TUGAS AKHIR - IF184802

IMPLEMENTASI ALGORITMA ASOSIASI DATA BERBASIS BATASAN STRUKTURAL PADA ONLINE MULTI OBJECT TRACKING

DANDY NAUFALDI
NRP 05111640000011

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Dini Adni Navastara, S.Kom., M.Sc.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

[*Halaman ini sengaja dikosongkan*]

TUGAS AKHIR - IF184802

IMPLEMENTASI ALGORITMA ASOSIASI DATA BERBASIS BATASAN STRUKTURAL PADA ONLINE MULTI OBJECT TRACKING

DANDY NAUFALDI
NRP 05111640000011

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Dini Adni Navastara, S.Kom., M.Sc.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

[*Halaman ini sengaja dikosongkan*]

UNDERGRADUATE THESIS - IF184802

IMPLEMENTATION OF DATA ASSOCIATION ALGORITHM BASED ON STRUCTURAL CONSTRAINT FOR ONLINE MULTI OBJECT TRACKING

DANDY NAUFALDI
NRP 05111640000011

Supervisor 1
Rully Soelaiman, S.Kom., M.Kom.

Supervisor 2
Dini Adni Navastara, S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

[*Halaman ini sengaja dikosongkan*]

LEMBAR PENGESAHAN

IMPLEMENTASI ALGORITMA ASOSIASI DATA BERBASIS BATASAN STRUKTURAL PADA ONLINE MULTI OBJECT TRACKING

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat Memperoleh Gelar
Sarjana Komputer
pada
Bidang Studi Algoritma Pemrograman Program Studi S-l
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh:
Dandy Naufaldi
NRP. 05111640000011

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom.
NIP. 197002131994021001 (Pembimbing 1)

Dini Adni Navastara, S.Kom., M.Sc.
NIP. 198510172015042001 (Pembimbing 2)

SURABAYA
JUNI 2020

[*Halaman ini sengaja dikosongkan*]

IMPLEMENTASI ALGORITMA ASOSIASI DATA BERBASIS BATASAN STRUKTURAL PADA ONLINE MULTI OBJECT TRACKING

Nama	:	Dandy Naufaldi
NRP	:	05111640000011
Departemen	:	Departemen Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas, ITS
Pembimbing I	:	Rully Soelaiman, S.Kom., M.Kom.
Pembimbing II	:	Dini Adni Navastara, S.Kom., M.Sc.

Abstrak

Object tracking merupakan proses pengenalan identitas objek lintas frame video. Proses ini banyak diterapkan dalam satu alur dengan proses object detection yang menghasilkan informasi bounding box objek. Dari object tracking, dapat diperoleh informasi lintasan pergerakan objek.

Kemajuan teknologi membuat perangkat perekaman video menjadi semakin kecil bahkan cukup dengan menggunakan ponsel. Perangkat perekaman tersebut memudahkan pengguna dalam merekam video kapanpun dan di manapun. Namun, rekaman video yang dihasilkan banyak mengalami masalah dari pergerakan perangkat perekam sehingga video menjadi tidak stabil. Pergerakan tersebut dapat berdampak negatif terhadap hasil proses object tracking karena posisi objek yang tidak stabil sehingga asosiasi data antara frame menjadi sulit dilakukan.

Pada tugas akhir ini, penulis akan mengimplementasikan algoritma asosiasi data berbasis batasan struktural pada online multi object tracking yang diusulkan oleh Yoon dkk. Pada algoritma ini, batasan struktural digunakan untuk mendapatkan posisi objek relatif terhadap objek lain sehingga dapat mengurangi dampak per-

gerakan kamera. Program object tracking yang dibuat terintegrasi dengan Kalman filter untuk mendapatkan prediksi posisi berdasarkan informasi posisi objek sebelumnya. Implementasi algoritma ini diujikan pada dataset 2DMOT15 dan mendapat nilai MOTA 20,4% pada data latih.

Kata kunci: asosiasi data, batasan struktural, dataset 2DMOT15, Kalman filter, online multi object tracking

IMPLEMENTATION OF DATA ASSOCIATION ALGORITHM BASED ON STRUCTURAL CONSTRAINT FOR ONLINE MULTI OBJECT TRACKING

Name	:	Dandy Naufaldi
Student ID	:	05111640000011
Department	:	Department of Informatics Engineering, Faculty of Intelligent Electrical and Informatics Technology, ITS
Supervisor I	:	Rully Soelaiman, S.Kom., M.Kom.
Supervisor II	:	Dini Adni Navastara, S.Kom., M.Sc.

Abstract

Object tracking is a process to recognize object's identity across frame in a video. This process has been implemented as a pipeline with object detection process which produce bounding boxes information. From the output of object tracking, we can get object's movement track.

Technology advances have enabled the existence of smaller video recording devices even a phone can give good recordings. Those recording devices enable users to record video anytime and anywhere. But, the video suffer a lot from the movement of the recording device which cause the video to be unstable. The movement might give bad impact to the result of object tracking because unstable object's position make data association process interframe hard to be done.

In this thesis, the author will implement a data association algorithm based on structural constraint for online multi object tracking proposed by Yoon et.al. In this algorithm, structural constraint is used to determine object's position relative to other objects which help reduce the effect of camera movement. The program will be

integrated with Kalman filter to help predict current object position based on past information. This implementation has been tested on 2DMOT15 dataset and achieved MOTA score of 20.4% on training data.

Keywords: data association, Kalman filter, online multi object tracking, structural constraint, 2DMOT15 dataset

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa. Atas rahmat dan kasih sayangNya, penulis dapat menyelesaikan tugas akhir dan laporan akhir dalam bentuk buku ini.

Pengerjaan buku ini penulis tujukan untuk mengeksplorasi lebih mendalam topik-topik yang tidak diwadahi oleh kampus, namun banyak menarik perhatian penulis. Selain itu besar harapan penulis bahwa pengerjaan tugas akhir sekaligus pengerjaan buku ini dapat menjadi batu loncatan penulis dalam menimba ilmu yang bermanfaat.

Penulis ingin menyampaikan rasa terima kasih kepada banyak pihak yang telah membimbing, menemani dan membantu penulis selama masa pengerjaan tugas akhir maupun masa studi.

1. Kedua orangtua, Hernowo dan Retno Susilowati, serta keluarga penulis sebagai penyemangat pertama yang telah memberikan dukungan doa dan moral sehingga penulis dapat menyelesaikan tugas akhir ini dengan baik.
2. Bapak Rully Soelaiman, S.Kom., M.Kom., selaku pembimbing penulis yang telah memberikan dukungan baik berupa bimbingan, ilmu, semangat, perhatian dan nasihat selama masa studi penulis.
3. Ibu Dini Adni Navastara, S.Kom., M.Sc., selaku pembimbing penulis yang telah memberikan dukungan baik berupa bimbingan dan ilmu dalam mengerjakan tugas akhir.
4. Kakak penulis, Dinda Novitasari, yang telah menjadi sosok panutan dalam kuliah dan menjadi teman diskusi dalam mengerjakan tugas akhir ini.
5. Seluruh dosen dan karyawan Departemen Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama masa kuliah di Teknik Informatika ITS.

6. Teman-teman keluarga laboratorium Komputasi Cerdas dan Visi yang telah menemani dan membantu penulis dalam berproses selama kuliah.
7. Yolanda, Fadli, dan Ferdinand yang telah menemani dan membantu penulis dalam berbagai kesempatan selama menempuh masa studi di Departemen Teknik Informatika ITS.
8. Teman-teman mahasiswa Departemen Teknik Informatika ITS, khususnya angkatan 2016 yang senantiasa membantu, menemani, memberi semangat dan kenangan selama kurang lebih 4 tahun masa studi.
9. Komunitas *open source*, grup diskusi di Telegram, dan forum Stack Overflow yang membantu penulis dalam mendapatkan jawaban atas kendala yang dihadapi selama mengerjakan tugas akhir ini.
10. Yoshima dan Yolanda yang telah membantu mengoreksi penulisan buku tugas akhir ini.
11. Serta semua pihak yang turut membantu penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa buku ini jauh dari kata sempurna. Maka dari itu, penulis memohon maaf apabila terdapat salah kata maupun makna pada buku ini. Penulis berharap buku ini dapat berkontribusi dalam ilmu pengetahuan dan memberikan manfaat bagi pembaca.

Surabaya, Juni 2020

Dandy Naufaldi

DAFTAR ISI

LEMBAR PENGESAHAN . . . . . **vii**
ABSTRAK . . . . . **ix**
ABSTRAK . . . . . **xi**
KATA PENGANTAR . . . . . **xiii**
DAFTAR ISI . . . . . **xv**
DAFTAR GAMBAR . . . . . **xix**
DAFTAR TABEL . . . . . **xxi**
DAFTAR PSEUDOCODE . . . . . **xxvii**
DAFTAR KODE SUMBER . . . . . **xxix**
BAB I PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Masalah 3
1.4 Tujuan 3
1.5 Manfaat 3
1.6 Metodologi 4
1.7 Sistematika Penulisan 5
BAB II DASAR TEORI **7**
2.1 Deteksi Objek 7
2.2 Asosiasi Data 7
2.3 Online Multi Object Tracking 8
2.4 K-Means 11
2.4.1 K-Means dengan Batasan 12
2.5 Kalman Filter 13
2.6 Asosiasi Data Berbasis Batasan Struktural 15

2.6.1 Structural Constraint Event Aggregation (SCEA) 17
2.6.2 Structural Constraint Object Recovery (SCOR) 23
2.6.3 Manajemen Objek dan Batasan Struktural . 24
2.7 Metrik dalam Multi Object Tracking 27
BAB III DESAIN **31**
3.1 Desain Umum Sistem 31
3.2 Desain Program Utama 31
3.2.1 Desain Fungsi Main 32
3.2.2 Desain Fungsi Object Tracking 33
3.2.3 Desain Fungsi SCEA 34
3.2.4 Desain Fungsi Gating 36
3.2.5 Desain Fungsi Partisi 37
3.2.6 Desain Fungsi Perhitungan Asosiasi Per Partisi 38
3.2.7 Desan Fungsi Pembangkit Kombinasi Asosiasi 40
3.2.8 Desain Fungsi Cost Fs 42
3.2.9 Desain Fungsi Cost Fa 43
3.2.10 Desain Fungsi Cost Fc 44
3.2.11 Desain Fungsi Penggabungan Matriks Asosiasi 46
3.2.12 Desain Fungsi SCOR 47
3.2.13 Desain Fungsi Penentuan Objek Gamma . 48
3.2.14 Desain Fungsi Perhitungan Matriks Cost . 50
3.2.15 Desain Fungsi Cost Fr 51
3.2.16 Desain Fungsi Penentuan Asosiasi 52
3.2.17 Desain Fungsi Pembaruan Nilai Objek . . . 53

3.2.18 Desain Fungsi Pembaruan Nilai Batasan Struktural 56
3.2.19 Desain Fungsi Prediksi Nilai Objek 57
3.2.20 Desain Fungsi Prediksi Batasan Struktural . 58
3.2.21 Desain Fungsi Eliminasi Objek 59
3.2.22 Desain Fungsi Pembuatan Objek Baru . . . 61
3.2.23 Desain Fungsi Seleksi Pelacakan Beruntun 63
BAB IV IMPLEMENTASI 65
4.1 Lingkungan Implementasi 65
4.2 Implementasi Program Utama 65
4.2.1 Penggunaan Pustaka 66
4.2.2 Implementasi Class DetectionState 67
4.2.3 Implementasi Class ObjectState 69
4.2.4 Implementasi Class StructuralConstraint . . 71
4.2.5 Implementasi Modul cost 72
4.2.6 Implementasi Modul partition 75
4.2.7 Implementasi Modul scea 78
4.2.8 Implementasi Modul scor 88
4.2.9 Implementasi Class ObjectStateTracker . . 92
4.2.10 Implementasi Class StructuralConstraint-Tracker 97
4.2.11 Implementasi Class Tracker 101
4.2.12 Implementasi Fungsi Main 126
BAB V UJI COBA DAN EVALUASI 131
5.1 Lingkungan Uji Coba 131
5.2 Dataset 131
5.3 Skenario Uji Coba 135
5.3.1 Uji Coba Performa Asosiasi Data 136
5.3.2 Uji Coba Prediksi Kalman Filter 138

5.3.3 Uji Coba Metode Inisialisasi Objek Baru . 140
5.3.4 Uji Coba Batas Minimal Deteksi Beruntun 141
5.3.5 Uji Coba Luar pada Data Uji 142
5.4 Hasil dan Analisis 143
BAB VI KESIMPULAN DAN SARAN 153
6.1 Kesimpulan 153
6.2 Saran 154
DAFTAR PUSTAKA 155
LAMPIRAN A: Detail Hasil Uji Coba Performa Asosiasi Data 159
LAMPIRAN B: Detail Hasil Uji Coba Prediksi Kalman Filter 165
LAMPIRAN C: Detail Hasil Uji Coba Metode Inisialisasi Objek Baru 167
LAMPIRAN D: Detail Hasil Uji Coba Batas Minimal Deteksi Beruntun 169
LAMPIRAN E: Detail Hasil Uji Luar dengan Data Uji . 173
BIODATA PENULIS 175

DAFTAR GAMBAR

Gambar 2.1: Hasil Deteksi Wajah [6] 8

Gambar 2.2: Contoh Asosiasi Data [7] 9

Gambar 2.3: Ilustrasi *Online* (atas) dan *Offline* (bawah) *Tracking* [9] 10

Gambar 2.4: Ilustrasi Proses Pembuatan Partisi [3]: (a) Ruang Pencarian Awal, (b) Eliminasi dengan Gating, (c) Partisi dengan K-Means 20

Gambar 2.5: Contoh Penggunaan Batasan Struktural dalam Anchor [3] 21

Gambar 2.6: Perbandingan *Ground Truth* dengan Hipotesis *Tracking*: (a) *Frame* t, (b) *Frame* $t + 1$, dan (c) *Frame* $t + 2$ 28

Gambar 3.1: Diagram Alir Sistem 31

Gambar 3.2: Ilustrasi Pemilihan Objek yang Menjadi *Anchor* dalam Kombinasi Asosiasi 39

Gambar 3.3: Ilustrasi Kombinasi Asosiasi antara Objek dan Deteksi 41

Gambar 3.4: Ilustrasi Penggunaan Batasan Struktural dalam Menentukan Posisi di Perhitungan IoU: (a) Kondisi Awal dan (b) Kondisi Setelah Pergeseran Berdasarkan Batasan Struktural . . . 44

Gambar 3.5: *Sliding Window* antara *Frame t-1* dan *t* 61

Gambar 3.6: Ilustrasi Seleksi berdasarkan Pelacakan Beruntun dengan Batas Minimal Sebesar Dua . . 63

Gambar 5.1: Perbandingan *Bounding Box* pada Video KITTI-17 *Frame* ke-1: (a) data *gt* dan (b) data *det* 134
Gambar 5.2: Peringkat Hasil Program *Object Tracking* pada *Website* MOTChallenge 143
Gambar 5.3: Perbandingan Data pada Video KITTI-13 *Frame* ke-12 hingga 16: (a) data *gt* dan (b) data *det* 150
Gambar 5.4: Ilustrasi *Sliding Window* Deteksi yang Tidak Terasosiasi pada Video KITTI-13 *gt Frame* 71 dan 72 151

DAFTAR TABEL

Tabel 2.1: Perbandingan *Online* dan *Offline Tracking* [9] . 11

Tabel 3.1: Masukan, Proses, dan Keluaran dari Fungsi MAIN 32

Tabel 3.2: Masukan, Proses, dan Keluaran dari Fungsi OBJECTTRACKER 33

Tabel 3.3: Masukan, Proses, dan Keluaran dari Fungsi SCEA 35

Tabel 3.4: Masukan, Proses, dan Keluaran dari Fungsi GATING 36

Tabel 3.5: Masukan, Proses, dan Keluaran dari Fungsi SUBGROUPBYCLUSTER 37

Tabel 3.6: Masukan, Proses, dan Keluaran dari Fungsi ASSIGNMENTBYSUBGROUP 39

Tabel 3.7: Masukan, Proses, dan Keluaran dari Fungsi ASSIGNMENTGENERATOR 42

Tabel 3.8: Masukan, Proses, dan Keluaran dari Fungsi FS . 43

Tabel 3.9: Masukan, Proses, dan Keluaran dari Fungsi FA . 43

Tabel 3.10: Masukan, Proses, dan Keluaran dari Fungsi FC . 45

Tabel 3.11: Masukan, Proses, dan Keluaran dari Fungsi AGGREGATE 46

Tabel 3.12: Masukan, Proses, dan Keluaran dari Fungsi SCOR 48

Tabel 3.13: Masukan, Proses, dan Keluaran dari Fungsi GETGAMMAOBJECTS 49

Tabel 3.14: Masukan, Proses, dan Keluaran dari Fungsi CALCASSIGNMENTCOST 50

Tabel 3.15: Masukan, Proses, dan Keluaran dari Fungsi FR . 51

Tabel 3.16: Masukan, Proses, dan Keluaran dari Fungsi ASSIGNMENT 52
Tabel 3.17: Masukan, Proses, dan Keluaran dari Fungsi UPDATEOBJECT 55
Tabel 3.18: Masukan, Proses, dan Keluaran dari Fungsi UPDATESC 57
Tabel 3.19: Masukan, Proses, dan Keluaran dari Fungsi PREDICTOBJECT 58
Tabel 3.20: Masukan, Proses, dan Keluaran dari Fungsi PREDICTSC 59
Tabel 3.21: Masukan, Proses, dan Keluaran dari Fungsi ELIMINATE 60
Tabel 3.22: Masukan, Proses, dan Keluaran dari Fungsi NEWOBJECTS 62
Tabel 3.23: Masukan, Proses, dan Keluaran dari Fungsi FILTERSTREAK 64

