *node* tersebut dan seluruh proses sebelumnya akan kembali dilakukan.

2.3 Penelitian Terkait

Penelitian untuk mencari penempatan sensor visual secara optimal telah dilakukan sebelumnya oleh Horster dan Lienhart [4]. Masalah yang dibahas pada penelitian mereka adalah cara menentukan jumlah sensor visual minimum beserta dengan lokasi penempatan dan arah pandangnya. Masalah ini dibatasi pada bidang 2D dan ruangan yang digunakan diasumsikan berbentuk persegi panjang.



Gambar 2.4: Pemodelan daerah cakupan kamera

Pada penelitian ini, daerah cakupan kamera dimodelkan dalam bentuk segitiga seperti pada gambar 2.4. Kamera ditempatkan pada posisi c_x, c_y dan menghadap ke arah φ . Untuk mendapatkan daerah cakupan kamera, mulanya kamera ditranslasi ke titik *origin* pada sistem koordinat:

$$x' = x - c_x \quad (2.2)$$

$$y' = y - c_y \quad (2.3)$$

Selanjutnya, daerah cakupan diputar sehingga sumbu pandang menjadi paralel terhadap sumbu x:

$$x'' = \cos(\varphi) \cdot x' + \sin(\varphi) \cdot y' \quad (2.4)$$

$$y'' = -\sin(\varphi) \cdot x' + \cos(\varphi) \cdot y' \quad (2.5)$$

Daerah cakupan kamera dapat ditentukan dengan garis l_1, l_2, l_3 :

$$l_1 : x'' \leq d \quad (2.6)$$

$$l_2 : y'' \leq \frac{a}{2d} \cdot x'' \quad (2.7)$$

$$l_3 : y'' \geq -\frac{a}{2d} \cdot x'' \quad (2.8)$$

Dengan substitusi, didapatkan daerah cakupan kamera berdasarkan tiga persamaan berikut ini:

$$\cos(\varphi) \cdot (x - c_x) + \sin(\varphi) \cdot (y - c_y)' \leq d \quad (2.9)$$

$$-\sin(\varphi) \cdot (x - c_x) + \cos(\varphi) \cdot (y - c_y) \leq \frac{a}{2d} \cdot (\cos(\varphi) \cdot (x - c_x) + \sin(\varphi) \cdot (y - c_y)) \quad (2.10)$$

$$-\sin(\varphi) \cdot (x - c_x) + \cos(\varphi) \cdot (y - c_y) \geq -\frac{a}{2d} \cdot (\cos(\varphi) \cdot (x - c_x) + \sin(\varphi) \cdot (y - c_y)) \quad (2.11)$$

Ruangan dimodelkan menggunakan *grid point* 2 dimensi dengan jarak antar titik sebesar f_a . Kamera hanya dapat diletakkan pada titik *grid point* dan cakupan kamera hanya diperiksa terhadap titik *grid point*. Ruangan berbentuk persegi panjang sehingga ukuran ruangan terdiri dari lebar w dan tinggi h . Di dalam ruangan tidak terdapat penghalang apapun.

Titik penempatan kamera dinyatakan dalam himpunan s_x dan himpunan s_y sesuai dengan dimensi x - dan y - pada *grid point*. Selain itu, terdapat himpunan s_φ yang menyatakan arah pandang yang memungkinkan bagi setiap kamera. Kamera yang ditempatkan pada (c_x, c_y) dengan arah pandang φ dapat mencakup *grid point* (x, y) jika pernyataan 2.9, 2.10, dan 2.11 dipenuhi.

Masalah dimodelkan ke dalam bentuk masalah *binary integer programming* dengan tujuan mendapatkan jumlah kamera minimum berdasarkan *grid point* dan model kamera yang diberikan dengan tetap memastikan bahwa setiap titik pada *grid point* tercakup oleh setidaknya satu kamera. Berikut ini merupakan definisi variabel biner $x_{ij\varphi}$ yang digunakan dalam model masalah *binary integer programming*:

$$x_{ij\varphi} = \begin{cases} 1 & \text{jika kamera ditempatkan pada } \textit{grid point} (i, j) \text{ dengan arah pandang } \varphi \\ 0 & \text{jika sebaliknya} \end{cases} \quad (2.12)$$

Total kamera (N) didapatkan dengan fungsi tujuan berikut ini:

$$N = \sum_{\varphi=0}^{s_\varphi-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{ij\varphi} \quad (2.13)$$

Untuk menentukan ketercakupannya titik, dibentuk fungsi biner $c(i1, j1, \varphi1, i2, j2)$:

$$c(i1, j1, \varphi1, i2, j2) = \begin{cases} 1 & \text{jika kamera yang ditempatkan pada } \textit{grid point} (i1, j1) \\ & \text{dengan arah pandang } \varphi1 \text{ dapat mencakup } \textit{grid point} (i2, j2) \\ 0 & \text{jika sebaliknya} \end{cases} \quad (2.14)$$

Dengan demikian masalah dapat diformulasikan ke dalam bentuk masalah *binary integer program-*

ming berikut:

$$\begin{aligned}
 \min \quad & \sum_{\varphi=0}^{s_{\varphi}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{ij\varphi} \\
 \text{s.t.} \quad & \sum_{\varphi=0}^{s_{\varphi}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{i1,j1,\varphi1} \cdot c(i1,j1,\varphi1,i2,j2) \geq 1 \\
 & 0 \leq i2 \leq (s_x - 1), 0 \leq j2 \leq (s_y - 1)
 \end{aligned} \tag{2.15}$$

Batasan pada 2.15 memastikan bahwa setiap titik pada *grid point* akan dicakup oleh setidaknya 1 kamera. Untuk memastikan bahwa hanya terdapat maksimal 1 kamera yang dapat ditempatkan pada setiap titik, maka dapat ditambahkan batasan berikut ini:

$$\begin{aligned}
 & \sum_{\varphi=0}^{s_{\varphi}-1} x_{ij\varphi} \leq 1 \\
 & 0 \leq i \leq (s_x - 1), 0 \leq j \leq (s_y - 1)
 \end{aligned} \tag{2.16}$$

Jumlah variabel $x_{ij\varphi}$ dalam model *binary integer programming* ini adalah sebesar $s_x \times s_y \times s_{\varphi}$. Jika ukuran *grid point* diperbesar, maka jumlah variabel dan batasan pada model *binary integer programming* juga akan semakin besar. Karena masalah ini merupakan masalah *binary integer programming* (2.2), maka masalah ini dapat diselesaikan dengan menggunakan algoritma *Balas's additive* (2.2.1).

BAB 3

ANALISIS

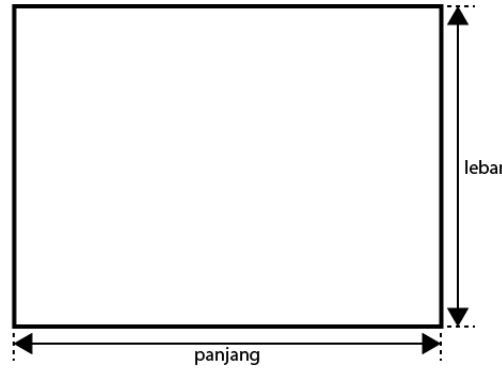
Pada bagian ini, masalah akan dianalisis lebih lanjut. Awalnya, masalah akan dimodelkan terlebih dahulu sebelum diselesaikan. Hasil dari pemodelan masalah akan digunakan untuk membentuk model masalah *binary integer programming* (2.2) yang akan diselesaikan menggunakan algoritma *Balas's additive* (2.2.1). Dengan model masalah *binary integer programming*, solusi optimal masalah dapat ditemukan, yaitu solusi berupa penempatan kamera CCTV yang berjumlah minimum yang dapat mencakup seluruh isi ruangan. Selanjutnya, pemodelan dan penyelesaian tersebut akan digunakan untuk merancang perangkat lunak yang dapat menerima masalah ini dan menyelesaikannya. Setiap kebutuhan perangkat lunak yang terdiri dari masukan, keluaran, dan kelas-kelas akan dibahas lebih lanjut pada bagian analisis kebutuhan perangkat lunak.

3.1 Pemodelan Masalah

Masalah yang dibahas pada penelitian ini perlu dimodelkan terlebih dahulu agar menjadi konkret. Pemodelan masalah terdiri dari pemodelan ruangan, kamera CCTV, dan cakupan kamera CCTV. Pemodelan yang dilakukan pada penelitian sebelumnya (2.3) akan diterapkan kembali pada penelitian ini dengan adanya modifikasi. Modifikasi dilakukan pada pemodelan ruangan dan pemodelan cakupan kamera CCTV. Sebelumnya, ruangan dimodelkan menggunakan *grid point* sehingga suatu bagian daerah dalam ruangan dinyatakan dalam bentuk titik. Sedangkan pada penelitian ini, suatu bagian dalam ruangan dinyatakan dalam bentuk *cell*. Modifikasi pemodelan ruangan turut disertai dengan modifikasi pemodelan daerah cakupan kamera CCTV. Sebelumnya, cakupan kamera CCTV terdiri dari kumpulan titik. Namun, karena suatu daerah dalam ruangan dinyatakan dalam bentuk *cell*, maka cakupan kamera CCTV pada penelitian ini berubah sehingga terdiri dari kumpulan *cell*.

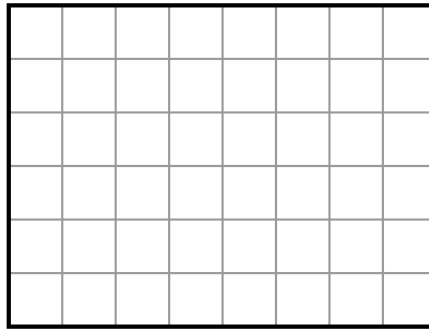
3.1.1 Ruangan

Bentuk ruangan pada masalah ini dibatasi sehingga berbentuk persegi panjang dalam bidang 2 dimensi. Karena berbentuk persegi panjang, maka ruangan terdiri dari ukuran panjang dan lebar seperti pada gambar 3.1. Kedua ukuran ini menggunakan satuan ukuran sentimeter (cm).



Gambar 3.1: Pemodelan ruangan

Ruangan akan dipecah menjadi matriks *cell* yang berukuran lebih kecil seperti pada gambar 3.2. Hal ini bertujuan agar daerah dalam ruangan dinyatakan dalam kumpulan daerah yang lebih kecil. Setiap bagian daerah yang lebih kecil tersebut disebut sebagai *cell*. *Cell* juga menyatakan bagian daerah terkecil yang tidak dapat dibagi lagi.