Tabel 4.1: Nama dan Penjelasan Pustaka 67
Tabel 4.2: Nama dan Penjelasan Variabel Instance dari Class DETECTIONSTATE 67
Tabel 4.3: Nama dan Penjelasan Variabel Instance dari Class OBJECTSTATE 70
Tabel 4.4: Nama dan Penjelasan Variabel Instance dari Class STRUCTURALCONSTRAINT 71
Tabel 4.5: Nama dan Penjelasan Variabel Instance dari Class CONFIGSCEA 79
Tabel 4.6: Nama dan Penjelasan Variabel Instance dari Class CONFIGSCOR 89
Tabel 4.7: Nama dan Penjelasan Variabel Instance dari Class OBJECTSTATETRACKER 93

Tabel 4.8: Nama dan Penjelasan Variabel Instance dari Class StructuralConstraintTracker . . . 97
Tabel 4.9: Nama dan Penjelasan Variabel Instance dari Class Tracker 102

Tabel 5.1: Detail Data Anotasi 132
Tabel 5.2a: Spesifikasi Data Latih dari 2DMOT15 (1) . . . 132
Tabel 5.2b: Spesifikasi Data Latih dari 2DMOT15 (2) . . . 133
Tabel 5.3a: Spesifikasi Data Uji dari 2DMOT15 (1) 133
Tabel 5.3b: Spesifikasi Data Uji dari 2DMOT15 (2) 134
Tabel 5.4: Parameter Dasar pada Sistem *Object Tracking* . 136
Tabel 5.5: Perbandingan Pasangan ID Hasil Asosiasi: (a) *Ground Truth* dan (b) Prediksi 137
Tabel 5.6: Pemetaan Nilai IoU ke $default_case_d0$ pada Fungsi *Cost* Orisinal dan Modifikasi 138
Tabel 5.7: Nilai *Weighted Precision* dan *Recall* pada Uji Asosiasi Data 139
Tabel 5.8: Nilai Rata-Rata RMSE per Komponen Pengukuran pada *Kalman Filter* untuk Objek 139
Tabel 5.9: Nilai Hasil Uji Metode Inisialisasi Objek Baru . 141
Tabel 5.10: Nilai Hasil Uji Coba Batas Minimal Deteksi Beruntun 142
Tabel 5.11: Nilai Hasil Uji Coba Luar pada Data Uji 142
Tabel 5.12: Parameter Optimal untuk Sistem 144
Tabel 5.13: Perbandingan Rata-Rata Banyak Pasangan Asosiasi per *Frame* di Data Latih *gt* 145
Tabel 5.14: Perbandingan Rata-Rata Banyak Kombinasi Asosiasi yang Dievaluasi pada Data Latih gt . . 146
Tabel 5.15: Perbandingan Banyak Data *Bounding Box* pada Data Latih *gt* dan *det* 147

Tabel 5.16: Daftar Nilai IoU dari Pasangan Deteksi pada *Frame* 71 dan 72 149

Tabel A.1: Hasil Uji Asosiasi dengan Fungsi *Cost* Orisinal dan $default_cost_d0$ 1,2 159
Tabel A.2: Hasil Uji Asosiasi dengan Fungsi *Cost* Orisinal dan $default_cost_d0$ 1,6 160
Tabel A.3: Hasil Uji Asosiasi dengan Fungsi *Cost* Orisinal dan $default_cost_d0$ 2,3 160
Tabel A.4: Hasil Uji Asosiasi dengan Fungsi *Cost* Orisinal dan $default_cost_d0$ 3,5 161
Tabel A.5: Hasil Uji Asosiasi dengan Fungsi *Cost* Modifikasi dan $default_cost_d0$ 1,0 161
Tabel A.6: Hasil Uji Asosiasi dengan Fungsi *Cost* Modifikasi dan $default_cost_d0$ 0,9 162
Tabel A.7: Hasil Uji Asosiasi dengan Fungsi *Cost* Modifikasi dan $default_cost_d0$ 0,8 162
Tabel A.8: Hasil Uji Asosiasi dengan Fungsi *Cost* Modifikasi dan $default_cost_d0$ 0,7 163

Tabel B.1: Hasil Uji Prediksi Kalman Filter dengan Matriks Q Orisinal 165
Tabel B.2: Hasil Uji Prediksi Kalman Filter dengan Matriks Q Modifikasi 166

Tabel C.1: 167
Tabel C.2: Hasil Uji Inisialisasi Objek Baru dengan Metode II 168

Tabel D.1: 169
Tabel D.2: Hasil Uji Coba Batas Minimal Deteksi Beruntun Sebesar 3 170

Tabel D.3: Hasil Uji Coba Batas Minimal Deteksi Beruntun Sebesar 4 171

Tabel E.1: 173

[*Halaman ini sengaja dikosongkan*]

DAFTAR PSEUDOCODE

Pseudocode 2.1: K-MEANS 12
Pseudocode 2.2: OBJECT TRACKING DENGAN SCEA DAN SCOR 17
Pseudocode 2.3: SCEA 18
Pseudocode 3.1: Fungsi MAIN 32
Pseudocode 3.2: Fungsi OBJECTTRACKER 34
Pseudocode 3.3: Fungsi SCEA 35
Pseudocode 3.4: Fungsi GATING 36
Pseudocode 3.5a: Fungsi SUBGROUPBYCLUSTER (1) . . 37
Pseudocode 3.5b: Fungsi SUBGROUPBYCLUSTER (2) . . 38
Pseudocode 3.6a: Fungsi ASSIGNMENTBYSUBGROUP (1) 39
Pseudocode 3.6b: Fungsi ASSIGNMENTBYSUBGROUP (2) 40
Pseudocode 3.7: Fungsi ASSIGNMENTGENERATOR . . . 42
Pseudocode 3.8: Fungsi FS 43
Pseudocode 3.9: Fungsi FA 44
Pseudocode 3.10: Fungsi FC 45
Pseudocode 3.11a: Fungsi AGGREGATE (1) 46
Pseudocode 3.11b: Fungsi AGGREGATE (2) 47
Pseudocode 3.12: Fungsi SCOR 48
Pseudocode 3.13a: Fungsi GETGAMMAOBJECTS (1) . . . 49
Pseudocode 3.13b: Fungsi GETGAMMAOBJECTS (2) . . . 50
Pseudocode 3.14a: Fungsi CALCASSIGNMENTCOST (1) . 50
Pseudocode 3.14b: Fungsi CALCASSIGNMENTCOST (2) . 51
Pseudocode 3.15: Fungsi FR 52
Pseudocode 3.16: Fungsi ASSIGNMENT 53
Pseudocode 3.17: Fungsi UPDATEOBJECT 56

Pseudocode 3.18: Fungsi UPDATESC 57
Pseudocode 3.19: Fungsi PREDICTOBJECT 58
Pseudocode 3.20: Fungsi PREDICTSC 59
Pseudocode 3.21: Fungsi ELIMINATE 60
Pseudocode 3.22: Fungsi NEWOBJECTS Metode I 62
Pseudocode 3.23a:Fungsi NEWOBJECTS Metode II (1) . . 62
Pseudocode 3.23b:Fungsi NEWOBJECTS Metode II (2) . . 63
Pseudocode 3.24: Fungsi FILTERSTREAK 64

DAFTAR KODE SUMBER

Kode Sumber 4.1 Daftar Deklarasi *Import* Program 66
Kode Sumber 4.2 *Class* DETECTIONSTATE 68
Kode Sumber 4.1a Fungsi Instansiasi dari Bounding Box (1) 68
Kode Sumber 4.1b Fungsi Instansiasi dari Bounding Box (2) 69
Kode Sumber 4.2 *Class* OBJECTSTATE 70
Kode Sumber 4.3 *Class* STRUCTURALCONSTRAINT 71
Kode Sumber 4.4 Fungsi Instansiasi dari Pasangan ObjectState 72
Kode Sumber 4.5 Fungsi Cost Fs 73
Kode Sumber 4.6 Fungsi Cost Fa 73
Kode Sumber 4.7 Fungsi Cost Fc 74
Kode Sumber 4.8a Fungsi Cost Fr (1) 74
Kode Sumber 4.8b Fungsi Cost Fr (2) 75
Kode Sumber 4.9 Fungsi Gating 76
Kode Sumber 4.10 Fungsi Partisi 77
Kode Sumber 4.11 Fungsi Pembangkit Kombinasi Asosiasi 78
Kode Sumber 4.12 Class CONFIGSCEA 79
Kode Sumber 4.13 Fungsi Cost Anchor 79
Kode Sumber 4.14 Fungsi Cost Non-Anchor 80
Kode Sumber 4.15a Fungsi Cost Berdasarkan Anchor dalam Kombinasi (1) 81
Kode Sumber 4.15b Fungsi Cost Berdasarkan Anchor dalam Kombinasi (2) 82
Kode Sumber 4.16 Fungsi Perhitungan Cost Asosiasi dari Kombinasi 83

Kode Sumber 4.17a Fungsi Perhitungan Asosiasi Per Partisi (1) 84
Kode Sumber 4.17b Fungsi Perhitungan Asosiasi Per Partisi (2) 85
Kode Sumber 4.18a Fungsi Penggabungan Matriks Asosiasi (1) 85
Kode Sumber 4.18b Fungsi Penggabungan Matriks Asosiasi (2) 86
Kode Sumber 4.19a Fungsi Penentuan Matriks Asosiasi (1) . 87
Kode Sumber 4.19b Fungsi Penentuan Matriks Asosiasi (2) . 88
Kode Sumber 4.20 *Class* CONFIGSCOR 89
Kode Sumber 4.21 Fungsi Penentuan Objek Gamma 89
Kode Sumber 4.22a Fungsi Perhitungan Matriks Cost (1) . . 90
Kode Sumber 4.22b Fungsi Perhitungan Matriks Cost (2) . . 91
Kode Sumber 4.23a Fungsi Penentuan Asosiasi (1) 91
Kode Sumber 4.23b Fungsi Penentuan Asosiasi (2) 92
Kode Sumber 4.24 *Class* OBJECTSTATETRACKER 93
Kode Sumber 4.25a Fungsi Constructor ObjectStateTracker (1) 94
Kode Sumber 4.25b Fungsi Constructor ObjectStateTracker (2) 95
Kode Sumber 4.26 Fungsi Pembaruan State Objek 96
Kode Sumber 4.27 Fungsi Pembaruan Histogram Objek . . 96
Kode Sumber 4.28 Fungsi Prediksi State Objek 97
Kode Sumber 4.29a *Class* STRUCTURALCONSTRAINTTRACKER (1) 97
Kode Sumber 4.29b *Class* STRUCTURALCONSTRAINTTRACKER (2) 98
Kode Sumber 4.30 Fungsi Constructor StructuralConstraintTracker 99
Kode Sumber 4.31 Fungsi Pembaruan State Batasan Struktural 100
Kode Sumber 4.32 Fungsi Prediksi State Batasan Struktural 101

Kode Sumber 4.33a *Class* TRACKER (1) 102
Kode Sumber 4.33b *Class* TRACKER (2) 103
Kode Sumber 4.33c *Class* TRACKER (3) 104
Kode Sumber 4.34 Fungsi Constructor Tracker 105
Kode Sumber 4.35a Fungsi Pemroses SCEA (1) 106
Kode Sumber 4.35b Fungsi Pemroses SCEA (2) 107
Kode Sumber 4.35c Fungsi Pemroses SCEA (3) 108
Kode Sumber 4.36a Fungsi Pemroses SCOR (1) 108
Kode Sumber 4.36b Fungsi Pemroses SCOR (2) 109
Kode Sumber 4.36c Fungsi Pemroses SCOR (3) 110
Kode Sumber 4.37 Fungsi Update Objek Terlacak dengan Deteksi 111
Kode Sumber 4.38 Fungsi Update ObjectTracker 112
Kode Sumber 4.39a Fungsi Update StructuralConstraint-Tracker (1) 113
Kode Sumber 4.39b Fungsi Update StructuralConstraint-Tracker (2) 114
Kode Sumber 4.40 Fungsi Update Counter Frame Terakhir Terlacak dan Terlacak Beruntun 115
Kode Sumber 4.41a Fungsi Eliminasi Objek (1) 115
Kode Sumber 4.41b Fungsi Eliminasi Objek (2) 116
Kode Sumber 4.42 Fungsi Prediksi ObjectStateTracker . . 117
Kode Sumber 4.43 Fungsi Prediksi StructuralConstraint-Tracker 117
Kode Sumber 4.44a Fungsi Pembuatan Objek Metode I (1) . 118
Kode Sumber 4.44b Fungsi Pembuatan Objek Metode I (2) . 119
Kode Sumber 4.45a Fungsi Pembuatan Objek Metode II (1) 119
Kode Sumber 4.45b Fungsi Pembuatan Objek Metode II (2) 120
Kode Sumber 4.46a Fungsi Update Tracker (1) 121

Kode Sumber 4.46b Fungsi Update Tracker (2) 122
Kode Sumber 4.46c Fungsi Update Tracker (3) 123
Kode Sumber 4.46d Fungsi Update Tracker (4) 124
Kode Sumber 4.46e Fungsi Update Tracker (5) 125
Kode Sumber 4.46f Fungsi Update Tracker (6) 126
Kode Sumber 4.47a Fungsi MAIN (1) 127
Kode Sumber 4.47b Fungsi MAIN (2) 128
Kode Sumber 4.47c Fungsi MAIN (3) 129

BAB I
PENDAHULUAN

Pada bab ini, akan dijelaskan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi pengerjaan, dan sistematika penulisan tugas akhir.

1.1. Latar Belakang

Pemanfaatan teknologi video dalam bidang keamanan sudah semakin canggih. Banyak teknologi deteksi yang diterapkan pada video , seperti pengenalan macam objek, pengenalan wajah, dan pengenalan pelat nomor. Dari proses deteksi objek akan didapatkan keluaran berupa *bounding box* yang membatasi bagian dari *frame* yang mengandung objek. Dari objek-objek yang diperoleh dari satu *frame*, proses berikutnya adalah *object tracking* yaitu proses mengenali identitas objek antar *frame*, sehingga didapatkan asosiasi antara kemunculan objek dalam *frame* dengan identitas yang diberikan. Salah satu masalah dalam asosiasi objek dan identitas adalah faktor pergerakan kamera yang memengaruhi jalur pergerakan objek antar *frame*. Faktor tersebut dapat memberikan dampak negatif terhadap hasil asosiasi berupa kesalahan dalam memberikan identitas dan ketidakmampuan dalam mendeteksi objek yang hilang sesaat dalam *frame*.

Metode *multi object tracking* yang sudah ada, kurang handal menangani masalah pergerakan pada alat perekam video. Metode yang sudah ada bersifat *batch* atau *semi online*, seperti pada [1] dan [2]. Untuk mengatasi masalah pergerakan pada kamera, diperlukan penggunaan informasi batasan struktural terkait posisi dan pergerakan objek. Pada [3], telah dicoba penggunaan informasi batasan struktural untuk *online multi object tracking* yang mampu memberikan hasil lebih baik pada video yang kurang stabil.

Dalam tugas akhir ini, akan diimplementasikan algoritma

asosiasi objek yang terdeteksi dengan identitasnya yang lebih mampu menangani variasi pergerakan kamera berupa *yaw*, *pitch*, dan gerak translasi. Algoritma ini bekerja berdasarkan batasan struktural dari objek yang terdeteksi dalam *frame*. Algoritma ini terbagi dalam dua tahap, yaitu *Structural Constraint Event Aggregation* (SCEA) dan *Structural Constraint Object Recovery* (SCOR). SCEA bertugas untuk mengasosiasi antara objek yang terdeteksi pada *frame* saat ini dengan deteksi yang pernah ada. Berikutnya, pada SCOR dilakukan asosiasi antara objek yang menghilang dari *frame* terdahulu dengan deteksi yang ada sekarang. Data yang digunakan diambil dari dataset MOTChallenge 2015 [4]. Dataset ini berisikan *frame* dari total 22 video dengan label *bounding box* deteksi pejalan kaki dan identitas berupa nomor unik untuk setiap objek pejalan kaki.

Dari tugas akhir ini, diharapkan dapat membantu proses pelacakan objek dalam rekaman video terutama saat data rekaman video yang dimiliki berasal dari perangkat bergerak sehingga banyak terdapat goncangan dalam video dan diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut :

1. Bagaimana cara memanfaatkan batasan struktural antar objek dalam proses asosiasi data?
2. Bagaimana cara untuk mengoptimasi fungsi *cost* yang digunakan dalam asosiasi data?
3. Bagaimana cara mengatasi deteksi yang *false positive*?

1.3. Batasan Masalah

Permasalahan yang dibahas pada tugas akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Dataset yang digunakan menggunakan dataset MOTChallenge 2015 [4] dalam format 2D.
2. Implementasi algoritma menggunakan bahasa pemrograman Python.
3. Masukan program berupa citra per *frame* dari video beserta label *bounding box* hasil deteksi objek dari *frame* tersebut.

1.4. Tujuan

Tujuan tugas akhir ini adalah sebagai berikut:

1. Memanfaatkan batasan struktural melalui algoritma SCEA (*Structural Constraint Event Aggregation*) dan SCOR (*Structural Constraint Object Recovery*) untuk menyelesaikan kasus *online multi object tracking*.
2. Mengoptimasi fungsi *cost* yang digunakan untuk menghasilkan asosiasi data yang lebih baik.
3. Mengatasi data deteksi *false positive* untuk memperbaiki performa *object tracking*.

1.5. Manfaat

Manfaat tugas akhir ini adalah sebagai berikut:

1. Membantu memahami penerapan algoritma SCEA dan SCOR dalam menyelesaikan permasalahan *online multi object tracking*.
2. Membantu memahami kendala dalam *online multi object tracking* dengan data video yang tidak stabil.
3. Mengetahui hasil evaluasi kinerja *tracker* dengan algoritma SCEA dan SCOR.

1.6. Metodologi

Metodologi pengerjaan yang digunakan pada tugas akhir ini memiliki beberapa tahapan. Tahapan-tahapan tersebut yaitu:

1. Penyusunan proposal
   Pada tahapan ini penulis memberikan penjelasan mengenai apa yang penulis akan lakukan dan mengapa tugas akhir ini dilakukan. Penjelasan tersebut dituliskan dalam bentuk proposal tugas akhir.
2. Studi literatur
   Pada tahapan ini penulis mengumpulkan referensi yang diperlukan guna mendukung pengerjaan tugas akhir. Referensi yang digunakan dapat berupa hasil penelitian yang sudah pernah dilakukan, buku, artikel internet, atau sumber lain yang bisa dipertanggungjawabkan.
3. Desain
   Pada tahapan ini, penulis melakukan desain rancangan implementasi algoritma SCEA dan SCOR yang akan digunakan untuk *online multi object tracking*.
4. Implementasi algoritma
   Pada tahapan ini penulis mulai mengembangkan algoritma sesuai dengan hasil rancangan desain untuk menyelesaikan permasalahan. Implementasi ini dilakukan dengan menggunakan bahasa pemrograman Python.
5. Pengujian dan evaluasi
   Pada tahapan ini penulis menguji performa implementasi algoritma SCEA dan SCOR menggunakan dataset MOTChallenge 2015. Evaluasi dilakukan dari aspek asosiasi data dan *multi object tracking*.
6. Penyusunan buku
   Pada tahapan ini penulis menyusun hasil pengerjaan tugas akhir mengikuti format penulisan tugas akhir.

1.7. Sistematika Penulisan

Sistematika laporan tugas akhir yang akan digunakan adalah sebagai berikut :

1. BABI : PENDAHULUAN
   Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan tugas akhir.
2. BAB II : DASAR TEORI
   Bab ini berisi dasar teori mengenai algoritma yang digunakan dalam tugas akhir
3. BAB III : DESAIN
   Bab ini berisi desain algoritma dan struktur data yang digunakan dalam penyelesaian permasalahan.
4. BAB IV : IMPLEMENTASI
   Bab ini berisi implementasi berdasarkan desain algortima yang telah dilakukan pada tahap desain.
5. BAB V : PENGUJIAN DAN EVALUASI
   Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.
6. BAB VI : PENUTUP
   Bab ini berisi kesimpulan dan saran yang didapat dari hasil uji coba yang telah dilakukan.

[*Halaman ini sengaja dikosongkan*]

BAB II
DASAR TEORI

Pada bab ini dijelaskan beberapa dasar teori yang akan penulis gunakan sebagai landasan pengerjaan tugas akhir ini.

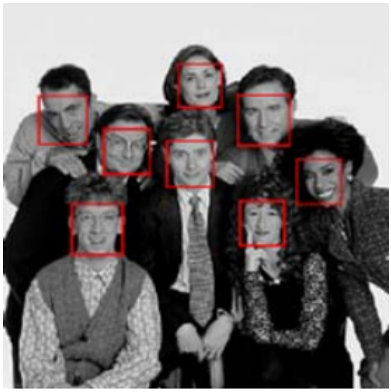
2.1. Deteksi Objek

Deteksi objek merupakan proses mendapatkan semua objek yang berasal dari satu atau beberapa kelas yang ada dalam satu citra terlepas dari skala, lokasi, pose, sudut pandang kamera, halangan dan kondisi pencahayaan [5]. Deteksi objek menjadi proses pertama yang dilakukan dalam banyak sistem visi komputer karena dapat membantu memperoleh informasi lebih lanjut, antara lain mengenali individu tertentu, *track* atau melacak objek dalam rangkaian citra, dan menggali informasi tambahan seperti menentukan keberadaan atau lokasi objek lain dalam satu gambar dan mengenali jenis pemandangan sebagai informasi kontekstual tambahan. Gambar 2.1 merupakan hasil deteksi wajah berupa kotak pembatas yang digambar di sekitar wajah-wajah yang terdapat dalam gambar.

Deteksi objek telah dimanfaatkan dalam berbagai bidang, seperti interaksi manusia dan komputer, robotika, peralatan elektronik, sistem temu kembali (mesin pencarian), dan transportasi (pengemudian otomatis).

2.2. Asosiasi Data

Asosiasi data adalah metode untuk menghubungkan kemunculan objek terdeteksi pada *frame* waktu t dengan objek yang muncul pada *frame* j, dengan $j < t$ serta selisih t dan j sesuai dengan *time window* yang digunakan [7]. Dengan asosiasi data, posisi suatu objek dapat dilacak dari *frame* ke *frame* sesuai label identitas yang telah diberikan sebelumnya. Salah satu metode untuk asosi-



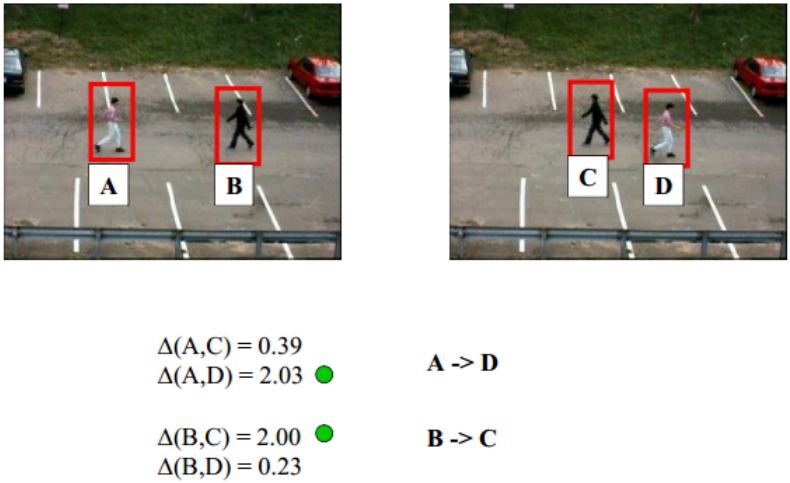
Gambar 2.1 Hasil Deteksi Wajah [6]

asi data adalah dengan menggunakan *linear assignment* pada *cost matrix* untuk mendapatkan asosiasi elemen matriks yang menghasilkan *cost* serendah mungkin. Contoh asosiasi data sebagai berikut pada Gambar 2.2.

Pada Gambar 2.2, di bagian kiri terdapat dua objek yaitu A dan B yang bergerak berlawanan arah. Lalu, pada bagian kanan dapat dilihat bahwa terdapat dua objek C dan D. Pada asosiasi data, akan dihitung *cost* untuk asosiasi yang mungkin seperti [(A, C), (B, D)] dan [(A, D), (B, C)] untuk mendapatkan *cost* terkecil, sehingga pada Gambar 2.2 *cost* terkecil didapat pada asosiasi [(A, D), (B, C)].

2.3. Online Multi Object Tracking

Object tracking merupakan permasalahan mengenai pengenalan pergerakan objek berdasarkan deretan gambar. Masukan dari *object tracking* berupa data-data pengukuran yang diambil dari gambar, seperti posisi beberapa titik di gambar, posisi dan momentum bagian gambar, dan data lainnya [8]. Data-data tersebut digunakan untuk merepresentasikan keadaan objek dan digunakan untuk

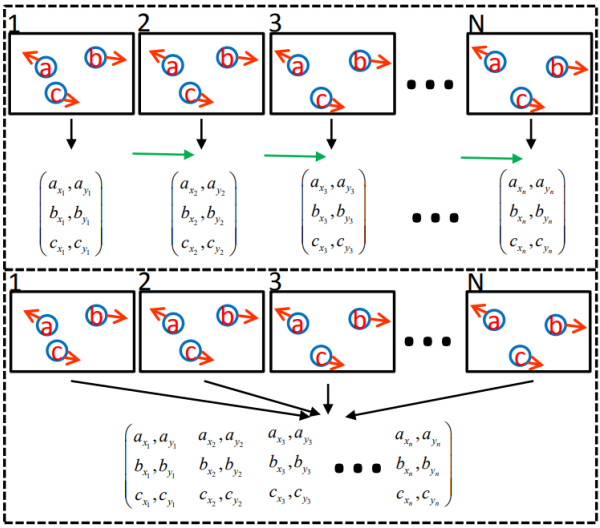


Gambar 2.2 Contoh Asosiasi Data [7]

memodelkan perubahan keadaan objek dari waktu ke waktu. Pada *multi object tracking*, banyaknya objek yang dilacak pergerakannya bisa lebih dari satu objek.

Terdapat dua strategi sederhana untuk melakukan *object tracking*, yaitu *tracking* berdasarkan deteksi dan *tracking* berdasarkan pencocokan [8]. Pada *tracking* berdasarkan deteksi, informasi awal yang dimiliki adalah hasil deteksi objek dari *frame* yang kemudian informasi ini dihubungkan dengan deteksi di *frame* lain untuk mendapatkan kecocokan dan menghasilkan *tracklet*. Pada *tracking* berdasarkan pencocokan, informasi awal yang dimiliki adalah model pergerakan objek yang digunakan untuk menemukan domain objek di *frame* ke-$n+1$ yang sesuai dengan domain objek di *frame* ke-n.

Berdasarkan metode pemrosesannya, ada dua metode *tracking* yaitu *online tracking* dan *offline tracking*. Perbedaan utama ada pada data yang digunakan, *online tracking* hanya menggunakan data terkini dan data-data sebelumnya, sedangkan *offline tracking*



Gambar 2.3 Ilustrasi *Online* (atas) dan *Offline* (bawah) *Tracking* [9]

menggunakan semua data secara keseluruhan atau dalam *batch* kecil sehingga dapat menggunakan informasi dari *frame* sesudah [9].

Pada Gambar 2.3 (atas), *online tracking* (atas) memiliki tiga objek a, b, dan c. Panah hijau menunjukkan observasi di masa lampau. Hasil *tracking* berupa lokasi dan ID objek yang dihasilkan secara langsung berdasarkan data observasi terkini. Pada Gambar 2.3 (bawah), *offline tracking* menggunakan data *frame* dalam bentuk *batch* atau kumpulan sehingga diperlukan data observasi secara keseluruhan terlebih dahulu sebelum dianalisis untuk didapatkan hasil *tracking*.

Detail perbedaan *online* dan *offline tracking* pada Tabel 2.1 menunjukkan bahwa *online tracking* memiliki kelebihan dapat digunakan untuk kebutuhan *realtime* sedangkan *offline* tracking secara teori mampu menghasilkan keluaran yang mencapai *global opti-*

Tabel 2.1 Perbandingan *Online* dan *Offline Tracking* [9]

Item	*Online Tracking*	*Offline Tracking*
Masukan	Observasi terkini	Semua observasi
Metode	Mengembangkan lintasan secara bertahap dengan observasi terkini	Menghubungkan semua observasi menjadi lintasan-lintasan
Kelebihan	Tepat untuk kebutuhan *realtime*	Dapat mencapai hasil *global optimum* secara teori
Kekurangan	Terkendala kurangnya observasi	Ada jeda hingga didapat hasil akhir

mum. Kelemahan dari *online tracking* adalah kurangnya data observasi yang dapat digunakan sehingga hasil yang didapat bisa terjebak *local optima*, sedangkan *offline tracking* memiliki jeda hingga didapat hasil akhir sehingga tidak dapat digunakan untuk kebutuhan *realtime*.

2.4. K-Means

K-Means merupakan salah satu algoritma *clustering* yang bersifat iteratif dengan tujuan untuk membagi data menjadi K *cluster*. Dalam K-Means diberikan bilangan bulat k dan himpunan n titik data $\chi \subset \mathbb{R}^d$. Kemudian dipilih k titik pusat C untuk meminimalkan fungsi potensi ϕ seperti pada Persamaan 2.1 [10]. *Pseudocode* dari algoritma K-Means dapat dilihat pada Pseudocode 2.1

$$\phi = \sum_{x \in \chi} \min_{c \in C} \|x - c\|^2 \tag{2.1}$$

K-Means menggunakan fungsi jarak dalam menentukan ke-

Pseudocode 2.1: K-MEANS

```
1: function KMeans(DATA, K)
2:     C ← initial K centers [c_1, c_2, ..., c_K] from Data
3:     while C not changed do
4:         for i in [1..K] do
5:             c_i ←{x | x ∈ Data,
                   dist(x, C_i) < dist(x, C_j) ∀j ∈ K ∧ i ≠ j}
6:         end for
7:         for i in [1..K] do
8:             C_i ← 1/|C_i| Σ_{x∈c_i} x
9:         end for
10:    end while
11: end function
```

anggotaan *cluster*. Umumnya fungsi yang digunakan adalah jarak *Euclidean* seperti pada Persamaan 2.2.

$$dist\,(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{2.2}$$

2.4.1. K-Means dengan Batasan

Penerapan K-Means standar pada data dengan dimensi tinggi ($n > 10$) atau data dengan persebarang yang luas dapat menghasilkan *cluster* dengan banyak anggota yang sangat tidak seimbang bahkan hanya berisikan 1 anggota. Hasil seperti ini tidap dapat digunakan dalam beberapa kasus yang membutuhkan penerapan banyak minimal dari anggota dalam satu *cluster*.

$$\begin{aligned} &\min_{C,T} \sum_{i=1}^{n} \sum_{h=1}^{K} T_{i,h}. \left(\frac{1}{2} \left\| x^i - C^h \right\| \right) \\ &\text{subject to} \sum_{h=1}^{K} T_{i,h} = 1, i = 1, \ldots, m, \\ &\qquad T_{i,h} \geq 0, i = 1, \ldots, m, h = 1, \ldots, k, \\ &\qquad \sum_{h=1}^{k} T_h \leq m. \end{aligned} \tag{2.3}$$

Penerapan batasan banyak anggota dalam K-Means dapat dimodelkan dengan Persamaan 2.3 [11]. Diberikan n titik data yang akan dikelompokkan menjadi K *cluster* dengan maksimal banyak anggota m di tiap $cluster$ dan C merupakan *centroid* dari tiap *cluster*. Untuk menerapkan batasan banyak anggota digunakan variabel T sebagai variabel selektor keanggotaan titik data terhadap suatu *cluster*. Penyelesaian K-Means dengan batasan maksimal banyak anggota mirip seperti K-Means standar tetapi memiliki perbedaan pada proses *update* keanggotaan *cluster* yang memperhatikan variabel T dan diselesaikan dengan optimasi jaringan linier *Minimum Cost Flow* [11].

2.5. Kalman Filter

Kalman filter merupakan rangkaian persamaan matematika yang digunakan untuk melakukan estimasi nilai keadaan suatu sistem dengan tujuan meminimalkan nilai *error* yang didapat. *Kalman filter* dapat melakukan prediksi nilai keadaan sistem untuk kondisi lampau, masa kini, dan yang akan datang dengan keakuratan yang tinggi tanpa mengetahui sifat atau model dari sistem yang diprediksi [12]. *Kalman filter* banyak diaplikasikan di bidang navigasi dan sensor tanam untuk meningkatkan kehandalan hasil pengukuran yang diproses.