Gambar 3.2: Pemecahan ruangan menjadi matriks *cell*

Ukuran *cell* dapat ditentukan berdasarkan ukuran daerah terkecil. Namun, apabila *cell* diharuskan berbentuk persegi, maka terdapat kemungkinan di mana susunan *cell* tidak dapat membentuk ukuran ruangan yang utuh. Untuk menyiasati hal tersebut, *cell* dibuat dapat berbentuk persegi panjang dan ukurannya didapatkan berdasarkan suatu ukuran. Ukuran tersebut adalah ukuran terbesar *cell* yang menyatakan ukuran terbesar yang dapat dimiliki *cell* sehingga ukuran panjang dan lebarnya tidak melebihi ukuran ini. Terdapat perhitungan yang dilakukan untuk mendapatkan ukuran *cell* berdasarkan ukuran terbesar *cell*. Didefinisikan variabel sebagai berikut:

$$\begin{aligned}
 l &: \text{panjang ruangan} \\
 w &: \text{lebar ruangan} \\
 l_c &: \text{panjang cell} \\
 w_c &: \text{lebar cell} \\
 m &: \text{ukuran terbesar cell}
 \end{aligned} \tag{3.1}$$

Selanjutnya, akan dicari ukuran matriks berdasarkan ukuran ruangan dan ukuran terbesar *cell*:

$$\begin{aligned}
 &cols : \text{jumlah kolom pada matriks } cell \\
 &rows : \text{jumlah baris pada matriks } cell \\
 &cols = \text{ceil} \left(\frac{l}{m} \right) \\
 &rows = \text{ceil} \left(\frac{w}{m} \right)
 \end{aligned} \tag{3.2}$$

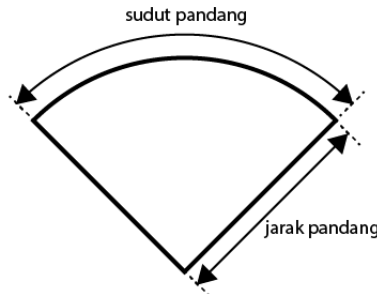
Setelah mendapatkan ukuran matriks, ukuran ruangan akan dibagi berdasarkan ukuran matriks:

$$\begin{aligned}
 l_c &= \frac{l}{cols} \\
 w_c &= \frac{w}{rows}
 \end{aligned} \tag{3.3}$$

Dengan demikian, didapatkan ukuran *cell* berdasarkan ukuran terbesar *cell*. Ukuran ini merupakan ukuran terbesar yang tidak melebihi ukuran terbesar *cell*.

3.1.2 Kamera CCTV

Kamera CCTV pada masalah ini dimodelkan dalam bentuk sebagian lingkaran pada bidang 2 dimensi. Kamera CCTV memiliki parameter spesifikasi yang beragam. Namun, pada masalah ini, hanya digunakan 2 parameter spesifikasi, yaitu jarak pandang dan besar sudut pandang seperti pada gambar 3.3. Kedua parameter ini menentukan daerah yang dapat dicakup oleh kamera CCTV. Jarak pandang dinyatakan dalam satuan ukuran sentimeter (cm) dan besar sudut pandang dinyatakan dalam satuan ukuran derajat ($^{\circ}$).



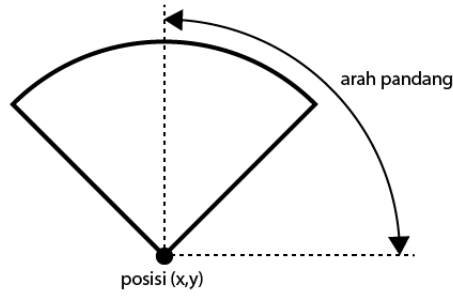
Gambar 3.3: Pemodelan kamera CCTV

Penempatan kamera CCTV dalam ruangan terdiri dari 2 bagian, yaitu posisi penempatan dan sudut arah pandang seperti pada gambar 3.4. Posisi penempatan kamera CCTV dinyatakan dalam titik koordinat kartesius 2 dimensi (x, y) . Sudut arah pandang menunjukkan arah yang dituju kamera CCTV. Arah pandang kamera CCTV dinyatakan dalam jarak sudut derajat ($^{\circ}$) antara arah yang dituju dengan garis 0° .

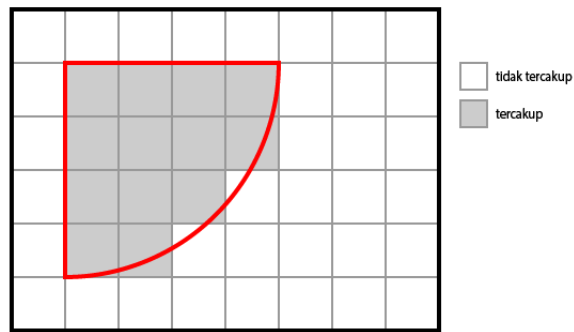
3.1.3 Cakupan Kamera CCTV

Dengan pemecahan ruangan ke dalam matriks *cell*, maka cakupan kamera CCTV terdiri dari kumpulan *cell*. Contoh cakupan kamera CCTV dapat dilihat pada gambar 3.5.

Setiap *cell* memiliki titik tengah yang berada di tengah *cell*. Titik tengah digunakan untuk menentukan ketercakupan *cell* oleh suatu penempatan kamera CCTV. Untuk menentukan apakah suatu *cell* dapat dicakup oleh suatu penempatan kamera CCTV, maka perlu dilakukan 2 jenis pengecekan. Pengecekan pertama dilakukan untuk memastikan bahwa jarak titik tengah *cell*



Gambar 3.4: Penempatan kamera CCTV dalam ruangan

Gambar 3.5: Cakupan kamera CCTV yang terdiri dari kumpulan *cell*

terhadap titik penempatan kamera CCTV lebih kecil atau sama dengan jarak pandang kamera CCTV. Untuk melakukannya, terlebih dahulu didefinisikan variabel sebagai berikut:

$$\begin{aligned}
 (x_{cam}, y_{cam}) &: \text{titik penempatan kamera CCTV} \\
 (x_{cell}, y_{cell}) &: \text{titik tengah } cell \\
 r &: \text{jarak pandang kamera CCTV}
 \end{aligned}
 \tag{3.4}$$

Pengecekan pertama dinyatakan berhasil apabila memenuhi pernyataan berikut:

$$\sqrt{(x_{cam} - x_{cell})^2 + (y_{cam} - y_{cell})^2} \leq r
 \tag{3.5}$$

Pengecekan kedua dilakukan untuk memastikan bahwa titik tengah *cell* berada di antara sudut

pandang kamera CCTV. Untuk melakukannya, didefinisikan variabel dan fungsi sebagai berikut:

$$\begin{aligned}
 &\alpha : \text{besar sudut pandang kamera CCTV} \\
 &\beta : \text{sudut arah pandang kamera CCTV} \\
 &\text{norm}(\theta) : \text{fungsi untuk menormalkan sudut sehingga } \theta \text{ berada dalam rentang } [0, 2\pi) \\
 &\text{atan2}(x, y) : \text{fungsi untuk mendapatkan sudut rotasi titik } (x, y) \text{ terhadap titik O} \\
 &\text{norm}(\theta) = \begin{cases} \text{norm}(\theta + 2\pi) & \text{jika } \theta < 0 \\ \text{norm}(\theta - 2\pi) & \text{jika } \theta \geq 2\pi \\ \theta & \text{jika } \theta \geq 0 \text{ dan } \theta < 2\pi \end{cases} \\
 &\text{atan2}(x, y) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{jika } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{jika } x < 0 \text{ dan } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{jika } x < 0 \text{ dan } y < 0 \\ +\frac{\pi}{2} & \text{jika } x = 0 \text{ dan } y > 0 \\ -\frac{\pi}{2} & \text{jika } x = 0 \text{ dan } y < 0 \\ \text{tak terdefinisi} & \text{jika } x = 0 \text{ dan } y = 0 \end{cases}
 \end{aligned} \tag{3.6}$$

Selanjutnya, akan dicari sudut pandang awal, sudut pandang akhir, dan sudut rotasi titik tengah *cell* terhadap titik penempatan kamera CCTV:

$$\begin{aligned}
 &\alpha_{half} : \text{setengah dari besar sudut pandang kamera CCTV} \\
 &\beta_{start} : \text{sudut pandang awal kamera CCTV} \\
 &\beta_{end} : \text{sudut pandang akhir kamera CCTV} \\
 &\beta_{cell} : \text{sudut rotasi titik tengah } cell \text{ terhadap titik penempatan kamera CCTV} \\
 &\alpha_{half} = \frac{\alpha}{2} \\
 &\beta_{start} = \text{norm}(\beta - \alpha_{half}) \\
 &\beta_{end} = \text{norm}(\beta + \alpha_{half}) \\
 &\beta_{cell} = \text{atan2}((y_{cell} - y_{cam}), (x_{cell} - x_{cam}))
 \end{aligned} \tag{3.7}$$

Pengecekan kedua dibedakan berdasarkan sudut pandang awal dan sudut pandang akhir. Apabila sudut pandang awal lebih kecil daripada sudut pandang akhir ($\beta_{start} < \beta_{end}$), maka kedua pernyataan berikut harus dipenuhi agar pengecekan kedua dinyatakan berhasil:

$$\begin{aligned}
 &\beta_{start} < \beta_{cell} \\
 &\beta_{cell} < \beta_{end}
 \end{aligned} \tag{3.8}$$

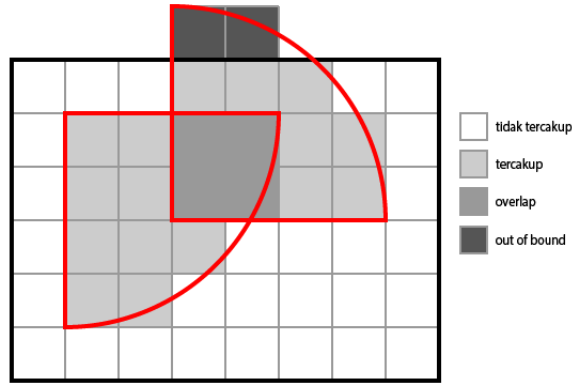
Apabila sudut pandang awal lebih besar atau sama dengan sudut pandang akhir ($\beta_{start} \geq \beta_{end}$), maka minimal satu dari kedua pernyataan berikut harus dipenuhi agar pengecekan kedua dinyatakan berhasil:

$$\begin{aligned}
 &\beta_{start} < \beta_{cell} \\
 &\beta_{cell} < \beta_{end}
 \end{aligned} \tag{3.9}$$

Dengan memenuhi pengecekan pertama dan kedua, maka *cell* dinyatakan tercakup oleh penempatan kamera CCTV.

Perhitungan tingkat *overlap* dan *out of bound* dapat dilakukan dengan membandingkan jumlah *cell*. *Overlap cell* adalah *cell* yang dicakup oleh lebih dari 1 kamera CCTV, sedangkan *out of bound cell* adalah *cell* di luar ruangan yang tercakup oleh kamera CCTV. Gambar 3.6 menggambarkan penempatan kamera CCTV yang menghasilkan *overlap cell* dan *out of bound cell*. Tingkat *overlap* dan *out of bound* dapat dihitung dengan membandingkan jumlah *overlap cell* dan *out of bound cell* dengan total *cell* dalam ruangan. Perhitungan tingkat *overlap* dan *out of bound* hanya dilakukan

apabila seluruh *cell* dalam ruangan telah tercakup oleh kamera CCTV.