Kalman filter memprediksi nilai keadaan sistem berdasarkan masukan nilai pengukuran atau observasi. Nilai pengukuran yang didapat umumnya memiliki *noise* atau kemungkinan *error* dan dapat ditangani dengan baik olek *Kalman filter*. Hal ini dimungkinkan dengan proses perhitungan yang melibatkan kovarians nilai pengukuran dan lingkungan eksternal sistem untuk mendapatkan nilai keadaan yang paling mungkin dari ruang probabilitas nilai keadaan sistem. Proses di dalam *Kalman filter* terbagi dalam dua tahap, yaitu proses perhitungan nilai terhadap waktu dan perhitungan *update* nilai internal terhadap nilai pengukuran.

$$\begin{aligned} \hat{x}_k^- &= F\hat{x}_{k-1} + Bu_{k-1} \\ P_k^- &= FP_{k-1}^- F^T + Q \end{aligned} \tag{2.4}$$

Proses perhitungan nilai terhadap waktu ada pada Persamaan 2.4. Nilai estimasi keadaan sistem pada waktu k atau estimasi *priori* diwakili oleh $\hat{x}_k^-$. F merupakan matriks transisi keadaan antar waktu. Untuk melibatkan pengaruh lingkungan eksternal, digunakan matriks kontrol B dan vektor kontrol u_{k-1} yang mewakili pengaruh yang diakibatkan lingkungan eksternal. P_k^- merupakan matriks kovarians nilai estimasi *priori* keadaan sistem di waktu k. Q merupakan matriks kovarians *noise* proses yang mewakili probabilitas dari lingkungan eksternal sistem.

$$\begin{aligned} K_k &= P_k^- H^T (HP_k^- H^T + R)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \\ P_k &= (I - K_k H)P_k^- \end{aligned} \tag{2.5}$$

Proses perhitungan *update* nilai internal terhadap hasil pengukuran ada pada Persamaan 2.5. K_k merupakan *Kalman gain* di waktu k. P_k^- merupakan matriks kovarians nilai estimasi *posterior* keadaan sistem di waktu k. H merupakan matriks model pengukuran. R merupakan matriks *noise* pengukuran. I merupakan matriks

identitas. Nilai pengukuran di waktu k diwakili oleh z_k yang digunakan dalam proses *update* nilai $\hat{x}$ posterior.

Proses perhitungan dalam *Kalman filter* terjadi dalam siklus antara perhitungan nilai terhadap waktu (nilai *priori*) dan perhitungan nilai terhadap pengukuran (nilai *posterior*). Saat pertama masuknya nilai pengukuran z_k, akan dihitung nilai K_k terkini. Kemudian, nilai pengukuran z_k dan *Kalman gain* K_k digunakan untuk menghitung nilai $\hat{x}_k$ *posterior*. Lalu, dihitung nilai kovarians P_k *posterior* dari K_k yang baru. Untuk memprediksi nilai $\hat{x}^-_{k+1}$ (nilai *priori*), digunakan hasil perhitungan $\hat{x}_k$. Kemudian dihitung juga nilai priori dari P^-_{k+1} berdasarkan nilai P^-_k.

2.6. Asosiasi Data Berbasis Batasan Struktural

Asosiasi data berbasis batasan struktural merupakan metode yang bertujuan untuk mengatasi kelemahan algoritma lain dalam melakukan asosiasi data pada data video yang tidak stabil dikarenakan pergerakan perangkat perekaman. Pada Persamaan 2.6, sebuah objek i di *frame* t dilambangkan dalam sebuah vektor s^i_t dengan (x^i_t, y^i_t) merupakan posisi objek dalam bidang 2D, $(\dot{x}^i_t, \dot{y}^i_t)$ merupakan kecepatan pada bidang 2D, dan (w^i_t, h^i_t) merupakan ukuran lebar dan tinggi objek. Himpunan objek di *frame* t didefinisikan sebagai $\mathcal{S}_t(s^t_i \in \mathcal{S}_t)$ dengan indeks $i \in \mathcal{I}_t \triangleq \{1, \ldots, N\}$. Sedangkan untuk deteksi di *frame* t didefinisikan dalam Persamaan 2.7 sebagai d^k_t. Himpunan deteksi di *frame* t didefinisikan sebagai $\mathcal{D}_t(d^k_t \in \mathcal{D}_t)$ dengan indeks $k \in \mathcal{K}_t \triangleq \{0, 1, \ldots, K\}$. Penggunaan indeks 0 ditujukan untuk alokasi deteksi bagi objek yang gagal dideteksi.

$$s^i_t = [x^i_t, y^i_t, \dot{x}^i_t, \dot{y}^i_t, w^i_t, h^i_t]^T \tag{2.6}$$

$$d^k_t = [x^k_t, y^k_t, w^k_t, h^k_t]^T \tag{2.7}$$

Batasan struktural yang digunakan dalam metode ini merupakan vektor $e^{i,j}_t$ seperti pada Persamaan 2.8 yang didapat dari seli-

sih vektor objek i dan j pada waktu t. Batasan struktural akan digunakan untuk mendapatkan posisi suatu objek relatif terhadap posisi objek lain sehingga dapat meminimalisasi dampak dari pergerakan perangkat perekaman video. Himpunan batasan struktural untuk objek i didefinisikan sebagai $\mathcal{E}_t^i = \{e_t^{i,j} | \forall j \in \mathcal{I}_t\}$ dan himpunan semua batasan struktural yang ada dalam *frame* t didefinisikan sebagai $\mathcal{E}_t = \{\mathcal{E}_t^i | \forall i \in \mathcal{I}_t\}$.

$$\begin{aligned} e_t^{i,j} &= [\chi_t^{i,j}, \xi_t^{i,j}, \dot{\chi}_t^{i,j}, \dot{\xi}_t^{i,j}]^T \\ &= [x_t^i - x_t^j, y_t^i - y_t^j, \dot{x}_t^i - \dot{x}_t^j, \dot{y}_t^i - \dot{y}_t^j]^T \end{aligned} \tag{2.8}$$

Permasalahan *multi object tracking* termasuk masalah asosiasi data dengan tujuan mendapatkan pasangan yang benar antara objek dan deteksi. Matriks pasangan asosiasi didefinisikan pada Persamaan 2.9. Pasangan objek i dan deteksi k diasosiasikan saat $a^{i,k}$ bernilai 1. Matriks asosiasi A harus menenuhi dua syarat, yaitu setiap deteksi diasosiasikan ke maksimal satu objek dan setiap objek diasosiasikan ke maksimal satu deteksi. Dalam asosiasi data, matriks asosiasi terbaik $\hat{A}$ dipilih dari matriks asosiasi dengan *cost* terkecil seperti pada Persamaan 2.10, dengan $a^{i,0}$ adalah kasus objek yang tidak terdeteksi sehingga jika semua objek tidak terdeteksi maka jumlah dari $a^{i,0}$ terhadap i sama dengan banyaknya objek $|I|$.

$$\begin{aligned} \mathcal{A} &= \{a^{i,k} | i \in \mathcal{I}, k \in \mathcal{K}\}, \\ a^{i,k} &= \{0.1\} \end{aligned} \tag{2.9}$$

$$\begin{aligned} &\hat{\mathcal{A}} = \arg\min_{\mathcal{A}} C(\mathcal{A}, \mathcal{S}, \mathcal{E}, \mathcal{D}), \\ &\text{s.t.} \sum_{\substack{i \in \mathcal{I} \\ k \neq 0}} a^{i,k} \leq 1 \wedge \sum_{k \in \mathcal{K}} a^{i,k} = 1 \wedge \sum_{i \in \mathcal{I}} a^{i,0} \leq |\mathcal{I}| \end{aligned} \tag{2.10}$$

Metode ini terdiri dari 3 tahap utama seperti dalam Pseudocode 2.2, yaitu *Structural Constraint Event Aggregation* (SCEA), *Structural Constraint Object Recovery* (SCOR), dan manajemen objek dan batasan struktural. $\mathcal{S}_w$ merupakan himpunan objek yang terlacak. $\mathcal{E}_w$ merupakan batasan struktural antar objek yang terlacak. $\mathcal{S}_m$ merupakan himpunan objek yang tidak terdeteksi. $\mathcal{E}_m$ merupakan batasan struktural antara objek terlacak dengan objek tidak terdeteksi. D merupakan himpunan deteksi.

Pseudocode 2.2: OBJECT TRACKING DENGAN SCEA DAN SCOR

Input: $\mathcal{S}_w, \mathcal{E}_w, \mathcal{S}_m, \mathcal{E}_m, \mathcal{D}$
Output: Trajectories of the targets

1: **for** video frame f **do**
2: $\mathcal{A}_w \leftarrow SCEA(\mathcal{S}_w, \mathcal{E}_w, \mathcal{D})$
3: $\mathcal{S}_w \leftarrow \{s_i = d^k | a^{i,k} = 1, \forall i \in \mathcal{I}_w, \forall k \in \mathcal{K}\}$
4: $\mathcal{A}_m \leftarrow SCOR(\mathcal{S}_m, \mathcal{E}_m, \mathcal{S}_w, \tilde{\mathcal{D}})$
5: $\mathcal{A} \leftarrow \mathcal{A}_w \cup \mathcal{A}_m$
6: Update $\mathcal{S}_w$ and $\mathcal{S}_m$ via object management
7: Update $\mathcal{E}_w$ and $\mathcal{E}_m$ via structural constraint update
8: $\mathcal{S}_w \leftarrow \{s^i | a^{i,k} = 1, \forall i \in \mathcal{I}_w \cup \mathcal{I}_m, \forall k \in \mathcal{K}\}$
9: **end for**

2.6.1. Structural Constraint Event Aggregation (SCEA)

Structural Constraint Event Aggregation (SCEA) merupakan tahap asosiasi antara himpunan objek yang sudah terlacak sebelumnya dengan himpunan deteksi di *frame* t. Dalam SCEA terdapat tiga tahap, yaitu pembuatan partisi, perhitungan *cost* dengan batasan struktural, dan agregat hasil asosiasi yang terdapat dalam Pseudocode 2.3.

Pseudocode 2.3: SCEA

Input: $\mathcal{S}_w, \mathcal{E}_w, \mathcal{D}$

Output: assignment matrix $\mathcal{A}_w$

1: $gated_mask \leftarrow GATING(\mathcal{S}_w, \mathcal{D})$

2: $\mathcal{S}_w^p, \mathcal{D}^p \leftarrow PARTITION(\mathcal{S}_w, \mathcal{D}); \mathcal{S}_w^P \subset \mathcal{S}_w, \mathcal{D}^P \subset \mathcal{D}$

3: $\mathcal{A}_{all}^p \leftarrow ASSIGNMENT_GENERATOR($
$\mathcal{S}_w^P, \mathcal{D}^p, gated_mask)$

4: $\mathcal{A}_w \leftarrow \emptyset$

5: **for** $p = 1 : P$ **do**
$C(\mathcal{A}^p, \mathcal{S}_w^p, \mathcal{E}_w, \mathcal{D}^p) \leftarrow$

6:
$$\frac{1}{\Delta} \sum_{\substack{i \in \mathcal{I}_w^P, k \in K^p \\ a^{i,k}=1}} \left(a^{i,k}\Omega_{i,k} + \sum_{\substack{j \in \mathcal{I}_w^p \\ j \neq i}} \sum_{\substack{q \in K^p \\ q \neq k}} a^{j,q}\Theta_{i,k}^{j,q} \right),$$
$$\Delta = \sum_{a^{i,k} \in \mathcal{A}^p} a^{i,k}$$

7: $\hat{A}^p \leftarrow \arg\min_{\mathcal{A}^p}(C(\mathcal{A}^p, \mathcal{S}_w^p, \mathcal{E}_w, \mathcal{D}^p)), \mathcal{A}^p \subset \mathcal{A}_{all}^p$

8: $\mathcal{A}_w \leftarrow \mathcal{A}_w \cup \hat{\mathcal{A}}^p$

9: **end for**

10: **return** $\mathcal{A}_w$

2.6.1.1. Pembuatan Partisi

Pada tahap pembuatan partisi dilakukan dua proses, yaitu *gating* dan pembagian menjadi partisi. Pada proses *gating*, akan dilakukan eliminasi matriks asosiasi $\mathcal{A}$ untuk asosiasi yang tidak memenuhi syarat pada Persamaan 2.11. p^i dan p_d^k masing-masing merupakan posisi objek i dan deteksi k. (w^i, h^i) merupakan lebar dan tinggi objek i. Nilai konstanta τ_s diisikan 0,7. Jika syarat terpenuhi, maka $a^{i,k}$ bernilai 1. Sebaliknya $a^{i,k}$ akan bernilai 0. Pada Gambar 2.4a dan 2.4b, terlihat bahwa ruang pencarian atau asosiasi yang memungkinkan yang diwakili daerah berwarna abu-abu men-

jadi semakin kecil setelah dilakukan proses *gating*. Kotak berwarna kuning menunjukkan deteksi di *frame* saat ini dan kotak berwarna yang lain menunjukkan objek yang sudah terlacak sebelumnya.

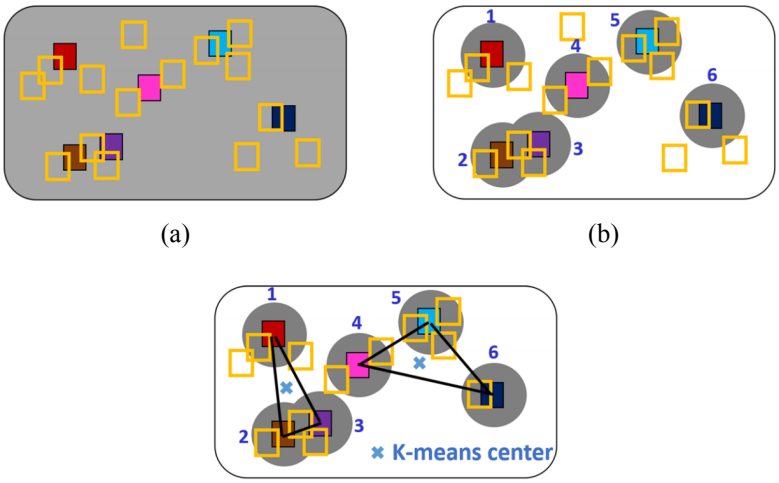
$$\left(\left\|p^i - p_d^k\right\| < \sqrt{(w^i)^2 + (h^i)^2}\right) \wedge \left(exp(-F_s(s^i, d^k) > \tau_s)\right) \tag{2.11}$$

Penyelesaian masalah asosiasi data dalam SCEA dilakukan dalam partisi-partisi kecil sehingga kompleksitasnya semakin rendah. Partisi dibuat berdasarkan titik pusat *cluster* hasil dari K-Means dengan masukan posisi objek. Banyaknya partisi yang dihasilkan adalah $P = \lceil \text{banyak objek}/5 \rceil$. Lalu, banyak anggota tiap partisi juga dibatasi maksimal 5 objek. Ilustrasi pembagian objek menjadi partisi terdapat pada Gambar 2.4c. Dari P partisi yang dihasilkan dibuat kombinasi matriks asosiasi dari tiap partisi yang memenuhi Persamaan 2.10 untuk kemudian dihitung *cost* dari tiap kombinasinya.

2.6.1.2. Perhitungan Cost dengan Batasan Struktural

Untuk mengatasi dampak pergerakan perangkat perekam, batasan struktural digunakan dalam perhitungan *cost* asosiasi objek dengan deteksi. Dari matriks asosiasi $\mathcal{A}$, dipilih satu asosiasi $a^{i,k}$ yang bernilai 1 untuk dijadikan *anchor*. *Anchor* digunakan sebagai referensi dalam menentukan posisi objek lain berdasarkan batasan struktural antara objek *anchor* dan objek lain.

Contoh penggunaan batasan struktural dengan objek *anchor* ada pada Gambar 2.5. Dalam contoh tersebut, asosiasi yang terjadi dalam matriks $\mathcal{A}_n$ adalah $a^{1.1} = a^{2.2} = a^{3.3} = 1$. Pada baris pertama, objek 1 diasosiasikan dengan deteksi 1 dan posisi objek 1 dipindahkan ke posisi deteksi 1. Kemudian posisi objek lain ditentukan berdasarkan batasan strukturalnya terhadap objek 1. Kemudian dapat dihitung *cost* asosiasi untuk *anchor* $a^{1.1} = 1$. Dengan



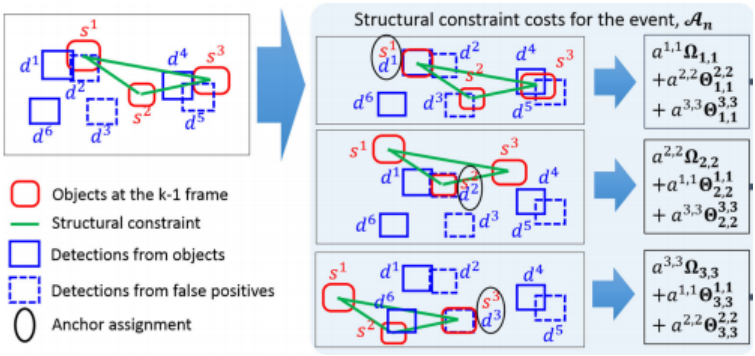
(a) (b)

(c)

Gambar 2.4 Ilustrasi Proses Pembuatan Partisi [3]: (a) Ruang Pencarian Awal, (b) Eliminasi dengan Gating, (c) Partisi dengan K-Means

mengganti *anchor* menjadi pada objek 2 dan 3 akan didapat 3 *cost* berbeda walaupun peristiwa asosiasinya sama.

Perhitungan *cost* berbasis batasan struktural didefinisikan pada Persamaan 2.12 dengan i dan k menujukkan indeks dari *cost* yang dihitung berdasarkan asosiasi $a^{i,k}$. *Anchor cost* $\Omega_{i,k}$ didefinisikan pada Persamaan 2.13. *Anchor cost* terdiri dari *cost* ukuran (F_s) dan *cost* penampilan (F_a) yang didefinisikan pada Persamaan 2.14. Dengan (w^i, h^i) dan (w_d^k, h_d^k) merupakan lebar dan tinggi dari objek i dan deteksi k. $p^n(s^i)$ dan $p^n(d^k)$ merupakan histogram.



Gambar 2.5 Contoh Penggunaan Batasan Struktural dalam Anchor [3]

b adalah indeks bin dan B adalah banyaknya bin.

$$C(\mathcal{A}^p, \mathcal{S}_w^p, \mathcal{E}_w, \mathcal{D}^p)|_{a^{i,k}} = a^{i,k}\Omega_{i,k} + \sum_{\substack{j\in\mathcal{I}_w \\ j\neq i}} \sum_{\substack{q\in K \\ q\neq k}} a^{j,q}\Theta_{i,k}^{j,q} \tag{2.12}$$

$$\Omega_{i,k} = F_s(s^i, d^k) + F_a(s^i, d^k) \tag{2.13}$$

$$\begin{aligned} F_s(s^i, d^k) &= -ln\left(1 - \frac{|h^i - h_d^k|}{2(h^i + h_d^k)} - \frac{|w^i - w_d^k|}{2(w^i + w_d^k)}\right) \\ F_a(s^i, d^k) &= -ln\sum_{b=1}^{B}\sqrt{p^b(s^i)p^b(d^k)} \end{aligned} \tag{2.14}$$

$$\Theta_{i,k}^{j,q} = \begin{cases} F_s(s^j, d^q) + F_a(s^j, d^q) + F_c(s^j, e^{j,i}, d^k, d^q) & \text{if } q \neq 0 \\ \tau & \text{if } q = 0 \end{cases} \tag{2.15}$$

$$F_c(s^j, e^{j,i}, d^k, d^q) = -ln\left(\frac{area\left(B(s^{j,k}) \cap B(d^q)\right)}{area\left(B(s^{j,k}) \cup B(d^q)\right)}\right) \qquad (2.16)$$
$$s^{j,k} = [x_d^k, y_d^k, 0, 0]^T + [\chi^{j,i}, \xi^{j,i}, w^j, h^j]^T$$

Berdasarkan objek *anchor*, dihitung *linked cost* berbasis batasan struktural seperti pada Persamaan 2.15. Nilai τ diatur bernilai 4 untuk kasus objek yang tidak terdeteksi d_0. Pada Persamaan 2.16, posisi objek j ditentukan berdasarkan posisi *anchor* (posisi deteksi k) dan batasan struktural $e^{j,i}$. Fungsi *cost* F_c dihitung menggunakan *overlap ratio* atau *intersection over union* (IoU) antara *bounding box* deteksi dan *ground truth*.

2.6.1.3. Agregat Hasil Asosiasi

Dari satu matriks asosiasi $\mathcal{A}$ didapat perhitungan *cost* untuk banyak *anchor* yang kemudian dilakukan agregasi sehingga didapatkan nilai *cost* akhir. Adanya proses agregasi dari berbagai *anchor cost* membantu mengurangi ambiguitas dari *false positive* objek yang dekat, dan deteksi yang keliru karena dilakukan perhitungan *cost* untuk satu matriks asosiasi $\mathcal{A}$ berulang kali sebanyak *anchor*.

$$C(\mathcal{A}, \mathcal{S}_w, \mathcal{E}_w, \mathcal{D}) = \frac{1}{\Delta} \sum_{i \in I} \sum_{\substack{k \in \mathcal{K} \\ a^{i,k}=1}} C(\mathcal{A}, \mathcal{S}_w, \mathcal{E}_w, \mathcal{D})|_{a^{i,k}},$$
$$\Delta = \sum_{i \in \mathcal{I}, k \in \mathcal{K}} a^{i,k} \qquad (2.17)$$

$$\mathcal{A}_w = \arg\min_{A} C(\mathcal{A}, \mathcal{S}_w, \mathcal{E}_w, \mathcal{D}), \mathcal{A} \subset \mathcal{A}_{all} \qquad (2.18)$$

Proses agregasi diformulasikan seperti pada Persamaan 2.17. Dengan Δ merupakan banyaknya *anchor* yang terpilih dari $\mathcal{A}$. Kemudian dipilih matriks asosiasi terbaik $\mathcal{A}_w$ dengan *cost* terkecil seperti pada Persamaan 2.18. Dari $\mathcal{A}_w$, nilai objek yang terdeteksi diperbarui dengan deteksi yang diasosiasikan $\hat{s}^i = d^k \text{if } a^{i,k} = 1$. Objek yang diperbarui ini akan digunakan pada proses *Structural Constraint Object Recovery* (SCOR).

2.6.2. Structural Constraint Object Recovery (SCOR)

Structural Constraint Object Recovery (SCOR) merupakan tahap setelah SCEA yang bertujuan untuk memulihkan objek yang hilang dari SCEA. Dari matriks asosiasi $\mathcal{A}_w$ dari SCEA, ada himpunan objek yang gagal diasosiasikan dan deteksi yang belum diasosiasikan dengan objek manapun. Matriks asosiasi hasil SCOR $\mathcal{A}_m$ didefinisikan pada Persamaan 2.19. $\mathcal{S}_m$ merupakan himpunan objek yang hilang. $\mathcal{S}_w$ merupakan himpunan objek yang terlacak di SCEA dan sudah diperbarui nilainya dengan deteksi yang diasosiasikan. $\mathcal{E}_m$ merupakan batasan struktural antara $\mathcal{S}_w$ dan $\mathcal{S}_m$. $\mathcal{I}_m$ merupakan indeks dari objek yang hilang. $\tilde{\mathcal{K}}$ merupakan indeks himpunan deteksi $\tilde{\mathcal{D}}$, dengan $\tilde{\mathcal{D}}$ merupakan himpunan yang belum diasosiasikan di SCEA ditambah dengan deteksi *dummy* d_0 untuk kasus objek salah deteksi.

$$\begin{aligned} &\mathcal{A}_m = \arg\min_{\mathcal{A}} C(\mathcal{S}_m, \mathcal{E}_m, \mathcal{S}_w, \tilde{\mathcal{D}}), \\ &s.t. \sum_{i \in \mathcal{I}_m} a^{i,q} = 1 \wedge \sum_{q \in \tilde{\mathcal{K}}} a^{i,q} = 1 \end{aligned} \tag{2.19}$$

Fungsi *cost* yang digunakan dalam SCOR didefinisikan pada Persamaan 2.20. τ bernilai positif dan diisikan dengan 4 pada program. Objek i yang hilang dipulihkan menggunakan batasan strukturalnya. *Constraint cost* didefinisikan pada Persamaan

2.21. s_1^{γ} merupakan nilai posisi objek yang diperbarui dari deteksi $[x_d^k, y_d^k], s_1^i \in \mathcal{S}_1^w$ if $a^{i,k} = 1$.

$$C(\mathcal{S}_m, \mathcal{E}_m, \mathcal{S}_w, \tilde{\mathcal{D}}) = \sum_{i \in \mathcal{I}_m} \sum_{q \in \tilde{\mathcal{K}}} a^{i,q} \Phi^{i,q}$$
$$\Phi^{i,q} = \begin{cases} F_s(s^i, d^q) + F_a(s^i, d^q) + F_r(s^i, \mathcal{E}_m, \mathcal{S}_w, d^q) & \text{if } q \neq 0 \\ \tau & \text{if } q = 0 \end{cases} \tag{2.20}$$

$$F_r(s^i, \mathcal{E}_m, \mathcal{S}_w, d^q) = -ln\left(\frac{area\left(B(s^{i,\gamma}) \cap B(d^q)\right)}{area\left(B(s^{i,\gamma}) \cup B(d^q)\right)}\right),$$
$$s^{i,\gamma} = [(s_1^{\gamma})^T, 0, 0] + [\chi^{i,\gamma}, \xi^{i,\gamma}, w^i, h^i]^T, \tag{2.21}$$
$$\gamma = \underset{j \in \mathcal{I}_w}{\arg\max} \frac{1}{\|[\dot{\chi}^{i,j}, \dot{\xi}^{i,j}]\|}$$

Untuk dapat menyelesaikan Persamaan 2.19, maka dirubah ke bentuk permasalahan *linear sum assignment* pada Persamaan 2.22. Nilai Φ^0 non-diagonal diisikan dengan ∞. Kemudian C diselesaikan dengan algoritma Hungarian untuk didapatkan asosiasi dengan *cost* terkecil.

$$C = \left[\Phi^{det}_{|N^m|\mathrm{x}|\bar{M}|} \Phi^0_{|N^m|\mathrm{x}|N^m|}\right],$$
$$\Phi^{det} = [\Phi^{i,q}], \forall i \in \mathcal{I}_m, \forall q \in \tilde{K}, \tag{2.22}$$
$$\Phi^0 = diag[\Phi^{i,0}], \forall i \in \mathcal{I}_m$$

2.6.3. Manajemen Objek dan Batasan Struktural

Berdasarkan matriks asosiasi akhir hasil penggabungan $\mathcal{A}_w$ dari SCEA dan $\mathcal{A}_m$ dari SCOR dilakukan pembaruan nilai objek dan batasan struktural. Dalam proses manajemen terdapat langkah

untuk memperbarui nilai, menghapus objek, dan membuat objek baru.

2.6.3.1. Manajemen Objek

Nilai objek yang diasosiasikan dengan deteksi tidak secara langsung diperbarui dengan nilai deteksi, tetapi melalui *Kalman filter* (KF). Proses pembaruan dengan Kalman dinyatakan dalam Persamaan 2.23. KF merupakan *Kalman filter* dengan model pergerakan F, matriks kovarians pergerakan Q, model pengukuran H, dan matriks kovarians *noise* pengukuran R.

$$\begin{aligned}\mathcal{S}_w = \{s^i = KF(s^i, d^k, F, Q, H, R) | a^{i,k} \\ = 1, \forall i \in \mathcal{I}_w \cup \mathcal{I}_m, \forall k \in \mathcal{K}\}\end{aligned} \tag{2.23}$$

Nilai masing-masing komponen *Kalman filter* ada pada Persamaan 2.24. I_n merupakan matriks identitas berukuran n x n dan 0_n merupakan matriks nol berukuran n x n. Dengan σ_q merupakan probabilitas perubahan kecepatan terhadap waktu dan σ_s merupakan probabilitas perubahan ukuran. Standar deviasi dari *noise* posisi dan ukuran deteksi dinyatakan dengan σ_x, σ_y, σ_w, dan σ_h. Dalam [3] nilai parameternya adalah $\sigma_q = 15$, $\sigma_x = \sigma_y = 3$, dan $\sigma_s = \sigma_w = \sigma_h = 15$.

Penampilan objek dalam histogramnya juga diperbarui seiring waktu. Jika histogram objek adalah $\mathcal{H}_{t-1}$ maka histogram akan diperbarui sesuai Persamaan 2.25. Dengan $\hat{\mathcal{H}}$ merupakan histogram objek dari deteksi di *frame* sekarang dan δ merupakan *learning rate* yang nilainya 0,1. Proses pembaruan histogram tidak diterapkan pada objek yang hilang. Selain itu, objek yang hilang juga tidak diperbarui nilai ukurannya.

Setelah pembaruan nilai objek, dilakukan eliminasi pada lintasan objek yang tidak diasosiasikan dengan deteksi selama beberapa *frame*. Pada [3], batasan yang digunakan adalah dua *frame* be-

runtun. Kemudian, dibuat juga objek baru untuk deteksi yang tidak diasosiasikan dengan objek manapun.

$$F = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \; H = [I_2,\ 0_2,\ I_2], \; R = diag(\sigma_x^2,\ \sigma_y^2,\ \sigma_w^2,\ \sigma_h^2)$$
$$Q = \begin{bmatrix} 0.25\sigma_q^2 & 0.5\sigma_q^2 & 0 & 0 & 0 & 0 \\ 0.25\sigma_q^2 & 0.5\sigma_q^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5\sigma_q^2 & \sigma_q^2 & 0 & 0 \\ 0 & 0 & 0.5\sigma_q^2 & \sigma_q^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_s^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_s^2 \end{bmatrix} \tag{2.24}$$

$$\mathcal{H}_t = (1 - \delta)\mathcal{H}_{t-1} + \delta\hat{\mathcal{H}} \tag{2.25}$$

2.6.3.2. Manajemen Batasan Struktural

Setelah dilakukan proses *tracking* pada objek, dilakukan pembaruan batasan struktural antar objek berdasarkan deteksi yang diasosiasikan. Pembaruan nilai menggunakan *Kalman filter* seperti pada Persamaan 2.26. Dengan $z^{i,j} = [x_d^i,\ y_d^i]^T - [x_d^j,\ y_d^j]^T$ sebagai masukan nilai pengukuran dan $[x_d^i,\ y_d^i]^T$ merupakan nilai posisi deteksi yang diasosiasikan ke objek i.

$$\begin{aligned} e^{i,j} &= KF(e^{i,j}, z^{i,j}, F_{sc}, Q_{sc}, H_{sc}, R_{sc}),\ \text{if}\ a^{i,k} \\ &= a^{j,k} = 1 \end{aligned} \tag{2.26}$$

Dari Persamaan 2.26, nilai model pergerakan F_{sc}, matriks kovarians pergerakan Q_{sc}, model pengukuran H_{sc}, dan matriks kova-

rians *noise* pengukuran R_{sc} dijelaskan pada Persamaan 2.27.

$$F = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} 0.25\sigma_{sc}^2 & 0.5\sigma_{sc}^2 & 0 & 0 \\ 0.25\sigma_{sc}^2 & 0.5\sigma_{sc}^2 & 0 & 0 \\ 0 & 0 & 0.5\sigma_{sc}^2 & \sigma_{sc}^2 \\ 0 & 0 & 0.5\sigma_{sc}^2 & \sigma_{sc}^2 \end{bmatrix},$$
$$H_{sc} = [I_2, 0_2], R_{sc} = diag(\sigma_x^2, \sigma_y^2) \tag{2.27}$$

Standar deviasi perubahan pergerakan σ_{sc} untuk batasan struktural menunjukkan kemungkinan perubahan terhadap waktu. Berbeda dengan pembaruan nilai objek, batasan struktural tidak terlalu dipengaruhi pergerakan kamera sehingga nilai σ_{sc} diisikan 1, sehingga memaksa perubahan batasan struktural untuk mengikuti model F_{sc} [3]. Untuk kovarians *noise* pengukuran R_{sc}, parameternya adalah $\sigma_x = \sigma_y = 3$ sama seperti pada pembaruan nilai objek. Khusus untuk batasan struktural dari pasangan objek yang salah satu objek atau keduanya hilang digunakan Persamaan 2.28.

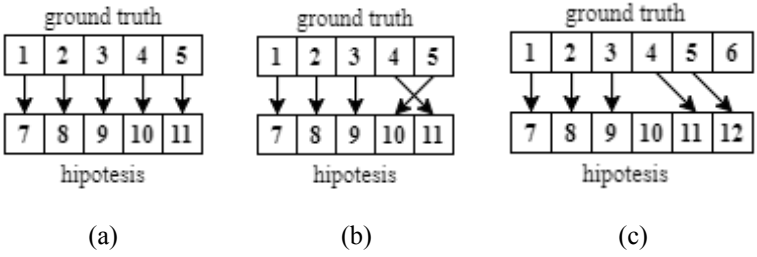
$$e_t^{i,j} = F_{sc} e_{t-1}^{i,j}, \text{ if } \neg(a^{i,k} = a^{j,k} = 1) \tag{2.28}$$

Ketika objek yang baru dibuat, nilai batasan strukturalnya dibuat sesuai Persamaan 2.8 dengan nilai $\dot{\chi}_t^{i,j} = \dot{\xi}_t^{i,j} = 0$. Jika suatu objek dieliminasi, maka semua batasan struktural yang dibuat dari objek tersebut akan dieliminasi juga.

2.7. Metrik dalam Multi Object Tracking

Metrik yang digunakan dalam pengukuran performa *multi object tracking* (MOT) secara umum mirip dengan metrik yang digunakan dalam *supervised learning* yaitu *false positive*, *false negative*, *precision*, dan *recall*. Namun, ada perbedaan definisi dari *false positive* dan *false negative* [13].

Informasi yang menjadi masukan dalam evaluasi MOT adalah ID *tracklet* dan data *bounding box* dari data *ground truth* dan hipotesis hasil *tracking*. Evaluasi dilakukan dengan mencoba memasangkan objek di *ground truth* dengan hipotesis yang ada. Penentuan pasangan menggunakan nilai IoU dari semua kombinasi pasangan yang ada lalu diselesaikan dengan *linear assignment*. Nilai IoU juga diberikan batasan nilai minimal yang biasanya diatur sebesar 0,5. Dari hasil pasangan yang didapat akan dikelompokkan menjadi beberapa jenis.



Gambar 2.6 Perbandingan *Ground Truth* dengan Hipotesis *Tracking*: (a) *Frame* t, (b) *Frame* $t + 1$, dan (c) *Frame* $t + 2$

Pada Gambar 2.6 terdapat ilustrasi perbandingan *ground truth* dengan hipotesis hasil *tracking*. Setiap kotak kecil mewakili ID objek yang ada di masing-masing sumber data. ID yang ada di *ground truth* dan hipotesis dapat memiliki format atau urutan yang berbeda. Panah pada gambar menunjukkan pasangan ID antara *ground truth* dan hipotesis dari hasil *linear assignment* proses evaluasi.