Gambar 3.6: *Overlap cell* dan *out of bound cell*

3.2 Penyelesaian Masalah

Setelah memodelkan masalah, masalah akan dimodelkan ke dalam bentuk masalah *binary integer programming*. Solusi yang diharapkan dari masalah ini adalah penempatan-penempatan kamera CCTV yang berjumlah minimum yang dapat mencakup seluruh isi ruangan. Berdasarkan pemodelan masalah, ruangan telah dibagi ke dalam matriks *cell* sehingga solusi penempatan kamera CCTV harus mencakup seluruh *cell* dalam ruangan. Untuk mendapatkan solusi tersebut, akan dibangun seluruh kemungkinan penempatan kamera CCTV. Dari setiap kemungkinan tersebut, akan dipilih penempatan-penempatan kamera CCTV berjumlah minimum yang dapat mencakup seluruh *cell* dalam ruangan. Pemilihan dapat dilakukan dengan memeriksa seluruh kombinasi kemungkinan penempatan hingga didapatkan kombinasi yang sesuai dengan kriteria solusi. Namun, metode ini bukan merupakan metode efisien karena bersifat *exhaustive search* sehingga dibutuhkan metode lainnya yang dapat mencari kombinasi penempatan kamera CCTV tanpa melibatkan *exhaustive search*. Salah satu metodenya adalah memodelkan masalah ke dalam bentuk masalah *binary integer programming* dan menyelesaikannya menggunakan algoritma *Balas's additive*. Pada 2.2.1 telah dibahas bagaimana algoritma ini dapat menyelesaikan masalah *binary integer programming* secara efisien tanpa melibatkan *exhaustive search*. Dengan demikian, solusi dari masalah ini dapat ditemukan dengan memodelkan masalah ke dalam bentuk masalah *binary integer programming* dan menyelesaikannya menggunakan algoritma Balas's additive.

3.2.1 Variabel

Variabel dalam model masalah *binary integer programming* ini terdiri dari seluruh kemungkinan penempatan kamera CCTV. Penempatan kamera CCTV dapat dilakukan di setiap titik sudut pada setiap *cell* yang dinyatakan dalam himpunan s_x dan himpunan s_y berdasarkan sumbu x dan sumbu y . Seluruh kemungkinan sudut arah pandang penempatan kamera CCTV akan dibangun berdasarkan jumlah kemungkinan arah pandang (n) dan dinyatakan dalam himpunan s_φ sehingga $s_\varphi = \{\frac{m}{n} \times 2\pi | m = 1, 2, \dots, n\}$. Setelah membangun seluruh kemungkinan penempatan kamera CCTV, didefinisikan variabel biner sebagai berikut:

$$x_{ij\varphi} = \begin{cases} 1 & \text{jika kamera CCTV ditempatkan pada titik } (i, j) \text{ dengan} \\ & \text{arah pandang } \varphi \\ 0 & \text{jika sebaliknya} \end{cases} \quad (3.10)$$

3.2.2 Fungsi Tujuan

Fungsi tujuan dalam model masalah *binary integer programming* ini dinyatakan dalam bentuk minimasi dari seluruh kemungkinan penempatan kamera CCTV. Dengan fungsi tujuan ini, maka solusi dari model masalah *linear program* ini merepresentasikan penempatan kamera CCTV yang berjumlah minimum. Berikut ini merupakan fungsi tujuan dalam model masalah *binary integer programming* ini:

$$\min z = \sum_{\varphi=0}^{s_{\varphi}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{ij\varphi} \quad (3.11)$$

3.2.3 Batasan

Selain mendapatkan penempatan kamera CCTV yang berjumlah minimum, setiap *cell* dalam ruangan harus tercakup oleh kamera CCTV. Untuk memenuhinya, pada model masalah *linear program* akan ditambahkan batasan di mana setiap *cell* harus dicakup oleh setidaknya 1 penempatan kamera CCTV. Terdapat fungsi biner yang digunakan untuk menyatakan ketercakupannya *cell* oleh suatu penempatan kamera CCTV. Berikut ini merupakan fungsi biner tersebut:

$$\text{cov}(i, j, \varphi, p, q) = \begin{cases} 1 & \text{jika kamera CCTV ditempatkan pada titik } (i, j) \text{ dengan} \\ & \text{arah pandang } \varphi \text{ dapat mencakup } \textit{cell} \text{ pada baris } p \text{ kolom } q \\ 0 & \text{jika sebaliknya} \end{cases} \quad (3.12)$$

Fungsi biner tersebut digunakan menyatakan ketercakupannya *cell* oleh setiap kemungkinan penempatan kamera CCTV pada batasan. Berikut ini merupakan batasan pada model masalah *binary integer programming* ini:

$$\begin{aligned} \sum_{\varphi=0}^{s_{\varphi}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{i,j,\varphi} \times \text{cov}(i, j, \varphi, p, q) &\geq 1 \\ 0 \leq p \leq (s_p - 1), 0 \leq q \leq (s_q - 1) \end{aligned} \quad (3.13)$$

3.2.4 Model Masalah *Binary Integer Programming*

Variable pada model masalah *binary integer programming* ini terdiri dari seluruh kemungkinan penempatan kamera CCTV. Fungsi tujuan ditujukan untuk mendapatkan penempatan kamera CCTV yang berjumlah minimum. Agar seluruh isi ruangan dapat tercakup, maka ditambahkan batasan di mana setiap *cell* dalam ruangan harus dicakup oleh setidaknya 1 penempatan kamera CCTV. Dengan menggabungkan ketiganya, didapatkan model masalah *binary integer programming* sebagai berikut:

$$\begin{aligned} \min z &= \sum_{\varphi=0}^{s_{\varphi}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{ij\varphi} \\ \text{s.t. } &\sum_{\varphi=0}^{s_{\varphi}-1} \sum_{i=0}^{s_i-1} \sum_{j=0}^{s_j-1} x_{i,j,\varphi} \times \text{cov}(i, j, \varphi, p, q) \geq 1 \\ &0 \leq p \leq (s_p - 1), 0 \leq q \leq (s_q - 1) \\ &x_{ij\varphi} \in \{0, 1\} \end{aligned} \quad (3.14)$$

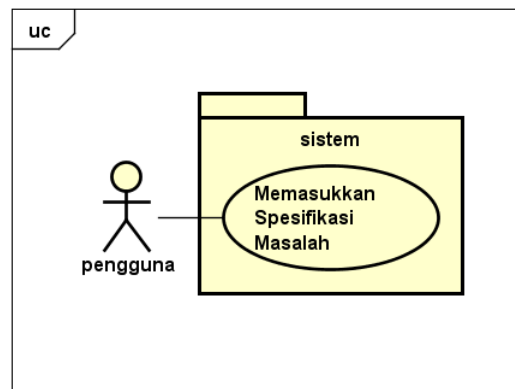
Selanjutnya, model ini akan diselesaikan menggunakan algoritma *Balas's additive* sehingga didapatkan solusi berupa kombinasi penempatan kamera CCTV yang berjumlah minimum yang dapat mencakup seluruh isi ruangan.

3.3 Analisis Kebutuhan Perangkat Lunak

Pada bagian ini, akan dibahas mengenai kebutuhan dari perangkat lunak yang akan dibangun. Terdapat diagram *use case* dan skenario untuk menjelaskan aksi yang dapat dilakukan pengguna terhadap perangkat lunak. Perangkat lunak yang dibangun dapat menerima masukan-masukan masalah dan menghasilkan keluaran yang sesuai dengan masukan tersebut. Terdapat juga diagram kelas sederhana yang digunakan untuk membangun perangkat lunak. Diagram kelas sederhana ini akan dikembangkan lebih lanjut pada tahap perancangan.

3.3.1 Diagram *Use Case* dan Skenario

Pada perangkat lunak yang dibangun, hanya terdapat 1 buah aktor, yaitu pengguna. Diagram *use case* pada gambar 3.7 menunjukkan aktor beserta dengan aksi yang dapat dilakukannya.



Gambar 3.7: Diagram *use case*

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

- Skenario: **Memasukkan Spesifikasi Masalah**

- Aktor: Pengguna
- Langkah:
 1. Aktor memasukkan spesifikasi masalah pada kolom masukan yang telah disediakan dan dilanjutkan dengan menekan tombol "submit".
 2. Sistem menampilkan tampilan penempatan kamera CCTV.

3.3.2 Kebutuhan Masukan Perangkat Lunak

Berdasarkan pemodelan masalah dan penyelesaian masalah menggunakan *binary integer programming*, ditetapkan masukan untuk perangkat lunak sebagai berikut:

- Panjang ruangan dalam satuan ukuran sentimeter (cm).
- Lebar ruangan dalam satuan ukuran sentimeter (cm).
- Jarak pandang kamera CCTV dalam satuan ukuran sentimeter (cm).
- Besar sudut pandang kamera CCTV dalam satuan sudut derajat (°).
- Ukuran terbesar *cell* dalam satuan ukuran sentimeter (cm).
- Jumlah kemungkinan sudut pandang penempatan kamera CCTV.

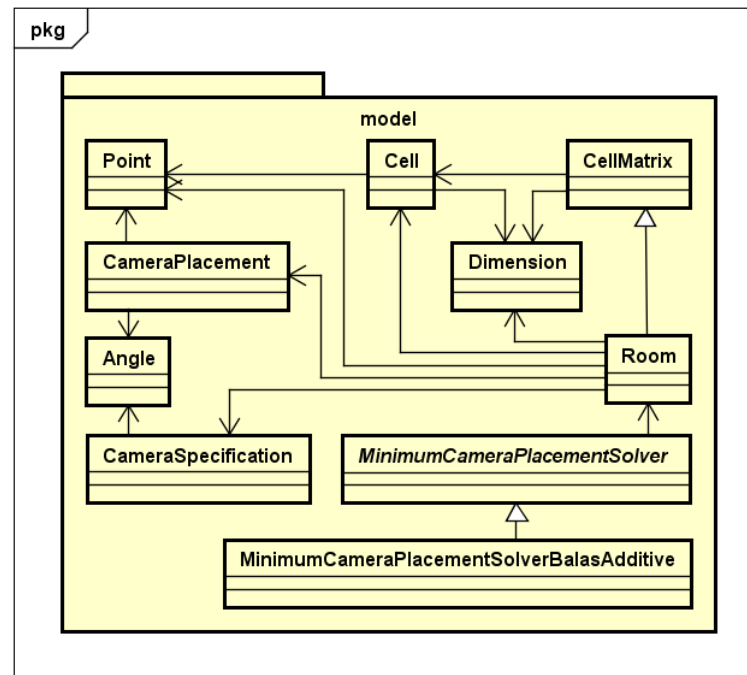
3.3.3 Kebutuhan Keluaran Perangkat Lunak

Perangkat lunak akan mencari solusi masalah berdasarkan masukan yang telah diberikan pengguna dan menghasilkan keluaran berupa:

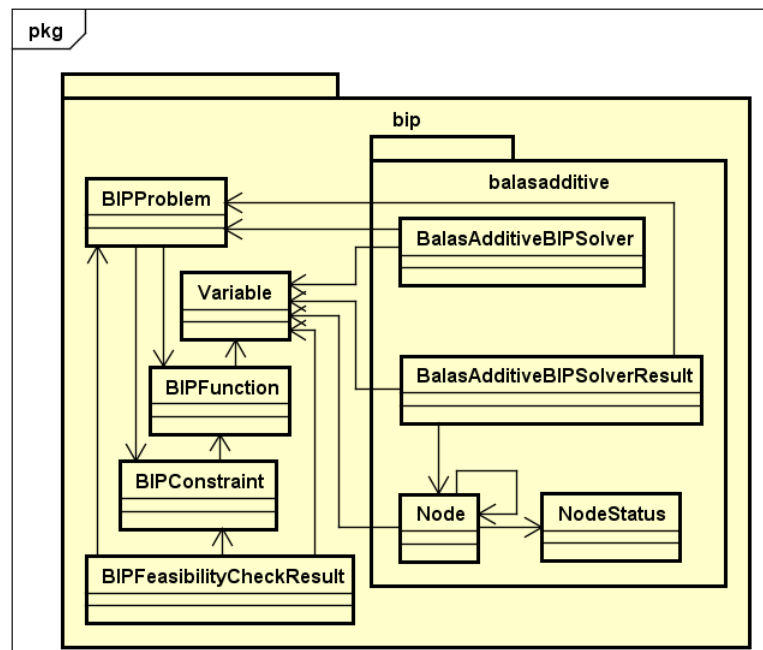
- Posisi dan sudut arah pandang dari setiap penempatan kamera CCTV yang berjumlah minimum yang dapat mencakup seluruh isi ruangan.

3.3.4 Diagram Kelas Sederhana

Pada bagian ini terdapat diagram kelas sederhana yang menunjukkan rancangan awal kelas-kelas yang akan digunakan untuk membangun perangkat lunak. Kelas-kelas tersebut dikelompokkan ke dalam 2 *package*. *Package* pertama adalah *package* model yang berisi kelas-kelas untuk memodelkan masalah. *Package* kedua adalah *package* bip yang berisi kelas-kelas untuk menyelesaikan masalah *binary integer programming*. Di dalam *package* bip terdapat *subpackage* *balasadditive* yang berisi kelas-kelas untuk menyelesaikan masalah *binary integer programming* menggunakan algoritma *Balas's additive*. Diagram kelas sederhana untuk kedua *package* tersebut dapat dilihat pada gambar 3.8 dan 3.9.



Gambar 3.8: Diagram kelas sederhana untuk *package* model



Gambar 3.9: Diagram kelas sederhana untuk *package* `bip` dan *subpackage* `balasadditive`

BAB 4

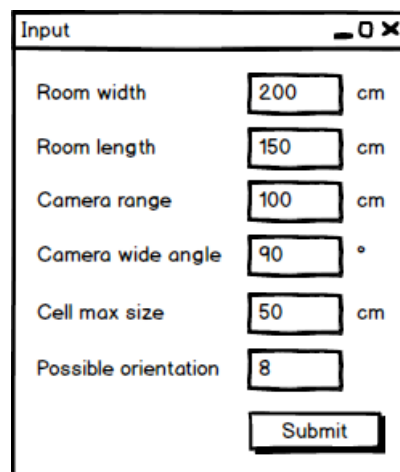
PERANCANGAN

4.1 Perancangan Antarmuka

Perangkat lunak yang dibangun akan menggunakan tampilan antarmuka grafis. Tampilan antarmuka ini digunakan agar mempermudah interaksi pengguna dengan perangkat lunak. Selain itu, jenis tampilan ini juga digunakan untuk menghasilkan visualisasi penempatan kamera CCTV sehingga pengguna dapat memahami penempatan-penempatan tersebut secara visual. Pada bagian ini akan dijelaskan bentuk dari setiap antarmuka yang terdapat dalam perangkat lunak. Berikut bentuk antarmuka tersebut:

- Antarmuka: **Penerima Masukan**

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



Input	
Room width	<input type="text" value="200"/> cm
Room length	<input type="text" value="150"/> cm
Camera range	<input type="text" value="100"/> cm
Camera wide angle	<input type="text" value="90"/> °
Cell max size	<input type="text" value="50"/> cm
Possible orientation	<input type="text" value="8"/>
<input type="button" value="Submit"/>	

Gambar 4.1: Perancangan antarmuka penerima masukan

- Antarmuka: **Penempatan Kamera CCTV**

Antarmuka ini berfungsi untuk menampilkan solusi masalah penempatan kamera CCTV yang berjumlah minimum yang dapat mencakup seluruh isi ruangan seperti pada gambar 4.2. Di dalam antarmuka ini terdapat 2 bagian, yaitu:

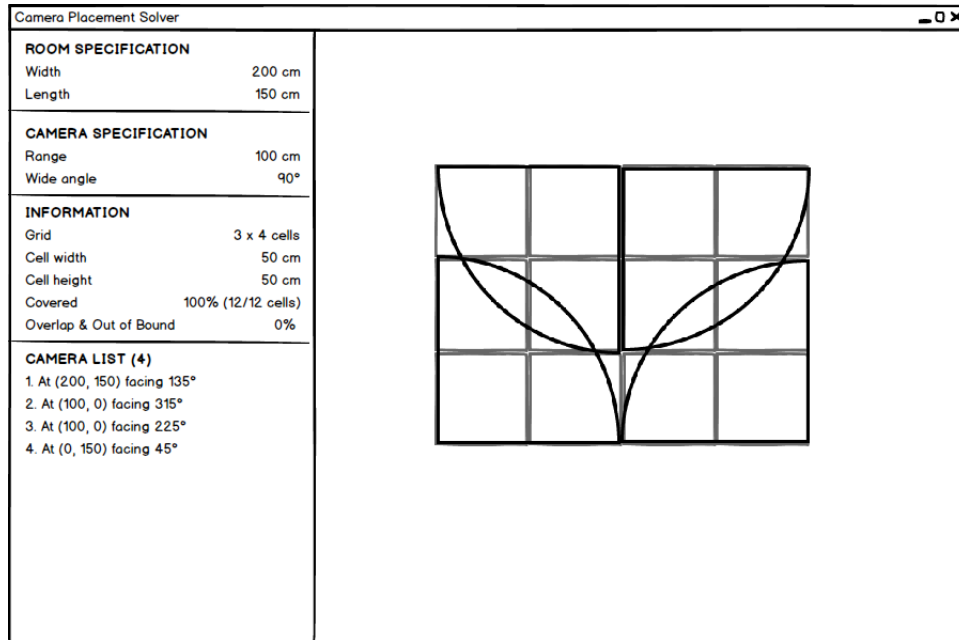
- Panel informasi

Panel ini berada di bagian kiri antarmuka yang berfungsi untuk memberikan informasi mengenai spesifikasi masalah dan solusi masalah. Solusi masalah yang terdiri dari

penempatan-penempatan kamera CCTV dapat dilihat pada bagian daftar kamera CCTV (*Camera List*). Pada setiap baris penempatan terdapat informasi titik lokasi penempatan dan juga sudut arah pandang yang dituju.

- Panel visualisasi penempatan kamera CCTV

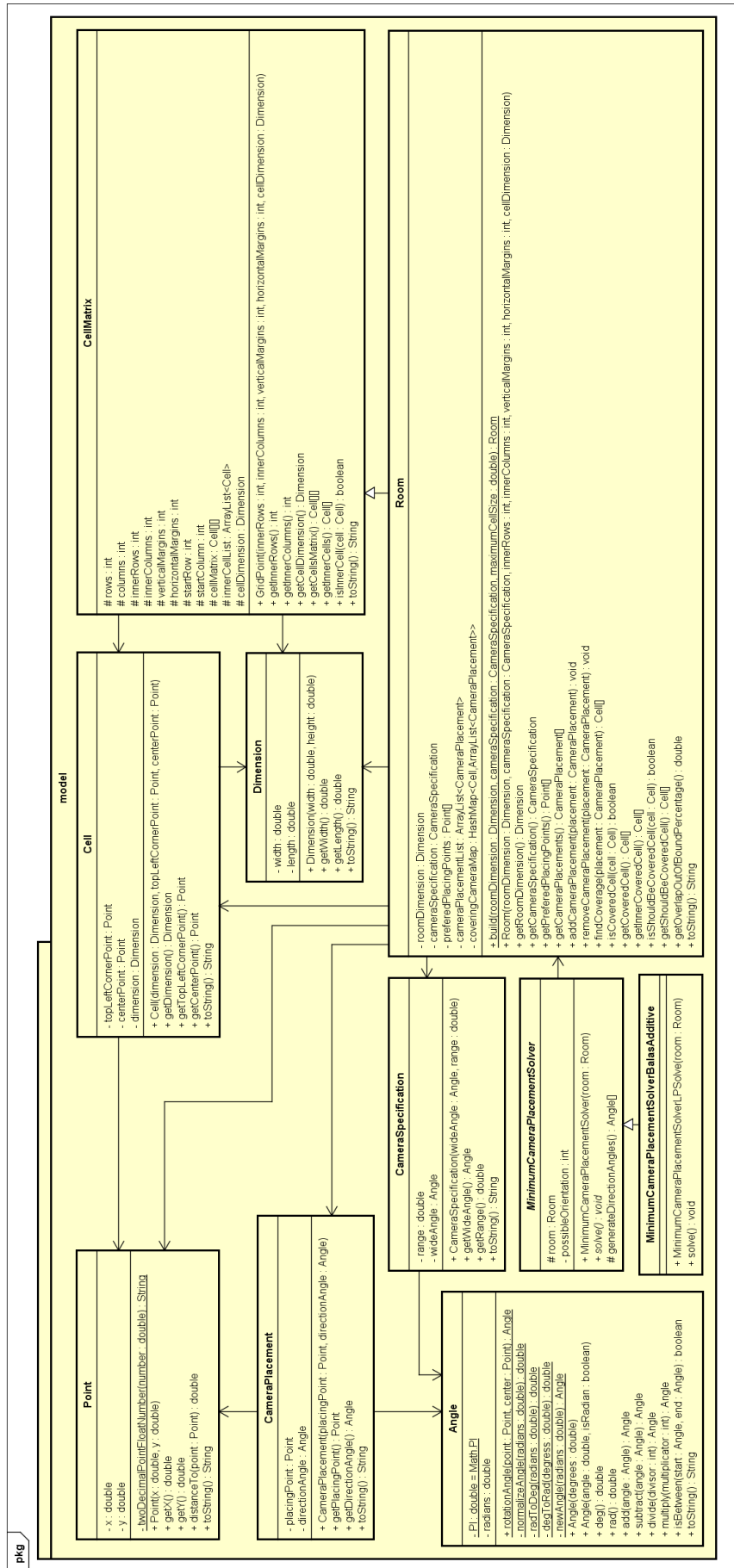
Panel ini berada di bagian kanan antarmuka yang berfungsi untuk menampilkan visualisasi matriks *cell* dan penempatan kamera CCTV.