Pada Gambar 2.6a yang menampilkan *frame* t, diasumsikan pasangan yang didapat sama dengan yang ada pada *frame* $t - 1$ sehingga semua objek di *ground truth* masih dipasangkan dengan objek hipotesis yang sama. Pada *frame* t terhitung ada lima *match* dan merupakan kondisi ideal.

Pada Gambar 2.6b yang menampilkan *frame* $t+1$, dapat dilihat ada pasangan yang berubah dari yang terdapat pada *frame* t. Pada *frame* sebelumya, objek ID 4 menjadi pasangan dari hipotesis ID 10 tetapi sekarang dipasangkan dengan hipotesis ID 11. Hal yang sama terjadi juga pada objek ID 5 yang sebelumnya dipasangkan dengan hipotesis ID 11 sekarang dipasangan dengan ID 10. Yang terjadi pada objek ID 4 dan 5 merupakan *switch*. Untuk objek ID 1, 2, dan 3 termasuk *match*. Setelah dihitung sebagai *switch*, pasangan objek ID 4 dengan hipotesis ID 11 dianggap sebagai pasangan yang benar untuk *frame* selanjutnya, berlaku juga pada objek ID 5. Jadi, pada *frame* $t+1$ terdapat 3 *match* dan 2 *switch*.

Pada Gambar 2.6c yang menampilkan *frame* $t+2$, dapat dilihat ada pasangan yang berubah dari *frame* sebelumnya dan ada objek serta hipotesis baru yang muncul. Objek ID 1, 2, 3, dan 4 masih dipasangkan dengan hipotesis yang sama seperti pada *frame* $t+1$ sehingga dihitung sebagai 4 *match*. Objek ID 5 dihitung sebagai *switch* karena pasangan hipotesisnya berubah. Objek ID 6 yang tidak memiliki pasangan hipotesis dihitung sebagai *missing* atau *false negative* (FN). Hipotesis ID 10 yang tidak memiliki pasangan objek *ground truth* dihitung sebagai *false positive* (FP). Jadi ada 4 *match*, 1 *switch*, 1 FN, dan 1 FP.

Precision pada MOT didefinisikan pada Persamaan 2.29. $|detections|$ merupakan banyaknya deteksi yang dihitung dari total *match* dan *switch*. *Recall* pada MOT didefinisikan pada Persamaan 2.30. $|objects|$ merupakan total banyaknya objek yang ada di *ground truth* dari semua *frame*.

$$precision = \frac{|detections|}{|FP| + |detections|} \tag{2.29}$$

$$recall = \frac{|detections|}{|objects|} \tag{2.30}$$

Selain *precision* dan *recall*, di MOT juga terdapat metrik *Mul-*

tiple Object Tracker Accuracy (MOTA) yang didefinisikan pada Persamaan 2.31.

$$MOTA = 1 - \frac{FP + FN + switch}{|objects|} \tag{2.31}$$

Metrik *Multiple Object Tracking Precision* (MOTP) mengukur performa sistem MOT dalam prediksi posisi objek. Nilai MOTP didapat dari rata-rata nilai $1 - IoU$ dari pasangan objek di *ground truth* dan hipotesis. MOTP didefinisikan pada Persamaan 2.32 dengan d_t^i merupakan nilai IoU antara objek o_i dengan hipotesisnya pada *frame* t dan c_t merupakan banyaknya *match* ditambah *switch* pada *frame* t.

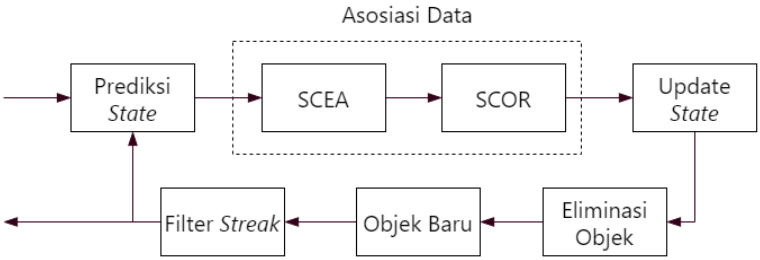
$$MOTP = 1 - \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{2.32}$$

BAB III
DESAIN

Pada bab ini akan dijelaskan desain data dan sistem *object tracking* yang menggunakan algoritma asosiasi data berbasis batasan struktural. Penjelasan dibagi menjadi dua bagian, yaitu Desain Umum Sistem dan Desain Program Utama.

3.1. Desain Umum Sistem

Sistem akan menerima masukan *sequence* yang menunjukkan nama rangkaian *frame* yang akan diproses. Kemudian, data *bounding box* deteksi dan citra tiap *frame* menjadi masukan dari proses *object tracking* menggunakan *Structural Constraint Event Aggregation* dan *Structural Constraint Object Recovery* yang telah dibahas pada subbab 2.6.1 dan 2.6.2. Lalu, keluaran yang didapat berupa label ID tiap *track* yang digunakan untuk proses evaluasi. Diagram alir sistem secara umum ditunjukkan pada Gambar 3.1.



Gambar 3.1 Diagram Alir Sistem

3.2. Desain Program Utama

Pada subbab ini akan dijelaskan desain fungsi dalam program yang digunakan untuk *object tracking*.

3.2.1. Desain Fungsi Main

Fungsi akan memuat data anotasi *bounding box* dan citra yang sesuai dengan rangkaian *frame* yang diberikan ke memori. Dari iterasi tiap *frame*, dilakukan *tracking* sesuai *bounding box* di *frame* tersebut. Kemudian dicetak nilai semua objek yang sudah melalui *tracking*. Masukan, proses, dan keluaran dari fungsi ini terdapat pada Tabel 3.1. Desain fungsi ini terdapat pada Pseudocode 3.1.

Tabel 3.1 Masukan, Proses, dan Keluaran dari Fungsi MAIN

Masukan	Proses	Keluaran
String $sequencename$ yang merupakan nama dari rangkaian *frame* yang akan	Menjalankan proses *object tracking* dan mencetak hasil datanya	-

Pseudocode 3.1: Fungsi MAIN

Input: $sequencename$

```
1:  annotation ← LOADCSV(sequencename)
2:  images ← LOADIMAGES(sequencename)
3:  for each (frame_num, bboxes) ∈ annotation do
4:      detections ← ARRAY()
5:      image ← images[frame_num]
6:      for each bbox ∈ bboxes do
7:          detection ← NEWDETECTION(bbox, image)
8:          insert detection into detections
9:      end for
10:     tracked ← OBJECTTRACKER(detections, frame_num)
11:     PRINT(tracked)
12: end for
```

3.2.2. Desain Fungsi Object Tracking

Data yang menjadi masukan *object tracking* berupa *bounding box* objek yang ada dalam satu *frame* dan citra satu *frame*. Semua objek dan batasan struktural diprediksi nilainya untuk *frame* saat ini. Deteksi diproses fungsi SCEA agar didapat asosiasi dengan objek dari *frame* sebelumnya. Dari SCEA didapat kumpulan objek yang dapat diasosiasikan dengan deteksi, kumpulan objek yang belum diasosiasikan, dan kumpulan deteksi yang belum diasosiasikan. Ketiga keluaran dari SCEA digunakan sebagai masukan bagi fungsi SCOR.

Pada fungsi SCOR, objek dan deteksi yang belum diasosiasikan di SCEA akan dicoba untuk diasosiasi. Kemudian didapat hasil kumpulan objek yang dapat diasosiasikan dengan deteksi, kumpulan objek yang belum diasosiasikan, dan kumpulan deteksi yang belum diasosiasikan dengan objek.

Objek diperbarui nilainya berdasarkan hasil asosiasi. Kemudian, objek yang hilang dieliminasi. Deteksi yang belum diasosiasikan menjadi objek baru. Kemudian, objek diseleksi dari lama terlacak secara beruntun. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.2. Desain fungsi terdapat pada Pseudocode 3.2.

Tabel 3.2 Masukan, Proses, dan Keluaran dari Fungsi OBJECTTRACKER

Masukan	Proses	Keluaran
Array $detections$ yang merupakan kumpulan deteksi dan bilangan bulat $frame_num$ yang merupakan ID *frame*	Melakukan rangkaian proses *object tracking* berdasarkan SCEA dan SCOR	*Array* $objects$ yang merupakan kumpulan objek hasil *object tracking*

Pseudocode 3.2: Fungsi OBJECTTRACKER

Input: $detections$, $frame_num$
Output: $objects$

$\mathcal{S}$, $\mathcal{E} \leftarrow$ PREDICTOBJECTS($\mathcal{S}$), PREDICTSC($\mathcal{E}$)
$\mathcal{S}_w^{SCEA}$, $\mathcal{D}_w^{SCEA}$, $\mathcal{D}_m^{SCEA} \leftarrow$ SCEA($\mathcal{S}_w$, $detections$, $\mathcal{E}_w$)
$\mathcal{S}_w^{SCOR}$, $\mathcal{D}_w^{SCOR}$, $\mathcal{D}_m^{SCOR} \leftarrow$SCOR(
    $\mathcal{S}_m$, $\mathcal{E}_m$, $\mathcal{S}_w^{SCEA}$, $\mathcal{D}_m^{SCEA}$)
$\mathcal{S}_w \leftarrow$ CONCAT($\mathcal{S}_w^{SCEA}$, $\mathcal{S}_w^{SCOR}$)
$\mathcal{D}_w \leftarrow$ CONCAT($\mathcal{D}_w^{SCEA}$, $\mathcal{D}_w^{SCOR}$)
$\mathcal{S}_w$, $\mathcal{E} \leftarrow$ UPDATEOBJECT($\mathcal{S}_w$, $\mathcal{D}_w$), UPDATESC($\mathcal{S}$, $\mathcal{E}$)
$\mathcal{S} \leftarrow$ ELIMINATE($\mathcal{S}$, $frame_num$)
$\mathcal{S}_{new}, \mathcal{E}_{new} \leftarrow$ NEWOBJECTS($\mathcal{D}_m^{SCOR}$)
$objects \leftarrow$ CONCAT($\mathcal{S}$, $\mathcal{S}_{new}$)
$objects \leftarrow$ FILTERSTREAK($objects$, min_streak)
return $objects$

3.2.3. Desain Fungsi SCEA

Dalam fungsi SCEA dilakukan asosiasi antara objek yang telah ada sebelumnya dengan deteksi dari *frame* saat ini. Matriks asosiasi yang dibuat memiliki dimensi MxN untuk M objek dan N deteksi. Dari M objek tersebut akan dibagi menjadi partisi sesuai batas maksimal anggota per partisi yang ditentukan. Partisi dibuat menggunakan hasil *cluster* dari K-Means. Kemudian dari semua kemungkinan asosiasi yang ada di matriks asosiasi MxN, diproses di fungsi GATING untuk mengeliminasi asosiasi dan memperkecil ruang pencarian.

Matriks asosiasi hasil fungsi GATING dan partisi objek dimasukkan ke dalam perhitungan asosiasi terbaik yang dihitung per partisi. Setelah didapat matriks asosiasi per partisi, dilakukan penggabungan matriks asosiasi menjadi hasil akhir dari SCEA. Masukan, proses, dan keluaran dari fungsi ini terdapat pada Tabel 3.3. Desain fungsi ini terdapat pada Pseudocode 3.3.

Tabel 3.3 Masukan, Proses, dan Keluaran dari Fungsi SCEA

Masukan	Proses	Keluaran
Array $\mathcal{S}_w$ yang merupakan kumpulan objek yang terlacak, *array* $\mathcal{D}$ yang merupakan kumpulan deteksi, *array* 2D $\mathcal{E}_w$ yang merupakan batasan struktural antar objek terlacak	Melakukan asosiasi antara objek terlacak dengan deteksi di *frame* saat ini	*array* $\mathcal{S}_w$ yang merupakan kumpulan objek terlacak dan sudah diperbarui, *array* $\mathcal{D}_w$ yang merupakan kumpulan deteksi yang sudah terasosiasi, *array* $\mathcal{D}_m$ yang merupakan kumpulan deteksi yang belum terasosiasi

Pseudocode 3.3: Fungsi SCEA

Input: $\mathcal{S}_w,\ \mathcal{D},\ \mathcal{E}_w$
Output: $\mathcal{S}_w,\ \mathcal{D}_w,\ \mathcal{D}_m$

```
mask ← GATING(S_w, detections)
subgroups ← SUBGROUPBYCLUSTER(S_w)
matrices, costs ← ARRAY(), ARRAY()
for each subgroup ∈ subgroups do
    matrix, cost ←ASSIGNMENTSUBGROUP(
                        mask, subgroup, S_w, D, E_w)
    insert matrix into matrices
    insert cost into costs
end for
best_matrix ← AGGREGATE(matrices, costs)
S_w, D_w, D_m ← best_matrix
return S_w, D_w, D_m
```

3.2.4. Desain Fungsi Gating

Gating bertujuan untuk mengeliminasi kandidat asosiasi objek dan deteksi yang tidak memenuhi batasan yang diberikan dan mengurangi besar ruang pencarian. Batasan atau syarat yang digunakan sesuai Persamaan 2.11 dengan nilai τ_s sebesar 0,7. Perhitungan dilakukan dalam matriks berdimensi MxN untuk M objek dan N deteksi untuk syarat jarak titik pusat dibanding diagonal dan nilai *cost* F_s. Hasil akhir berupa matriks *mask boolean* berdimensi MxN. Masukan, proses, dan keluaran dari fungsi ini terdapat pada Tabel 3.4. Desain fungsi ini terdapat pada Pseudocode 3.4.

Tabel 3.4 Masukan, Proses, dan Keluaran dari Fungsi GATING

Masukan	Proses	Keluaran
Array $objects$ yang merupakan kumpulan objek, *array* $detections$ yang merupakan kumpulan deteksi	Membuat matriks *masking* untuk eliminasi asosiasi	*array* 2D $mask$ yang merupakan *boolean mask* untuk matriks asosiasi

Pseudocode 3.4: Fungsi GATING

Input: $objects, detections$
Output: $mask$

1: $distance_matrix \leftarrow$ CENTERDIST($objects,\ detections$)
2: $diagonal_matrix \leftarrow$ DIAGONALDIST($objects$)
3: $fs_matrix \leftarrow$ FS($objects, detections$)
4: $diag_mask \leftarrow (distance_matrix < diagonal_matrix)$
5: $fs_mask \leftarrow$ EXP($-fs_matrix$) $> \tau_s$)
6: $mask \leftarrow$ LOGICALAND($diag_mask,\ fs_mask$)
7: **return** $mask$

3.2.5. Desain Fungsi Partisi

Partisi dilakukan pada objek yang telah dilacak. Partisi dibuat berdasarkan *centroid* hasil *clustering* K-Means. Masukan dari K-Means adalah posisi (x, y) dari semua objek. Kemudian didapat *cluster* yang mewakili kelompok objek berdasarkan aspek spasial. Maksimal banyak anggota per *cluster* diatur sebesar 5. Berdasarkan *cluster*, objek dikelompokkan ke dalam partisi yang akan digunakan dalam proses SCEA. Masukan, proses, dan keluaran dari fungsi ini terdapat pada Tabel 3.5. Desain fungsi ini terdapat pada Pseudocode 3.5a dan 3.5b.

Tabel 3.5 Masukan, Proses, dan Keluaran dari Fungsi SUBGROUPBYCLUSTER

Masukan	Proses	Keluaran
Array objects yang merupakan kumpulan objek	Membuat partisi berdasarkan hasil K-Means dari posisi objek	*array partitions* yang merupakan kumpulan indeks objek yang menjadi anggota tiap partisi

Pseudocode 3.5a: Fungsi SUBGROUPBYCLUSTER (1)

Input: $objects$
Output: $partitions$
1: $num_object \leftarrow$ LEN($objects$)
2: $num_cluster \leftarrow$ CEIL($num_object/5$)
3: $positions \leftarrow$ ARRAY()
4: **for each** $object \in objects$ **do**
5: insert $object$'s (x, y) to $positions$
6: **end for**

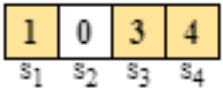
Pseudocode 3.5b: Fungsi SUBGROUPBYCLUSTER (2)

```
 7: cluster_labels ← KMEANS(positions)
 8: partitions ← ARRAY()
 9: for label ∈ UNIQUE(cluster_labels) do
10:     partition ← INDEX(cluster_labels = label)
11:     insert partition to partitions
12: end for
13: return partitions
```

3.2.6. Desain Fungsi Perhitungan Asosiasi Per Partisi

Pemilihan asosiasi terbaik dilakukan per partisi. Matriks asosiasi yang utama dieliminasi terlebih dahulu dengan hasil *mask* dari fungsi GATING. Kemudian dipilih bagian matriks asosiasi yang mengandung objek dalam partisi sehingga didapat matriks asosiasi partisi. Dari matriks asosiasi partisi dilakukan proses pembangkitan semua kombinasi asosiasi objek dan deteksi.