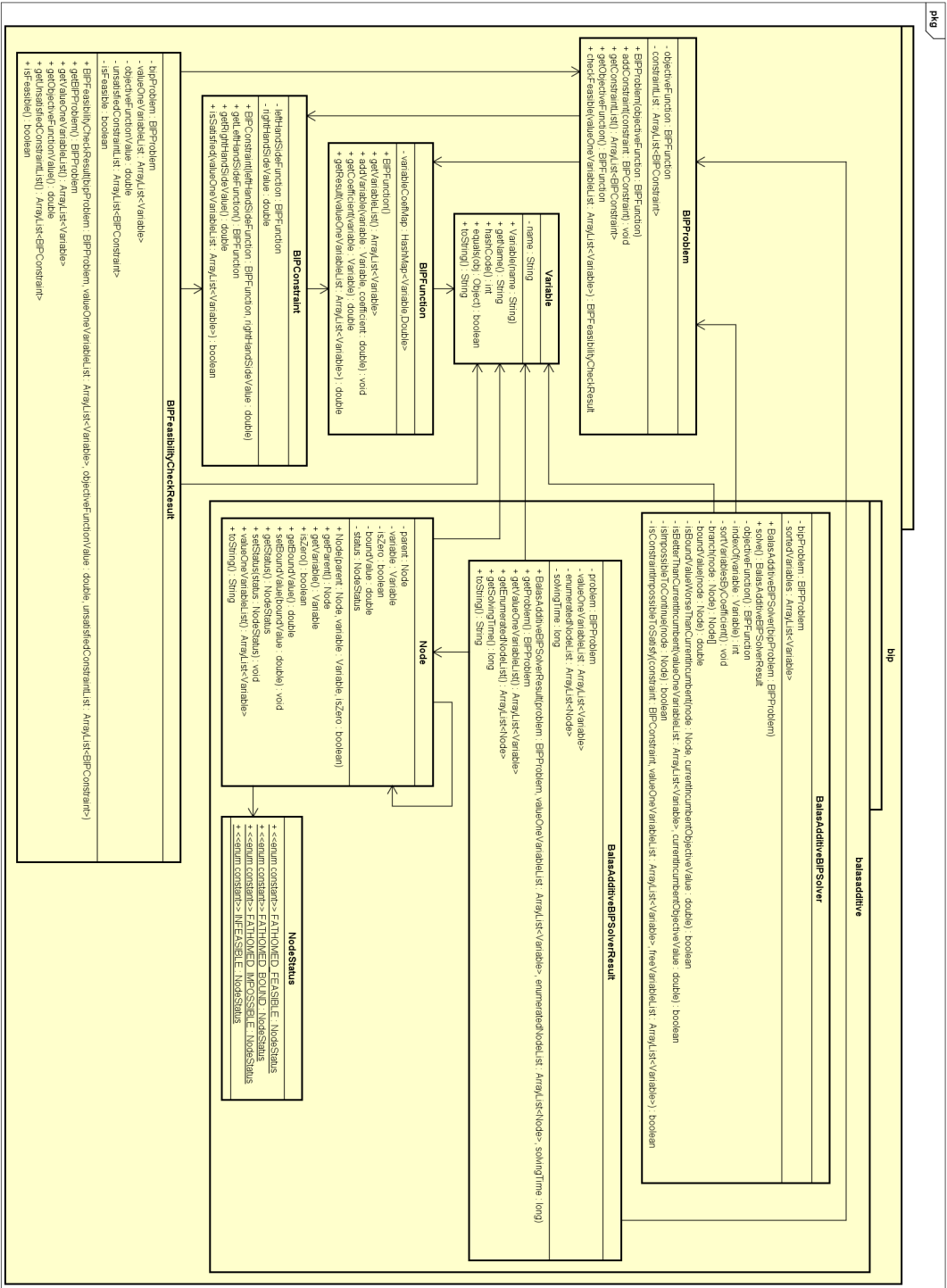


Gambar 4.2: Perancangan antarmuka penempatan kamera CCTV

4.2 Perancangan Kelas

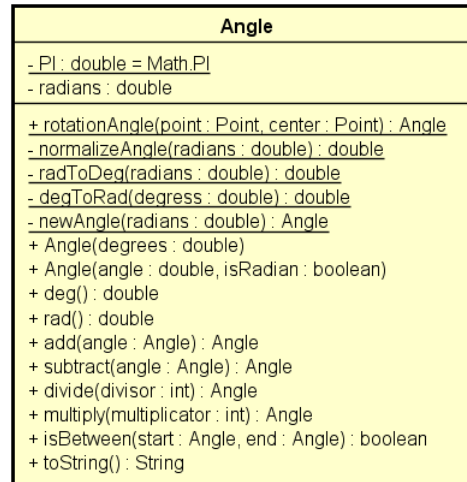
Rancangan awal kelas-kelas yang telah dipaparkan di 3.3.4 akan dikembangkan lebih lanjut pada bagian ini. Kelas-kelas yang digunakan untuk memodelkan masalah diletakkan dalam *package* model. Kelas-kelas untuk memodelkan dan menyelesaikan masalah *binary integer programming* diletakkan pada *package* bip. Diagram kelas rinci untuk kedua *package* tersebut dapat dilihat pada gambar 4.3 dan 4.4.

Gambar 4.3: Diagram kelas rinci untuk *package* model



Gambar 4.4: Diagram kelas rinci untuk *package* bip dan *subpackage* balasadditive

4.2.1 Kelas *Angle*



Gambar 4.5: Diagram kelas *Angle*

Kelas ini merepresentasikan sudut dan menangani fungsi-fungsi yang berhubungan dengan sudut. Diagram kelas *Angle* dapat dilihat pada gambar 4.5. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *Angle*:

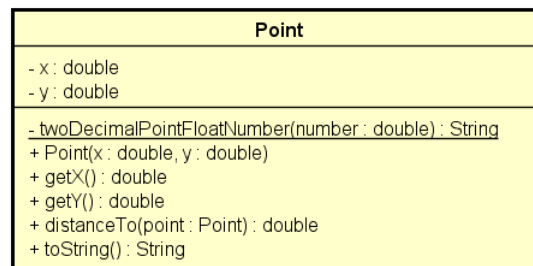
- ***PI* → double**
Atribut ini merupakan atribut statis bernilai π yang dapat digunakan oleh setiap objek dari kelas *Angle*.
- ***radians* → double**
Atribut ini berguna untuk menampung sudut dalam bentuk radian.

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 pengalihan 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 4.6: Diagram kelas *Point*

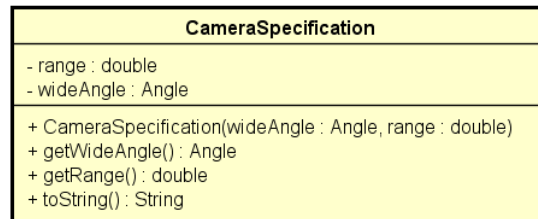
Kelas ini merepresentasikan titik koordinat 2D. Diagram kelas *Point* dapat dilihat pada gambar 4.6. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *Point*:

- ***x* → *double***
Atribut ini berguna untuk menampung nilai titik pada sumbu x.
- ***y* → *double***
Atribut ini berguna untuk menampung nilai titik pada sumbu y.

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

- ***twoDecimalPointFloatNumber(number → double) → 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.
- ***distanceTo(point → Point) → double***
Fungsi ini berguna untuk mendapatkan jarak antara titik objek ini dengan titik objek *point*.

4.2.3 Kelas *CameraSpecification*

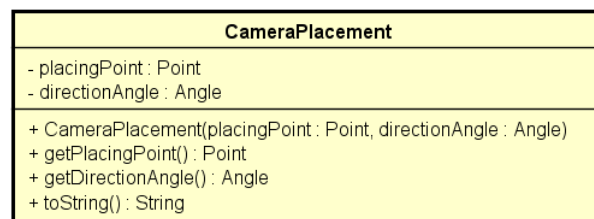


Gambar 4.7: Diagram kelas *CameraSpecification*

Kelas ini merepresentasikan spesifikasi kamera CCTV yang terdiri dari jarak pandang dan besar sudut pandang. Diagram kelas *CameraSpecification* dapat dilihat pada gambar 4.7. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *CameraSpecification*:

- ***range* → *double***
Atribut ini berguna untuk menampung jarak pandang kamera CCTV.
- ***wideAngle* → *Angle***
Atribut ini berguna untuk menampung besar sudut pandang kamera CCTV.

4.2.4 Kelas *CameraPlacement*

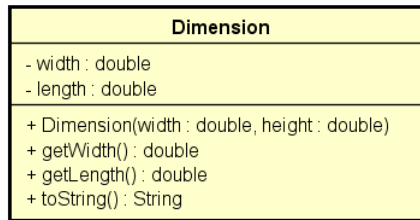


Gambar 4.8: Diagram kelas *CameraPlacement*

Kelas ini merepresentasikan penempatan kamera CCTV yang terdiri dari posisi dan sudut arah pandang. Diagram kelas *CameraPlacement* dapat dilihat pada gambar 4.8. 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*

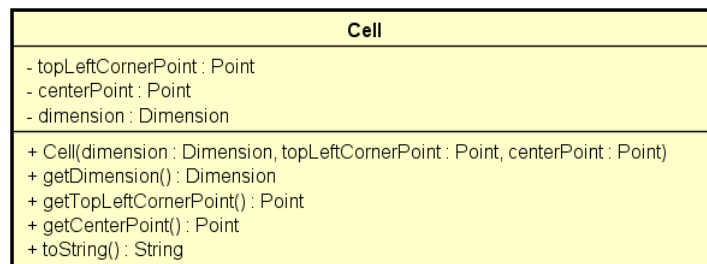


Gambar 4.9: Diagram kelas *Dimension*

Kelas ini merepresentasikan dimensi yang terdiri dari panjang dan lebar. Diagram kelas *Dimension* dapat dilihat pada gambar 4.9. 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*

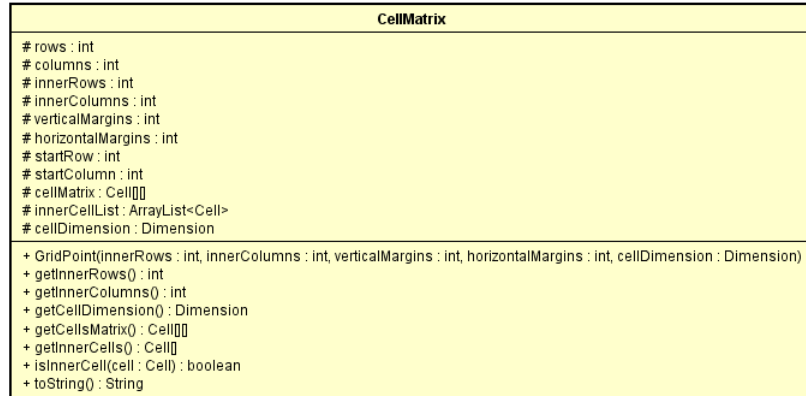


Gambar 4.10: Diagram kelas *Cell*

Kelas ini merepresentasikan cell. Diagram kelas *Cell* dapat dilihat pada gambar 4.10. 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 *CellMatrix*

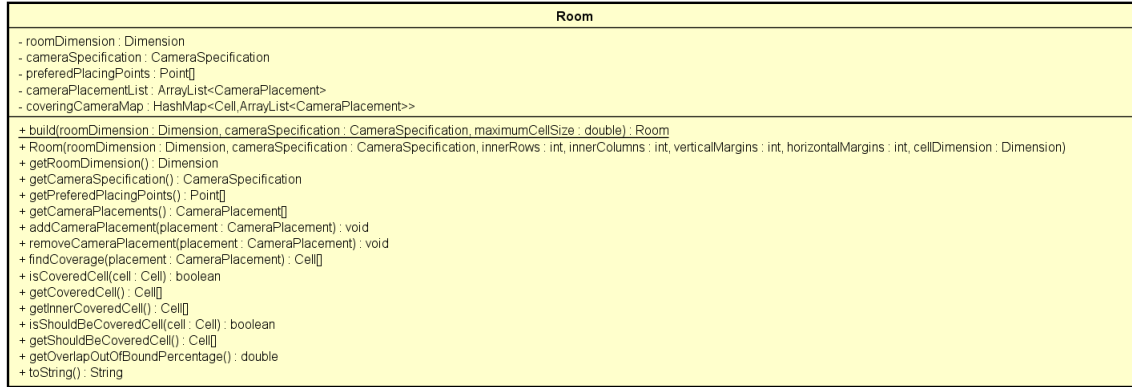


Gambar 4.11: Diagram kelas *CellMatrix*

Kelas ini merepresentasikan matriks *cell* dalam ruangan. Diagram kelas *MatrixCell* dapat dilihat pada gambar 4.11. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *CellMatrix*:

- ***rows* → *int***
Atribut ini berguna untuk menampung jumlah baris matriks *cell* secara keseluruhan.
- ***columns* → *int***
Atribut ini berguna untuk menampung jumlah kolom matriks *cell* secara keseluruhan.
- ***innerRows* → *int***
Atribut ini berguna untuk menampung jumlah baris matriks *cell* bagian dalam.
- ***innerColumns* → *int***
Atribut ini berguna untuk menampung jumlah kolom matriks *cell* bagian dalam.
- ***verticalMargins* → *int***
Atribut ini berguna untuk menampung jumlah margin vertikal matriks *cell*.
- ***horizontalMargins* → *int***
Atribut ini berguna untuk menampung jumlah margin horizontal matriks *cell*.
- ***startRow* → *int***
Atribut ini berguna untuk menampung indeks baris pertama matriks *cell* bagian dalam.
- ***startColumn* → *int***
Atribut ini berguna untuk menampung indeks kolom pertama matriks *cell* bagian dalam.
- ***cellMatrix* → *Cell*[][]**
Atribut ini berguna untuk menampung matriks *cell* 2 dimensi.
- ***innerCellList* → *ArrayList*<*Cell*>**
Atribut ini berguna untuk menampung seluruh *cell* pada matriks *cell* bagian dalam.
- ***cellDimension* → *Dimension***
Atribut ini berguna untuk menampung dimensi *cell*.

4.2.8 Kelas *Room*



Gambar 4.12: Diagram kelas *Room*

Kelas ini merepresentasikan ruangan yang dapat diisi oleh kamera-kamera CCTV. Kelas ini merupakan turunan dari kelas *CellMatrix*. Diagram kelas *Room* dapat dilihat pada gambar 4.12. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *Room*:

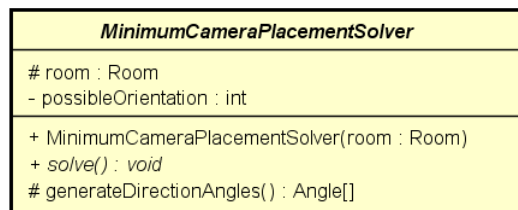
- ***roomDimension* → *Dimension***
Atribut ini berguna untuk menampung dimensi ruangan.
- ***cameraSpecification* → *CameraSpecification***
Atribut ini berguna untuk menampung spesifikasi dari kamera CCTV yang akan ditempatkan dalam ruangan.
- ***preferredPlacingPoints* → *Point*[]**
Atribut ini berguna untuk menampung posisi-posisi yang dapat ditempati oleh kamera CCTV.
- ***cameraPlacementList* → *ArrayList*<*CameraPlacement*>**
Atribut ini berguna untuk menampung daftar penempatan kamera CCTV.
- ***coveringCameraMap* → *HashMap*<*Cell*, *ArrayList*<*CameraPlacement*>>**
Atribut ini berguna untuk menampung pemetaan *cell* dengan penempatan kamera CCTV yang dapat mencakup *cell* tersebut.

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

- ***build*(*roomDimension* → *Dimension*, *cameraSpecification* → *CameraSpecification*, *maximumCellSize* → *double*) → *Room***
Fungsi ini merupakan fungsi statis yang berguna untuk membuat objek *Room* yang dimana jumlah baris, jumlah kolom, jumlah margin vertikal, jumlah margin horizontal, dan ukuran cell akan ditentukan berdasarkan ukuran ruangan *roomDimension*, spesifikasi kamera *cameraSpecification*, dan ukuran terbesar cell *maximumCellSize*.
- ***addCameraPlacement*(*placement* → *CameraPlacement*) → *void***
Fungsi ini berguna untuk menambahkan penempatan kamera CCTV ke dalam daftar penempatan kamera CCTV dan memperbaharui pemetaan cell pada atribut *coveringCameraMap*.
- ***removeCameraPlacement*(*placement* → *CameraPlacement*) → *void***
Fungsi ini berguna untuk membuang penempatan kamera CCTV dari daftar penempatan kamera CCTV dan memperbaharui pemetaan cell pada atribut *coveringCameraMap*.

- ***findCoverage(placement → CameraPlacement) → Cell[]***
Fungsi ini berguna untuk mendapatkan *cell* yang tercakup oleh penempatan kamera CCTV *placement*.
- ***isCoveredCell(cell → Cell) → boolean***
Fungsi ini berguna untuk mengetahui apakah *cell* telah tercakup oleh setidaknya 1 penempatan kamera CCTV.
- ***getCoveredCell() → Cell[]***
Fungsi ini berguna untuk mendapatkan *cell* yang telah tercakup oleh setidaknya 1 penempatan kamera CCTV.
- ***getInnerCoveredCell() → Cell[]***
Fungsi ini berguna untuk mendapatkan *cell* yang berada pada matriks *cell* bagian dalam yang telah tercakup oleh setidaknya 1 penempatan kamera CCTV.
- ***isShouldBeCoveredCell(cell → Cell) → boolean***
Fungsi ini berguna untuk mengetahui apakah *cell* berada pada matriks *cell* bagian dalam dan belum tercakup oleh setidaknya 1 penempatan kamera CCTV.
- ***getShouldBeCoveredCell() → Cell[]***
Fungsi ini berguna untuk mendapatkan *cell* yang berada pada matriks *cell* bagian dalam dan belum tercakup oleh setidaknya 1 penempatan kamera CCTV.
- ***getOverlapAndOutOfBoundPercentage() → double***
Fungsi ini berguna untuk mendapatkan persentase *overlap* dan *out of bound*.

4.2.9 Kelas *MinimumCameraPlacementSolver*



Gambar 4.13: Diagram kelas *MinimumCameraPlacementSolver*

Kelas ini merepresentasikan pemecah masalah penempatan kamera CCTV dalam ruangan yang berjumlah minimum yang dapat mencakup seluruh isi ruangan. Kelas ini merupakan kelas abstrak. Diagram kelas *MinimumCameraPlacementSolver* dapat dilihat pada gambar 4.13. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *MinimumCameraPlacementSolver*:

- ***room → Room***
Atribut ini berguna untuk menampung ruangan yang di mana masalah penempatan kamera CCTV-nya akan diselesaikan.
- ***possibleOrientation → int***
Atribut ini berguna untuk menampung jumlah kemungkinan sudut arah pandang kamera CCTV.

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

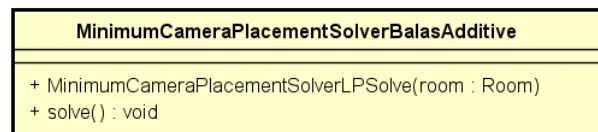
- ***solve()* → void**

Fungsi ini merupakan fungsi abstrak yang bertujuan untuk menyelesaikan masalah penempatan kamera CCTV dalam ruangan yang berjumlah minimum yang dapat mencakup seluruh isi ruangan.

- ***generateDirectionAngles()* → *Angle*[]**

Fungsi ini berguna untuk menghasilkan sudut-sudut arah pandang berdasarkan jumlah kemungkinan sudut arah pandang *possibleOrientation*.

4.2.10 Kelas *MinimumCameraPlacementSolverBalasAdditive*



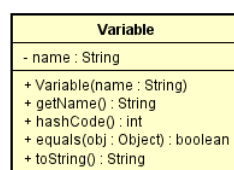
Gambar 4.14: Diagram kelas *MinimumCameraPlacementSolverBalasAdditive*

Kelas ini merepresentasikan pemecah masalah penempatan kamera CCTV dalam ruangan yang berjumlah minimum yang dapat mencakup seluruh isi ruangan dengan menggunakan algoritma *Balas's additive* (2.2.1). Kelas ini merupakan turunan dari kelas *MinimumCameraPlacementSolver*. Diagram kelas *MinimumCameraPlacementSolverBalasAdditive* dapat dilihat pada gambar 4.14. Berikut ini merupakan fungsi-fungsi yang terdapat pada kelas *MinimumCameraPlacementSolverBalasAdditive*:

- ***solve()* → void**

Fungsi ini berguna untuk menyelesaikan masalah penempatan kamera CCTV dalam ruangan yang berjumlah minimum yang dapat mencakup seluruh isi ruangan menggunakan algoritma *Balas's additive*.

4.2.11 Kelas *Variable*



Gambar 4.15: Diagram kelas *Variable*

Kelas ini merepresentasikan variabel. Diagram kelas *Variable* dapat dilihat pada gambar 4.15. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *Variable*:

- ***name* → *String***

Atribut ini berguna untuk menampung nama variabel.

4.2.12 Kelas *BIPFunction*

BIPFunction
- <code>variableCoefMap : HashMap<Variable, Double></code>
+ <code>BIPFunction()</code>
+ <code>getVariableList() : ArrayList<Variable></code>
+ <code>addVariable(variable : Variable, coefficient : double) : void</code>
+ <code>getCoefficient(variable : Variable) : double</code>
+ <code>getResult(valueOneVariableList : ArrayList<Variable>) : double</code>

Gambar 4.16: Diagram kelas *BIPFunction*

Kelas ini merepresentasikan persamaan yang terdiri dari variabel biner. Diagram kelas *BIPFunction* dapat dilihat pada gambar 4.16. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *BIPFunction*:

- ***variableCoefMap* → *HashMap<Variable, Double>***
Atribut ini berguna untuk menampung koefisien dari setiap variabel.

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

- ***getVariableList()* → *ArrayList<Variable>***
Fungsi ini berguna untuk mendapatkan daftar variabel yang terdapat dalam persamaan.
- ***addVariable(variable → Variable, coefficient → double) → void***
Fungsi ini berguna untuk menambahkan variabel beserta koefisiennya ke dalam persamaan.
- ***getCoefficient(variable → Variable) → double***
Fungsi ini berguna untuk mendapatkan koefisien *variable*.
- ***getResult(valueOneVariableList → ArrayList<Variable>) → double***
Fungsi ini berguna untuk mendapatkan hasil persamaan berdasarkan daftar variabel yang bernilai 1.

4.2.13 Kelas *BIPConstraint*

BIPConstraint
- <code>leftHandSideFunction : BIPFunction</code>
- <code>rightHandSideValue : double</code>
+ <code>BIPConstraint(leftHandSideFunction : BIPFunction, rightHandSideValue : double)</code>
+ <code>getLeftHandSideFunction() : BIPFunction</code>
+ <code>getRightHandSideValue() : double</code>
+ <code>isSatisfied(valueOneVariableList : ArrayList<Variable>) : boolean</code>

Gambar 4.17: Diagram kelas *BIPConstraint*

Kelas ini merepresentasikan batasan dalam masalah *binary integer programming*. Diagram kelas *BIPConstraint* dapat dilihat pada gambar 4.17. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *BIPConstraint*:

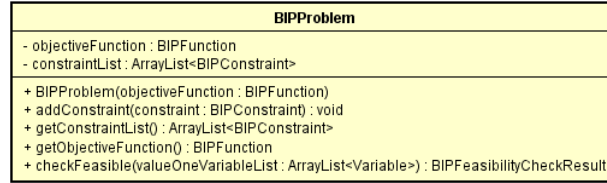
- ***leftHandSideFunction* → *BIPFunction***
Atribut ini berguna untuk menampung persamaan pada ruas kiri batasan.
- ***rightHandSideValue* → *double***
Atribut ini berguna untuk menampung nilai pada ruas kanan batasan.

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

- ***isSatisfied(valueOneVariableList → ArrayList<Variable>) → boolean***

Fungsi ini berguna untuk mengetahui apakah batasan dapat dipenuhi berdasarkan daftar variabel yang bernilai 1.