Perhitungan *cost* asosiasi dilakukan dengan menghitung *cost* dari penggunaan tiap objek sebagai *anchor* yang kemudian dihitung rata-ratanya. Objek dapat dijadikan sebagai anchor jika deteksi yang diasosiasikan bukan d_0. Seperti contoh kombinasi asosiasi pada Gambar 3.2, dari empat objek yang ada, objek s_2 tidak dapat dijadikan *anchor* karena menjadi d_0, sehingga total terdapat tiga *anchor* dalam contoh tersebut. Ketika satu objek dijadikan *anchor*, dapat dihitung *cost* dari objek *anchor* dan objek lain yang bukan *anchor*. Untuk objek non *anchor* yang diasosiasikan dengan d_0 maka nilai *cost*-nya diatur sebesar 4,0. Kemudian, *cost* dari semua anchor dihitung rata-ratanya dan menjadi *cost* untuk satu kombinasi asosiasi. Dari semua kombinasi dipilih yang *cost*-nya paling kecil untuk menjadi asosiasi terbaik dari partisi yang diberikan. Masukan, proses, dan keluaran dari fungsi ini terdapat pada Tabel 3.6. Desain fungsi ini terdapat pada Pseudocode 3.6a dan 3.6b.



Gambar 3.2 Ilustrasi Pemilihan Objek yang Menjadi *Anchor* dalam Kombinasi Asosiasi

Tabel 3.6 Masukan, Proses, dan Keluaran dari Fungsi ASSIGNMENTBYSUBGROUP

Masukan	Proses	Keluaran
Array $partition$ yang merupakan daftar objek dalam partisi, *array* 2D $matrix$ yang merupakan matriks asosiasi partisi, *array* $\mathcal{D}$ yang merupakan kumpulan deteksi, *array* $\mathcal{E}$ yang merupakan kumpulan batasan struktural	Mendapatkan asosiasi untuk partisi dengan *cost* terkecil	*Array* 2D $best_matrix$ yang merupakan matriks asosiasi objek dan deteksi terbaik untuk partisi tersebut dan bilangan real $best_cost$ yang merupakan *cost* dari asosiasi di $best_matrix$

Pseudocode 3.6a: Fungsi ASSIGNMENTBYSUBGROUP (1)

Input: $partition,\ matrix, \mathcal{D},\ \mathcal{E}$
Output: $best_matrix, best_cost$
1: $best_cost \leftarrow \infty^+$
2: $best_combination \leftarrow$ ARRAY()

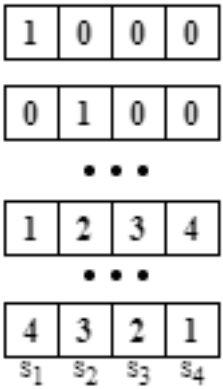
Pseudocode 3.6b: Fungsi ASSIGNMENTBYSUBGROUP (2)

```
3: for combination ∈ ASSIGNMENTGENERATOR(matrix) do
4:     n_anchor ← 0
5:     cost ← 0.0
6:     for object, detection ∈ combination do
7:         if detection is d_0 then
8:             continue
9:         end if
10:        n_anchor ← n_anchor + 1
11:        cost ←cost + FS(object, detection)+
                   FA(object, detection)
12:        for obj, det ∈ combination do
13:            if obj = object then
14:                continue
15:            end if
16:            if det is d_0 then
17:                cost ← cost + cost_d_0
18:            else
19:                cost ←cost + FS(obj, det) + FA(obj, det)+
                          FC(obj, det, object, sc)
20:            end if
21:        end for
22:    end for
23:    cost ← cost/n_anchor
24:    update best_cost and best_combination
25: end for
26: best_matrix ← ASMATRIX(best_combination)
27: return best_matrix, best_cost
```

3.2.7. Desan Fungsi Pembangkit Kombinasi Asosiasi

Pembangkitan kombinasi asosiasi objek dan deteksi diawali dengan mendaftar semua deteksi yang menjadi kandidat asosiasi

untuk tiap objek dengan tambahan d_0 untuk kasus gagal dideteksi. Dari daftar kandidat deteksi untuk semua objek dibuat produk *cartesian* yang menghasilkan kombinasi asosiasi objek dan deteksi. Dalam hasil produk *cartesian* masih terdapat kombinasi yang berupa duplikat karena satu deteksi diisikan ke beberapa objek kecuali untuk kasus d_0 yang boleh diisikan pada beberapa objek sekaligus. Hasil akhir kombinasi digunakan untuk perhitungan *cost* asosiasi masing-masing. Contoh kombinasi yang dihasilkan terdapat pada Gambar 3.3. Setiap kotak mewakili objek, yang pada contoh tersebut terdiri dari s_1 hingga s_4. Isi dari tiap kotak menunjukkan deteksi yang diasosiasikan, termasuk bilangan nol untuk kasus d_0. Pada contoh baris pertama, objek s_1 diasosiasikan dengan deteksi 1 dan objek lain menjadi d_0. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.7. Desain fungsi terdapat pada Pseudocode 3.7.



Gambar 3.3 Ilustrasi Kombinasi Asosiasi antara Objek dan Deteksi

Tabel 3.7 Masukan, Proses, dan Keluaran dari Fungsi ASSIGNMENTGENERATOR

Masukan	Proses	Keluaran
Array 2D $matrix$ yang merupakan matriks asosiasi dari satu partisi	Membangkitkan semua kombinasi asosiasi antara objek dan deteksi	*Array* $combinations$ yang merupakan kumpulan kombinasi asosiasi objek dan deteksi

Pseudocode 3.7: Fungsi ASSIGNMENTGENERATOR

Input: $matrix$
Output: $combinations$

```
1: detection_candidates ← ARRAY()
2: for each row ∈ matrix do
3:     detection_indices ← NONZERO(row)
4:     insert detection_indices into detection_candidates
5: end for
6: combinations ← CARTESIAN(detection_candidates)
7: combinations ← UNIQUE(combinations, exclude = 0)
8: return combinations
```

3.2.8. Desain Fungsi Cost Fs

Fungsi ini merupakan implementasi dari Persamaan 2.14 yang menghitung *cost* komponen dimensi dari objek dan deteksi. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.8. Desain fungsi terdapat pada Pseudocode 3.8.

Tabel 3.8 Masukan, Proses, dan Keluaran dari Fungsi FS

Masukan	Proses	Keluaran
$object$ yang merupakan objek dan $detection$ yang merupakan deteksi	Menghitung *cost* F_s sesuai persamaan	Bilangan real $cost$ yang merupakan hasil perhitungan *cost* F_s

Pseudocode 3.8: Fungsi FS

Input: $object$, $detection$
Output: $cost$

1: h_obj, $w_obj \leftarrow object.height$, $object.width$
2: h_det, $w_det \leftarrow detection.height$, $detection.width$
3: $h \leftarrow \text{ABS}(h_obj - h_det)/(2 * (h_obj + h_det))$
4: $w \leftarrow \text{ABS}(w_obj - w_det)/(2 * (w_obj + w_det))$
5: $cost \leftarrow -1 * \text{LOG}(1 - h - w)$
6: **return** $cost$

3.2.9. Desain Fungsi Cost Fa

Fungsi ini merupakan implementasi dari Persamaan 2.14 yang menghitung *cost* komponen penampilan dari histogram. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.9. Desain fungsi terdapat pada Pseudocode 3.9.

Tabel 3.9 Masukan, Proses, dan Keluaran dari Fungsi FA

Masukan	Proses	Keluaran
$object$ yang merupakan objek dan $detection$ yang merupakan deteksi	Menghitung *cost* F_a sesuai persamaan	Bilangan real $cost$ yang merupakan hasil perhitungan *cost* F_a

Pseudocode 3.9: Fungsi FA

Input: $object$, $detection$
Output: $cost$

1: $hist_obj \leftarrow object.histogram$
2: $hist_det \leftarrow detection.histogram$
3: $mul \leftarrow \text{DOTMUL}(hist_obj,\ hist_det)$
4: $root \leftarrow \text{SQRT}(mul)$
5: $summation \leftarrow \text{SUM}(root)$
6: $cost \leftarrow \text{LOG}(summation)$
7: **return** $cost$

3.2.10. Desain Fungsi Cost Fc

Fungsi ini merupakan implementasi dari Persamaan 2.16 yang menghitung *cost* komponen spasial dengan modifikasi penghapusan *log* sehingga keluaran dalam rentang $[0,\ 1]$.

Gambar 3.4 Ilustrasi Penggunaan Batasan Struktural dalam Menentukan Posisi di Perhitungan IoU: (a) Kondisi Awal dan (b) Kondisi Setelah Pergeseran Berdasarkan Batasan Struktural

Fungsi *cost* F_c merupakan fungsi yang mengintegrasikan batasan struktural dalam perhitungan *cost* asosiasi seperti pada Gambar 3.4. Pada Gambar 3.4a terdapat dua objek s_1 dan s_2 berupa

kotak bergaris hitam, dua deteksi d_1 dand_2 berupa kotak bergaris putus-putus hijau, dan batasan struktural $e_{1,2}$ antara s_1 dan s_2. Pada contoh tersebut, yang menjadi *anchor* adalah objek s_1 dengan d_1, sehingga yang menjadi masukan det_k adalah d_1. Posisi relatif s_2 terhadap d_1 didapat dari posisi d_1 dan batasan struktural $e_{1,2}$, kemudian bisa dihitung IoU dengan d_2 yang merupakan masukan det_q. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.10. Desain fungsi terdapat pada Pseudocode 3.10.

Tabel 3.10 Masukan, Proses, dan Keluaran dari Fungsi FC

Masukan	Proses	Keluaran
$object$ berupa objek, det_q berupa deteksi, det_k berupa deteksi yang diasosiasikan dengan objek *anchor*, sc berupa batasan struktural antara $object$ dan objek yang menjadi *anchor*	Menghitung *cost* F_c sesuai persamaan	Bilangan real $cost$ yang merupakan hasil perhitungan *cost* F_c

Pseudocode 3.10: Fungsi FC

Input: $object$, det_k, det_q, sc
Output: $cost$

1: $moved \leftarrow [det_k.x + sc.delta_x,\ det_k.y + sc.delta_y,\ object.width,\ object.height]$
2: $iou_score \leftarrow \text{IOU}(moved,\ det_q)$
3: $cost \leftarrow 1 - iou_score$
4: **return** $cost$

3.2.11. Desain Fungsi Penggabungan Matriks Asosiasi

Walaupun objek yang ada di dalam tiap partisi berbeda-beda, deteksi dapat berada di lebih dari satu partisi sesuai hasil *gating* di awal. Oleh karena itu, dilakukan pengecekan dan perbandingan *cost* antar matriks asosiasi tiap partisi, sehingga jika ditemukan deteksi yang diisikan pada lebih dari satu objek akan dipilih yang *cost* asosiasi partisinya lebih kecil. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.11. Desain fungsi terdapat pada Pseudocode 3.11a dan 3.11b.

Tabel 3.11 Masukan, Proses, dan Keluaran dari Fungsi AGGREGATE

Masukan	Proses	Keluaran
Array 3D $matrices$ yang merupakan kumpulan matriks asosiasi dari semua partisi	Menggabungkan matriks asosiasi dari semua partisi dan memastikan asosiasi yang dipilih dengan *cost* terkecil	*Array* 2D $matrix$ yang merupakan matriks asosiasi hasil penggabungan

Pseudocode 3.11a: Fungsi AGGREGATE (1)

Input: $matrices$
Output: $matrix$

1: $matrix \leftarrow$ 2D ARRAY() set to $False$
2: $cost_matrix \leftarrow$ 2D ARRAY() set to ∞
3: **for** $mat,\ local_cost \in matrices$ **do**
4: $\quad mask \leftarrow (mat = True)$
5: $\quad mask_local \leftarrow$ SUM$(mat,\ axis = 0)$

Pseudocode 3.11b: Fungsi AGGREGATE (2)

```
    mask_global ← SUM(matrix, axis = 0)
    intersection ← INTERSECT(mask_local, mask_global)
    for each col ∈ (intersection = True) do
        global_cost ← MIN(cost_matrix[:, col])
        global_cost_row ← ARGMIN(cost_matrix[:, col])
        if global_cost < local_cost then
            mask[:, col] ← False
        else
            matrix[global_cost_row, col] ← False
        end if
    end for
    matrix[mask] ← True
    cost_matrix[mask] ← local_cost
end for
return matrix
```

3.2.12. Desain Fungsi SCOR

Dalam proses SCOR dilakukan proses asosiasi antara objek yang menghilang atau gagal diasosiasikan di SCEA dan deteksi yang belum diasosiasikan. Masukan yang diperlukan oleh SCOR ada tiga, yaitu objek yang menghilang, deteksi yang belum diasosiasikan, dan objek yang sudah diperbarui di SCEA. Proses SCOR dibagi menjadi tiga tahap. Pertama, penentuan objek dari kumpulan objek yang sudah diperbarui di SCEA yang digunakan sebagai pasangan batasan struktural bagi tiap objek yang menghilang berdasarkan kemiripan pergerakan objek. Kemudian dihitung matriks *cost* untuk asosiasi objek dan deteksi. Terakhir, dilakukan penyelesaian *linear sum assignment* pada matriks *cost* untuk mendapatkan pasangan objek dan deteksi yang diasosiasikan. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.12. Desain fungsi terdapat pada Pseudocode 3.12.

Tabel 3.12 Masukan, Proses, dan Keluaran dari Fungsi SCOR

Masukan	Proses	Keluaran
Array $\mathcal{S}_m$ yang merupakan objek yang hilang, *array* $\hat{\mathcal{D}}$ yang merupakan deteksi belum diasosiasikan, *array* $\mathcal{S}_w$ yang merupakan objek yang sudah dilacak dan diperbarui di SCEA	Melakukan asosiasi antara objek $\mathcal{S}_m$ dengan deteksi $\hat{\mathcal{D}}$ di *frame* saat ini	*array* $\mathcal{S}_w$ yang merupakan kumpulan objek terlacak dan sudah diperbarui, *array* $\mathcal{D}_w$ yang merupakan kumpulan deteksi yang sudah terasosiasi, *array* $\mathcal{D}_m$ yang merupakan kumpulan deteksi yang belum terasosiasi

Pseudocode 3.12: Fungsi SCOR

Input: $\mathcal{S}_m$, $\hat{\mathcal{D}}$, $\mathcal{S}_w$
Output: $\mathcal{S}_w$, $\mathcal{D}_w$, $\mathcal{D}_m$
1: $\mathcal{S}_\gamma \leftarrow$ GETGAMMAOBJECTS($\mathcal{S}_m$, $\mathcal{S}_w$)
2: $cost_matrix \leftarrow$ CALCASSIGNMENTCOST($\mathcal{S}_m$, $\hat{\mathcal{D}}$, $\mathcal{S}_\gamma$)
3: $best_matrix \leftarrow$ ASSIGNMENT($cost_matrix$)
4: $\mathcal{S}_w$, $\mathcal{D}_w$, $\mathcal{D}_m \leftarrow best_matrix$
5: **return** $\mathcal{S}_w$, $\mathcal{D}_w$, $\mathcal{D}_m$

3.2.13. Desain Fungsi Penentuan Objek Gamma

Objek *gamma* digunakan sebagai pasangan batasan struktural bagi objek yang menghilang dan dipilih dari S_w berdasarkan kemiripan pergerakan seperti pada Persamaan 2.21. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.13. Desain fungsi terdapat pada Pseudocode 3.13a dan 3.13b.

Tabel 3.13 Masukan, Proses, dan Keluaran dari Fungsi GETGAMMAOBJECTS

Masukan	Proses	Keluaran
Array $missing_objects$ yang merupakan kumpulan objek menghilang, *array* $updated_objects$ yang merupakan kumpulan objek diperbarui di SCEA	Mendapatkan pasangan objek *gamma* untuk setiap objek yang menghilang	*array* $gamma_objects$ yang merupakan kumpulan pasangan objek *gamma* dari $missing_objects$

Pseudocode 3.13a: Fungsi GETGAMMAOBJECTS (1)

Input: $missing_objects$, $updated_objects$
Output: $gamma_objects$

```
1:  gamma_objects ← ARRAY()
2:  for each missing ∈ missing_objects do
3:      min_resultant ← ∞
4:      min_object ← None
5:      for each updated ∈ updated_objects do
6:          delta_vx ← missing.vx − updated.vx
7:          delta_vy ← missing.vy − updated.vy
8:          resultant ← delta_vx^2 + delta_vy^2
9:          if resultant < min_resultant then
10:             min_resultant ← resultant
11:             min_object ← updated
12:         end if
13:     end for
14:     insert min_object to gamma_objects
```

Pseudocode 3.13b: Fungsi GETGAMMAOBJECTS (2)

```
15: end for
16: return gamma_objects
```

3.2.14. Desain Fungsi Perhitungan Matriks Cost

Perhitungan sesuai dengan Persamaan 2.21 dan 2.20. Matriks *cost* yang dihasilkan berukuran MxN untuk M objek dan N deteksi . Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.14. Desain fungsi terdapat pada Pseudocode 3.14a dan 3.14b.