4.2.14 Kelas *BIPProblem*



Gambar 4.18: Diagram kelas *BIPProblem*

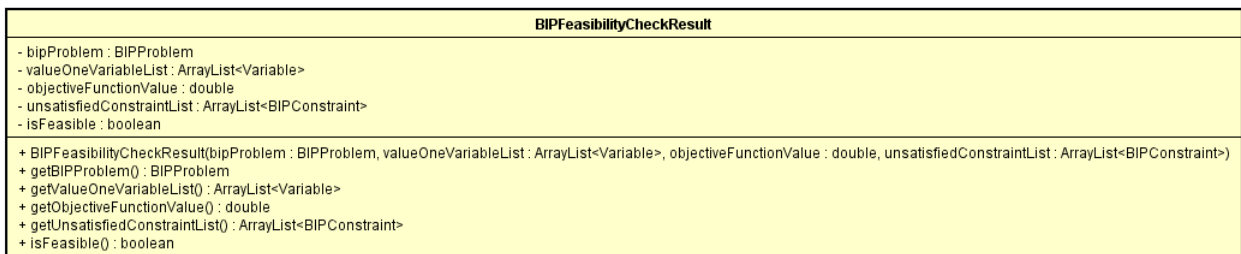
Kelas ini merepresentasikan model masalah *binary integer programming*. Diagram kelas *BIPProblem* dapat dilihat pada gambar 4.18. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *BIPProblem*:

- ***objectiveFunction → BIPFunction***
Atribut ini berguna untuk menampung persamaan fungsi tujuan.
- ***constraintList → ArrayList<BIPConstraint>***
Atribut ini berguna untuk menampung batasan-batasan.

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

- ***addConstraint(constraint → BIPConstraint) → void***
Fungsi ini berguna untuk menambahkan batasan pada model masalah *binary integer programming*.
- ***checkFeasible(valueOneVariableList → ArrayList<Variable>) → BIPFeasibilityCheckResult***
Fungsi ini berguna untuk mengembalikan hasil pengecekan solusi *feasible* berdasarkan daftar variabel bernilai 1.

4.2.15 Kelas *BIPFeasibilityCheckResult*

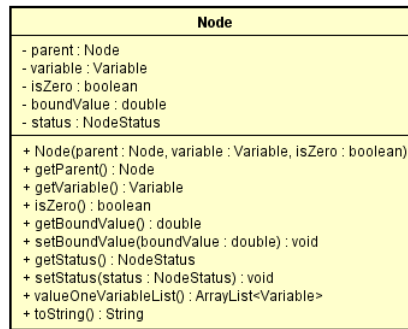


Gambar 4.19: Diagram kelas *BIPFeasibilityCheckResult*

Kelas ini merepresentasikan hasil pengecekan solusi *feasible* pada masalah *binary integer programming*. Diagram kelas *BIPFeasibilityCheckResult* dapat dilihat pada gambar 4.19. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *BIPFeasibilityCheckResult*:

- ***blpProblem* → *BLPProblem***
Atribut ini berguna untuk menampung model masalah *binary integer programming*.
- ***valueOneVariableList* → *ArrayList<Variable>***
Atribut ini berguna untuk menampung solusi berupa daftar variabel yang bernilai 1.
- ***objectiveFunctionValue* → *double***
Atribut ini berfungsi untuk menampung nilai hasil fungsi tujuan.
- ***unsatisfiedConstraintList* → *ArrayList<BIPConstraint>***
Atribut ini berfungsi untuk menampung batasan-batasan yang tidak dipenuhi.
- ***isFeasible* → *boolean***
Atribut ini berfungsi untuk menampung apakah solusi bersifat *feasible*.

4.2.16 Kelas *Node*



Gambar 4.20: Diagram kelas *Node*

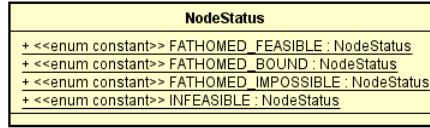
Kelas ini merepresentasikan *node* yang digunakan dalam algoritma *Balas's additive*. Diagram kelas *Node* dapat dilihat pada gambar 4.20. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *Node*:

- ***parent* → *Node***
Atribut ini berguna untuk menampung *node* orang tua dari *node* ini.
- ***variable* → *Variable***
Atribut ini berguna untuk menampung variabel yang dituju oleh *node* ini.
- ***isZero* → *boolean***
Atribut ini berguna untuk menampung apakah variabel pada *node* ini bernilai 1.
- ***boundValue* → *double***
Atribut ini berguna untuk menampung nilai *bound* dari *node* ini.
- ***status* → *NodeStatus***
Atribut ini berguna untuk menampung status dari *node* ini.

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

- ***valueOneVariableList()* → *ArrayList<Variable>***
Fungsi ini berguna untuk menghasilkan daftar variabel bernilai 1 yang dimulai dari *node* ini hingga *root node*.

4.2.17 Kelas *NodeStatus*

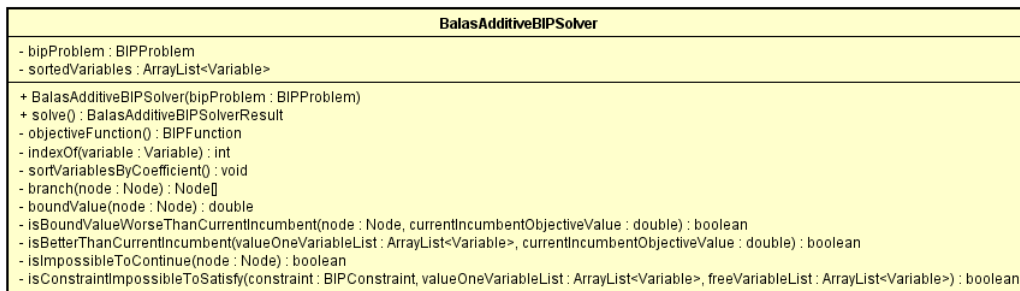


Gambar 4.21: Diagram kelas *NodeStatus*

Kelas ini merepresentasikan status yang dapat dimiliki *node*. Diagram kelas *NodeStatus* dapat dilihat pada gambar 4.21. Berikut ini merupakan pilihan-pilihan status yang terdapat pada kelas *NodeStatus*:

- ***FATHOMED_FEASIBLE* → *NodeStatus***
Pilihan ini menyatakan status *fathomed* dengan alasan solusi *feasible*.
- ***FATHOMED_BOUND* → *NodeStatus***
Pilihan ini menyatakan status *fathomed* dengan alasan nilai *bound* yang lebih buruk daripada solusi *incumbent*.
- ***FATHOMED_IMPOSSIBLE* → *NodeStatus***
Pilihan ini menyatakan status *fathomed* dengan alasan *impossible* atau tidak dapat menghasilkan *incumbent*.
- ***INFEASIBLE* → *NodeStatus***
Pilihan ini menyatakan status *infeasible*.

4.2.18 Kelas *BalasAdditiveBIPSolver*



Gambar 4.22: Diagram kelas *BalasAdditiveBIPSolver*

Kelas ini merepresentasikan metode penyelesaian masalah *binary integer programming* menggunakan algoritma *Balas's additive*. Diagram kelas *BalasAdditiveBIPSolver* dapat dilihat pada gambar 4.22. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *BalasAdditiveBIPSolver*:

- ***bipProblem* → *BIPProblem***
Atribut ini berguna untuk menampung model masalah *binary integer programming* yang akan diselesaikan.
- ***sortedVariables* → *ArrayList<Variable>***
Atribut ini berfungsi untuk menampung variabel biner yang telah diurut menaik berdasarkan koefisiennya.

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

- ***solve()* → *BalasAdditiveBIPSolverResult***
Fungsi ini berguna untuk mendapatkan solusi masalah *bipProblem* menggunakan algoritma *Balas's additive*.
- ***objectiveFunction()* → *BIPFunction***
Fungsi ini berguna untuk mendapatkan fungsi tujuan dari masalah *bipProblem*.
- ***indexOf(variable → Variable) → int***
Fungsi ini berguna untuk mendapatkan indeks *variable* dalam daftar variabel yang telah terurut.
- ***sortVariablesByCoefficient()* → *void***
Fungsi ini berguna untuk mengurutkan variabel secara menaik berdasarkan koefisiennya.
- ***branch(node → Node) → Node []***
Fungsi ini berguna untuk membuat cabang *node-0* dan *node-1* dari *node*.
- ***boundValue(node → Node) → double***
Fungsi ini berguna untuk menghitung nilai *bound* dari *node*.
- ***isBoundValue Worse Than Current Incumbent(node → Node, currentIncumbentObjectiveValue → double) → boolean***
Fungsi ini berguna untuk mengetahui apakah nilai *bound* pada *node* lebih buruk daripada nilai solusi *incumbent*.
- ***isBetterThanCurrentIncumbent(valueOneVariableList → ArrayList<Variable>, currentIncumbentObjectiveValue → double) → boolean***
Fungsi ini berguna untuk mengetahui apakah daftar variabel bernilai 1 dapat menghasilkan solusi yang lebih baik daripada solusi *incumbent*.
- ***isImpossibleToContinue(node → Node) → boolean***
Fungsi ini berguna untuk mengetahui apakah *node* tidak mungkin untuk menghasilkan *incumbent*.
- ***isConstraintImpossibleToSatisfy(constraint → BIPConstraint, valueOneVariableList → ArrayList<Variable>, freeVariableList → ArrayList<Variable>) → boolean***
Fungsi ini berguna untuk mengetahui apakah *constraint* dapat dipenuhi berdasarkan daftar variabel bernilai 1 dan daftar variabel bebas.

4.2.19 Kelas *BalasAdditiveBIPSolverResult*

BalasAdditiveBIPSolverResult
- problem : BIPProblem - valueOneVariableList : ArrayList<Variable> - enumeratedNodeList : ArrayList<Node> - solvingTime : long
+ BalasAdditiveBIPSolverResult(problem : BIPProblem, valueOneVariableList : ArrayList<Variable>, enumeratedNodeList : ArrayList<Node>, solvingTime : long) + getProblem() : BIPProblem + getValueOneVariableList() : ArrayList<Variable> + getEnumeratedNodeList() : ArrayList<Node> + getSolvingTime() : long + toString() : String

Gambar 4.23: Diagram kelas *BalasAdditiveBIPSolverResult*

Kelas ini merepresentasikan hasil dari penyelesaian masalah *binary integer programming* menggunakan algoritma *Balas's additive*. Diagram kelas *BalasAdditiveBIPSolverResult* dapat dilihat pada gambar 4.23. Berikut ini merupakan atribut-atribut yang terdapat pada kelas *BalasAdditiveBIPSolverResult*:

- ***bipProblem*** → ***BIPProblem***
Atribut ini berguna untuk menampung model masalah *binary integer programming*.
- ***valueOneVariableList*** → ***ArrayList<Variable>***
Atribut ini berguna untuk menampung solusi penyelesaian berupa daftar variabel yang bernilai 1.
- ***enumeratedNodeList*** → ***ArrayList<Node>***
Atribut ini berfungsi untuk menampung daftar *node* yang diperiksa selama melakukan penyelesaian masalah menggunakan algoritma *Balas's additive*.
- ***solvingTime*** → ***long***
Atribut ini berfungsi untuk menampung waktu yang digunakan dalam menyelesaikan masalah dalam satuan milisekon.

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

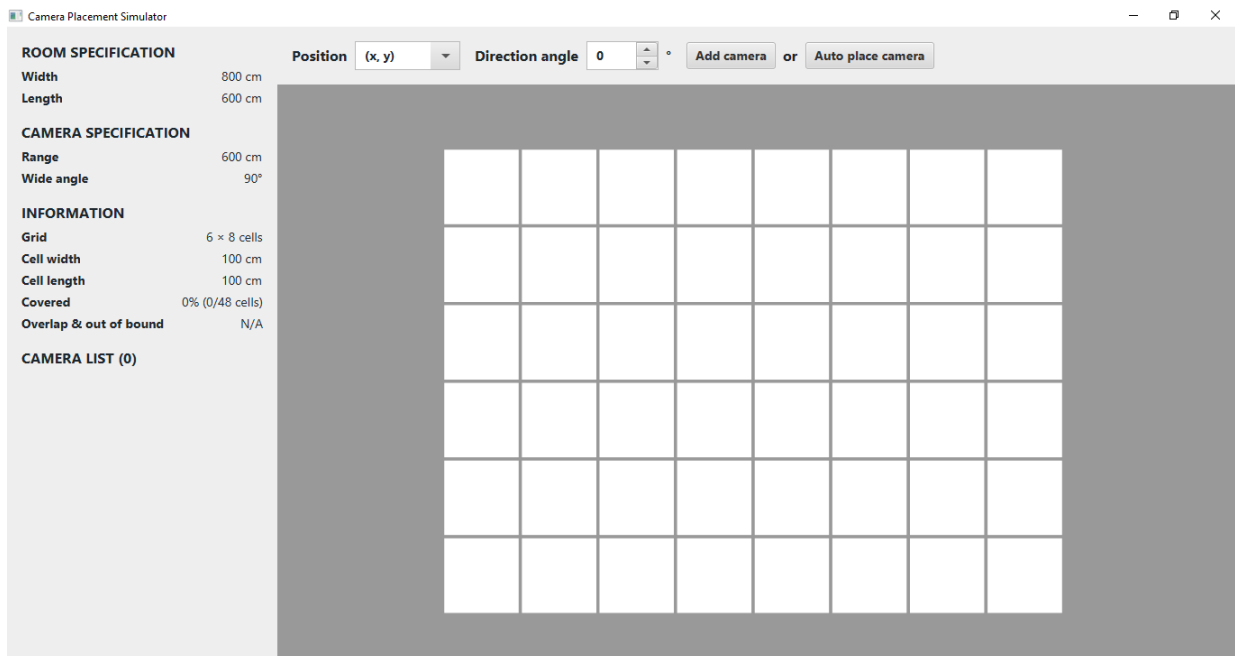
Pada bagian ini, akan dibahas hasil implementasi antarmuka sesuai dengan perancangan antarmuka yang dilakukan pada bab sebelumnya. Berikut ini adalah hasil dari implementasi antarmuka:

- 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

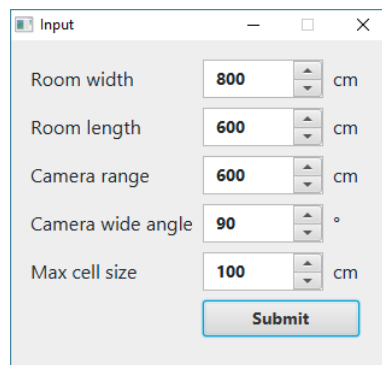
Pada bagian ini, perangkat lunak akan diuji untuk memastikan bahwa setiap fungsi-fungsi dalam perangkat lunak dapat bekerja sesuai tujuannya. Pengujian ini dilakukan sesuai dengan skenario-skenario yang terdapat pada use case. Berikut ini pengujian-pengujian fungsional yang dilakukan:

- 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.

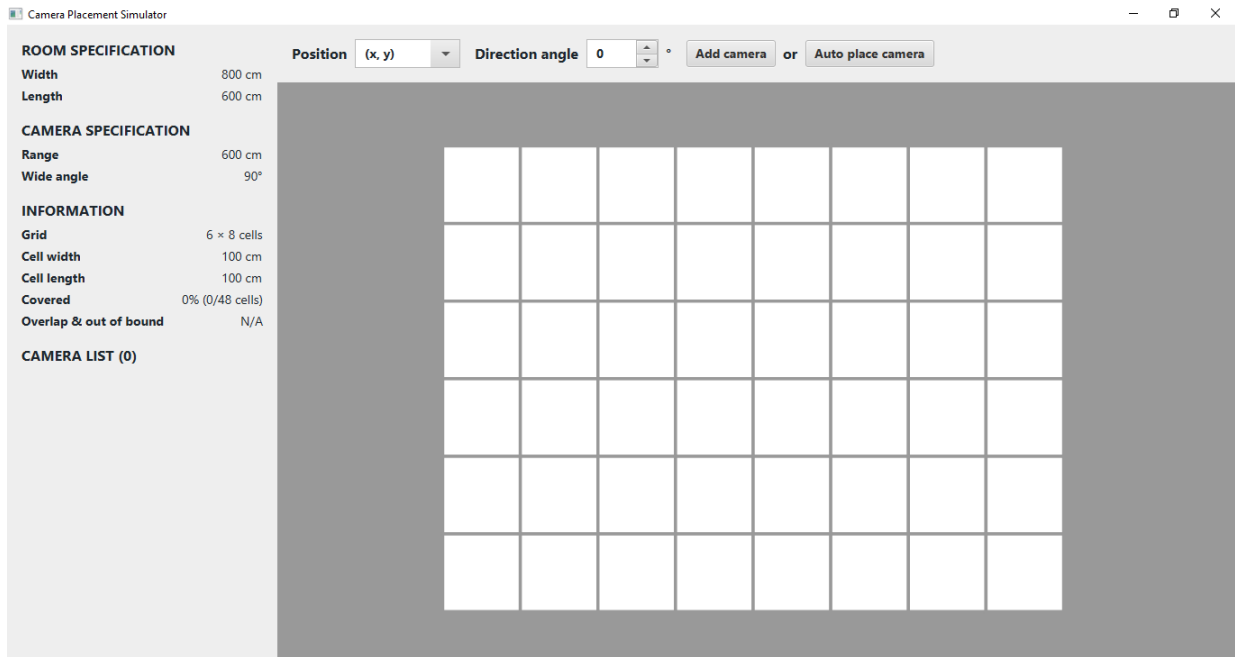


The image shows a software window titled "Input" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are five input fields, each with a label, a numeric input box, and a unit indicator. The inputs are: "Room width" with value 800 and unit "cm"; "Room 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

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

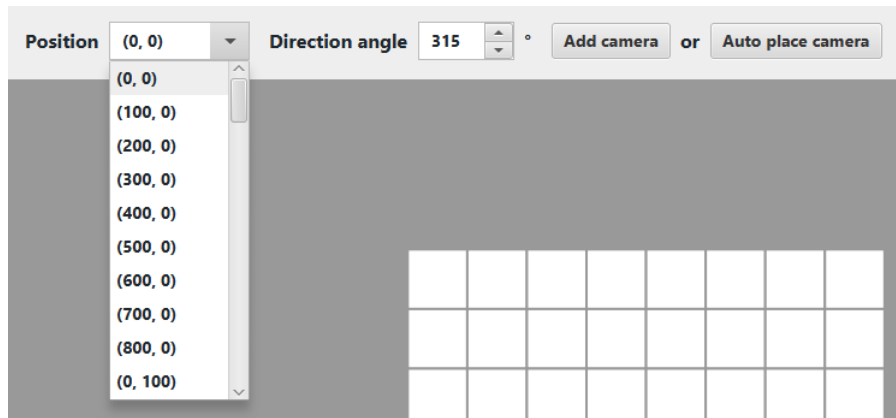
ROOM SPECIFICATION	
Width	800 cm
Length	600 cm
CAMERA SPECIFICATION	
Range	600 cm
Wide angle	90°
INFORMATION	
Grid	6 × 8 cells
Cell width	100 cm
Cell length	100 cm
Covered	0% (0/48 cells)
Overlap & out of bound	N/A
CAMERA LIST (0)	

Gambar 5.5: Tampilan panel informasi setelah pengisian masukan masalah

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

- Pengujian: **Menambahkan Penempatan Kamera CCTV**

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

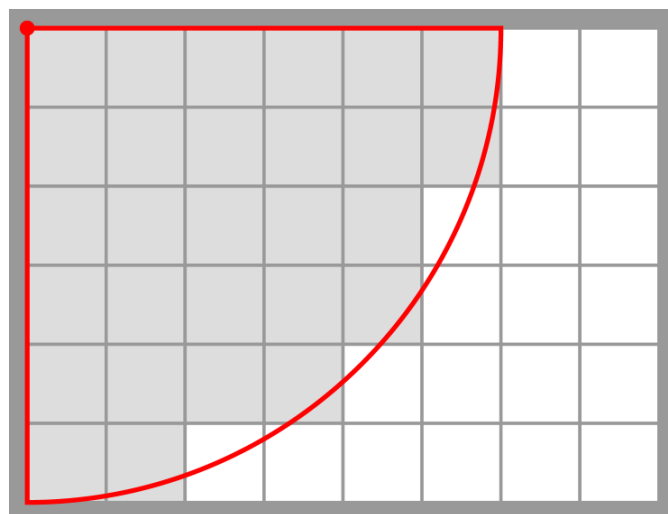


Gambar 5.6: Tampilan panel penambahan penempatan kamera CCTV

Setelah mengisi penempatan tersebut, tombol "add camera" ditekan. Perangkat lunak merespon penambahan penempatan tersebut dengan memperbaharui panel informasi dan 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

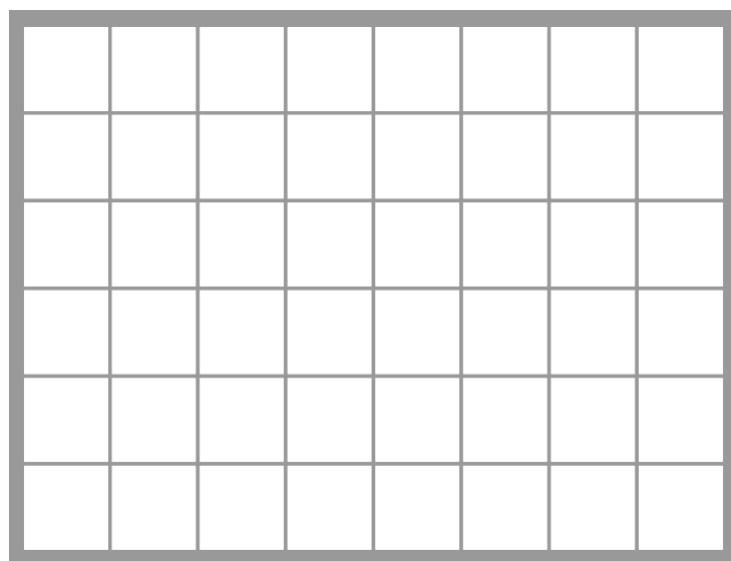
Pada panel informasi bagian daftar penempatan kamera CCTV, terdapat baris baru bertuliskan "At (0, 0) facing 315°" yang menunjukkan penempatan pada titik (0, 0) dengan sudut arah pandang 315°. Selain itu, informasi-informasi lainnya yang terdapat pada panel informasi juga telah diperbaharui. Pada panel 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.
- [2] Balas, E. (1965) An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, **13**, 517–546.
- [3] Narendra, P. M. dan Fukunaga, K. (1977) A branch and bound algorithm for feature subset selection. *IEEE Transactions on computers* , **?**, 917–922.
- [4] Horster, E. dan Lienhart, R. (2006) Approximating optimal visual sensor placement. *2006 IEEE International Conference on Multimedia and Expo*, pp. 1257–1260. IEEE.

LAMPIRAN A
KODE PROGRAM

LAMPIRAN B
HASIL EKSPERIMEN