Tabel 3.14 Masukan, Proses, dan Keluaran dari Fungsi CALCASSIGNMENTCOST

Masukan	Proses	Keluaran
Array $\mathcal{S}_m$ yang merupakan kumpulan objek menghilang, *array* $\mathcal{S}_\gamma$ yang merupakan kumpulan objek gamma, *array* $detections$ yang merupakan kumpulan deteksi	Menghitung matriks *cost* asosiasi objek dan deteksi	*Array* 2D $cost_matrix$ yang merupakan matriks *cost* asosiasi

Pseudocode 3.14a: Fungsi CALCASSIGNMENTCOST (1)

```
Input: S_m, S_γ, detections
Output: cost_matrix
1: cost_matrix ← ARRAY()
2: for each (s_m, s_γ) ∈ (S_m, S_γ) do
3:     cost_row ← ARRAY()
```

Pseudocode 3.14b: Fungsi CALCASSIGNMENTCOST (2)

```
    for each det ∈ detections do
        cost ← FA(s_m, det)+FS(s_m, det)+FR(s_m, det, s_γ)
        insert cost to cost_row
    end for
    insert cost_row to cost_matrix
end for
return cost_matrix
```

3.2.15. Desain Fungsi Cost Fr

Fungsi ini merupakan implementasi dari Persamaan 2.21 yang menghitung *cost* komponen spasial dengan modifikasi penghapusan log sehingga keluaran dalam rentang $[0,\ 1]$. Fungsi *cost* F_r mirip dengan F_c dan mengintegrasikan batasan struktural juga. Yang membedakan adalah posisi relatif dihitung menggunakan batasan struktural dengan s_γ. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.15. Desain fungsi terdapat pada Pseudocode 3.15.

Tabel 3.15 Masukan, Proses, dan Keluaran dari Fungsi FR

Masukan	Proses	Keluaran
$object$ yang merupakan objek, $detection$ yang merupakan deteksi, s_γ yang merupakan pasangan objek *gamma* dari $object$	Menghitung *cost* F_r sesuai persamaan	Bilangan real *cost* yang merupakan hasil perhitungan *cost* F_r

Pseudocode 3.15: Fungsi FR

Input: $object$, $detection$, s_γ
Output: $cost$

```
sc ← StructuralConstraint(object, matched_gamma)
moved ← [s_γ.x + sc.delta_x, s_γ.y + sc.delta_y, object.width, object.height]
iou_score ← IoU(moved, detection)
cost ← −Log(iou_score)
return cost
```

3.2.16. Desain Fungsi Penentuan Asosiasi

Dari matriks *cost*, ditambahkan matriks untuk *cost* kasus d_0. Matriks tambahan ini merupakan matriks berukuran NxN dengan diagonal bernilai *cost* khusus d_0 dan elemen lain bernilai tak hingga. Kemudian, matriks hasil penggabungan dimasukkan ke dalam algoritma *linear assignment* untuk mendapatkan pasangan objek dan deteksi yang sesuai. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.16. Desain fungsi terdapat pada Pseudocode 3.16.

Tabel 3.16 Masukan, Proses, dan Keluaran dari Fungsi ASSIGNMENT

Masukan	Proses	Keluaran
Array 2D $cost_matrix$ yang merupakan matriks *cost* asosiasi	Menentukan asosiasi berdasarkan *cost* dengan *linear assignment*	*array* 2D $matrix$ yang merupakan matriks asosiasi *boolean*

Pseudocode 3.16: Fungsi ASSIGNMENT

Input: $cost_matrix$
Output: $matrix$

$N,\ M \leftarrow cost_matrix.shape()$
$d0_matrix \leftarrow$ IDENTITY(N)
$d0_matrix[d0_matrix = 0] \leftarrow \infty$
$d0_matrix[d0_matrix = 1] \leftarrow cost_d0$
$combined_cost \leftarrow$ HSTACK$(cost_matrix,\ d0_matrix)$
$pair \leftarrow$ LINEARASSIGNMENT$(combined_cost)$
$matrix \leftarrow$ ARRAY() set to $False$
$matrix[pair] \leftarrow True$
return $matrix[:, : M]$

3.2.17. Desain Fungsi Pembaruan Nilai Objek

Semua objek yang berhasil diasosiasikan akan diperbarui nilainya berdasarkan deteksi yang diasosiasikan. Pembaruan nilai dilakukan melalui *Kalman filter*. Nilai pengukuran z yang menjadi masukan untuk *Kalman filter* adalah $[x,\ y,\ w,\ h]^T$ dengan $(x,\ y)$ merupakan komponen posisi dan $(w,\ h)$ merupakan komponen ukuran dari deteksi. Nilai internal $\hat{x}$ yang disimpan pada *Kalman filter* adalah $[x,\ y,\ \dot{x},\ \dot{y},\ w,\ h]^T$ dengan $(\dot{x},\ \dot{y})$ merupakan komponen kecepatan.

Matriks F dan R yang digunakan dalam *Kalman filter* sesuai dengan yang terdapat pada Persamaan 2.24. Untuk matriks H dilakukan perubahan karena ditemukan ketidakcocokan pada proses perhitungan $\hat{x}$ yang terdapat operasi $z - H\hat{x}^-$. Matriks H berfungsi memetakan nilai $\hat{x}^-$ ke z yang mana keduanya memiliki perbedaan pada keberadaan komponen $(\dot{x},\ \dot{y})$. Sehingga, matriks H yang digunakan seperti pada Persamaan 3.1.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Matriks Q yang digunakan pada *Kalman filter* juga mengalami modifikasi berdasarkan pemodelan pergerakan [14] yang ada pada Persamaan 3.2 dan hubungan kovarians antar komponen yang ada pada Persamaan 3.3. $\ddot{x}$ dan $\ddot{y}$ merupakan percepatan di sumbu x dan y. Pada Persamaan 3.3, posisi dan kecepatan saling berkaitan pada sumbu yang sama, sedangkan ukuran w dan h tidak saling berkaitan.

$$\begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + \dot{x}_{t-1}\Delta t + 1/2\ddot{x}_{t-1}\Delta t \\ y_{t-1} + \dot{y}_{t-1}\Delta t + 1/2\ddot{y}_{t-1}\Delta t \\ \dot{x}_{t-1} + \ddot{x}_{t-1}\Delta t \\ \dot{y}_{t-1} + \ddot{y}_{t-1}\Delta t \end{bmatrix} \tag{3.2}$$

$$Q = \begin{array}{c} \\ x \\ y \\ \dot{x} \\ \dot{y} \\ w \\ h \end{array} \begin{array}{c} \begin{array}{cccccc} x & y & \dot{x} & \dot{y} & w & h \end{array} \\ \begin{bmatrix} \sigma_x^2 & 0 & \sigma_x\sigma_{\dot{x}} & 0 & 0 & 0 \\ 0 & \sigma_y^2 & 0 & \sigma_y\sigma_{\dot{y}} & 0 & 0 \\ \sigma_{\dot{x}}\sigma_x & 0 & \sigma_{\dot{x}}^2 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{y}}\sigma_y & 0 & \sigma_{\dot{y}}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_w & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_h \end{bmatrix} \end{array} \tag{3.3}$$

Berdasarkan pemodelan pada Persamaan 3.2, kovarians untuk posisi $(\sigma_x,\ \sigma_y)$ dan kecepatan $(\sigma_{\dot{x}},\ \sigma_{\dot{y}})$ dapat dinyatakan dalam kovarians kecepatan σ_a atau ditulis sebagai σ_q pada Persamaan 2.24 menjadi seperti Persamaan 3.4 dan 3.5.

$$\sigma_x = \sigma_q \frac{\Delta t^2}{2}, \qquad \sigma_y = \sigma_q \frac{\Delta t^2}{2} \tag{3.4}$$

$$\sigma_{\dot{x}} = \sigma_q \Delta t, \qquad \sigma_y = \sigma_q \Delta t \tag{3.5}$$

Sehingga, dengan mensubtitusi nilai Δt sebesar 1,0 dan menyamakan $\sigma_w = \sigma_h = \sigma_s$ seperti Persamaan 2.24, maka didapat matriks Q seperti Persamaan 3.6. Nilai σ_q dan σ_s sama seperti yang dijelaskan pada Subbab 2.6.3.1.

$$Q = \begin{bmatrix} 0.25\sigma_q^2 & 0 & 0.5\sigma_q^2 & 0 & 0 & 0 \\ 0 & 0.25\sigma_q^2 & 0 & 0.5\sigma_q^2 & 0 & 0 \\ 0.5\sigma_q^2 & 0 & \sigma_q^2 & 0 & 0 & 0 \\ 0 & 0.5\sigma_q^2 & 0 & \sigma_q^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_s^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_s^2 \end{bmatrix} \tag{3.6}$$

Pembaruan juga dilakukan untuk histogram tiap objek sesuai dengan Persamaan 2.25. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.17. Desain fungsi terdapat pada Pseudocode 3.17.

Tabel 3.17 Masukan, Proses, dan Keluaran dari Fungsi UPDATEOBJECT

Masukan	Proses	Keluaran
Array $\mathcal{S}_w$ yang merupakan kumpulan objek, *array* $\mathcal{D}$ yang merupakan kumpulan deteksi	Memperbarui nilai *state* objek	*Array* $\mathcal{S}_w$ yang sudah diperbarui nilainya

Pseudocode 3.17: Fungsi UPDATEOBJECT

Input: $\mathcal{S}_w$, $\mathcal{D}$
Output: $\mathcal{S}_w$
1: **for each** $(s,\ d) \in (\mathcal{S}_w,\ \mathcal{D})$ **do**
2: $s \leftarrow$ KALMANFILTEROBJECTUPDATE(d)
3: $s.histogram \leftarrow$ UPDATEHIST$(d.histogram)$
4: **end for**
5: **return** $\mathcal{S}_w$

3.2.18. Desain Fungsi Pembaruan Nilai Batasan Struktural

Pembaruan batasan struktural dari pasangan objek di $\mathcal{S}_w$ dilakukan melalui *Kalman filter*. Nilai pengukuran z yang menjadi masukan untuk *Kalman filter* adalah $[\chi,\ \xi]^T$ dengan χ dan ξ merupakan komponen perbedaan posisi. Nilai internal $\hat{x}$ yang disimpan pada *Kalman filter* adalah $[\chi,\ \xi,\ \dot{\chi},\ \dot{\xi}]^T$ dengan $\dot{\chi}$ dan $\dot{\xi}$ merupakan komponen perbedaan kecepatan.

Matriks F_{sc}, H_{sc}, dan R_{sc} yang digunakan dalam *Kalman filter* sesuai dengan Persamaan 2.27. Untuk matriks Q dilakukan perubahan juga sesuai dengan dasar yang dijelaskan pada Subbab 3.2.17 sehingga menjadi seperti Persamaan 3.7 dengan nilai σ_{sc} diatur sebesar 1,0.

$$Q = \begin{bmatrix} 0.25\sigma_{sc}^2 & 0 & 0.5\sigma_{sc}^2 & 0 \\ 0 & 0.25\sigma_{sc}^2 & 0 & 0.5\sigma_{sc}^2 \\ 0.5\sigma_{sc}^2 & 0 & \sigma_{sc}^2 & 0 \\ 0 & 0.5\sigma_{sc}^2 & 0 & \sigma_{sc}^2 \end{bmatrix} \tag{3.7}$$

Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.18. Desain fungsi terdapat pada Pseudocode 3.18.

Tabel 3.18 Masukan, Proses, dan Keluaran dari Fungsi UPDATESC

Masukan	Proses	Keluaran
Array $\mathcal{S}_w$ yang merupakan kumpulan objek terlacak, *array* $\mathcal{D}$ yang merupakan kumpulan deteksi sesuai $\mathcal{S}_w$, *array* 2D $\mathcal{E}$ yang merupakan kumpulan batasan struktural	Memperbarui nilai *state* batasan struktural	*Array* 2D $\mathcal{E}$ yang sudah diperbarui nilainya

Pseudocode 3.18: Fungsi UPDATESC

Input: $\mathcal{S}_w$, $\mathcal{D}$, $\mathcal{E}$
Output: $\mathcal{E}$

```
for i ← 0 to LEN(E) − 1 do
    for j ← 0 to LEN(E) − 1 do
        if s_i ∈ S_w and s_j ∈ S_w then
            sc ← STRUCTURALCONSTRAINT(d_i, d_j)
            E[i][j] ← KALMANFILTERSCUPDATE(sc)
        end if
    end for
end for
return E
```

3.2.19. Desain Fungsi Prediksi Nilai Objek

Semua objek baik terlacak ataupun hilang, akan diprediksi nilainya menggunakan *Kalman filter* dengan matriks seperti pada Subbab 3.2.17 untuk mendapatkan prediksi nilai objek di *frame* saat

ini berdasarkan nilai yang terakhir masuk di UPDATEOBJECT. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.19. Desain fungsi terdapat pada Pseudocode 3.19.

Tabel 3.19 Masukan, Proses, dan Keluaran dari Fungsi PREDICTOBJECT

Masukan	Proses	Keluaran
Array $\mathcal{S}$ yang merupakan kumpulan semua objek	Memprediksi nilai *state* objek	*Array* $\mathcal{S}$ yang sudah diprediksi nilainya

Pseudocode 3.19: Fungsi PREDICTOBJECT

Input: $\mathcal{S}$
Output: $\mathcal{S}$
1: **for each** $s \in \mathcal{S}$ **do**
2: $s \leftarrow$ KALMANFILTEROBJECTPREDICT()
3: **end for**
4: **return** $\mathcal{S}$

3.2.20. Desain Fungsi Prediksi Batasan Struktural

Semua batasan struktural akan diprediksi nilainya untuk mendapatkan prediksi nilai batasan struktural di *frame* saat ini berdasarkan nilai yang terakhir masuk di UPDATEOBJECT. Batasan struktural yang pasangan objeknya terlacak akan diprediksi menggunakan *Kalman filter* dengan matriks seperti pada Subbab 3.2.18, sedangkan untuk yang salah satu atau kedua objeknya menghilang diprediksi dengan matriks F_{sc} saja. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.20. Desain fungsi terdapat pada Pseudocode 3.20.

Tabel 3.20 Masukan, Proses, dan Keluaran dari Fungsi PREDICTSC

Masukan	Proses	Keluaran
Array 2D $\mathcal{E}$ yang merupakan kumpulan semua batasan struktural	Memprediksi nilai *state* batasan struktural	*Array* $\mathcal{E}$ yang sudah diprediksi nilainya

Pseudocode 3.20: Fungsi PREDICTSC

Input: $\mathcal{E}$
Output: $\mathcal{E}$

```
1:  for i ← 0 to LEN(E) − 1 do
2:      for j ← 0 to LEN(E) − 1 do
3:          if s_i ∈ S_w and s_j ∈ S_w then
4:              E[i][j] ← KALMANFILTERSCPREDICT()
5:          else
6:              E[i][j] ← MATMUL(F_sc, E[i][j])
7:          end if
8:      end for
9:  end for
10: return E
```

3.2.21. Desain Fungsi Eliminasi Objek

Dilakukan eliminasi untuk objek yang sudah hilang dalam periode yang melebihi batas maksimal yang ditentukan. Objek-objek yang berhasil dilacak ditandai telah diperbarui pada *frame* saat ini. Kemudian, dihitung selisih jeda pembaruan nilai untuk semua objek. Objek-objek yang jedanya melebihi batas akan dihapus dari daftar objek. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.21. Desain fungsi terdapat pada Pseudocode 3.21.

Tabel 3.21 Masukan, Proses, dan Keluaran dari Fungsi ELIMINATE

Masukan	Proses	Keluaran
Array $\mathcal{S}$ yaitu kumpulan objek, *array* $\mathcal{I}_w$ yang menyatakan indeks objek terlacak, bilangan bulat num_frame yang menyatakan *frame* sekarang, bilangan bulat max_age yang menyatakan batas lama objek menghilang	Menghapus objek yang menghilang melebihi max_age	*Array* $\mathcal{S}_{selected}$ yaitu kumpulan objek terpilih

Pseudocode 3.21: Fungsi ELIMINATE

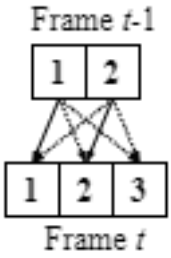
Input: $\mathcal{S}$, $\mathcal{I}_w$, num_frame, max_age
Output: $\mathcal{S}_{selected}$

1: **for each** $object \in \mathcal{S}[\mathcal{I}_w]$ **do**
2: set $last_update$ to num_frame
3: **end for**
4: $\mathcal{S}_{selected} \leftarrow$ ARRAY()
5: **for each** $object \in \mathcal{S}$ **do**
6: $diff \leftarrow last_update - num_frame$
7: **if** $diff < max_age$ **then**
8: insert $object$ to $\mathcal{S}_{selected}$
9: **end if**
10: **end for**
11: **return** $\mathcal{S}_{selected}$

3.2.22. Desain Fungsi Pembuatan Objek Baru

Deteksi yang belum diasosiasikan akan dibuat menjadi objek baru. Ada dua metode inisialisasi objek baru. Pada metode I semua deteksi yang belum diasosiasikan dengan objek di *frame* sekarang dibuat menjadi objek baru.

Pada metode II deteksi yang belum diasosiasikan tersebut disimpan dalam suatu *sliding window* dengan besar *window* 2 seperti pada Gambar 3.5. Pada contoh, di *frame* $t-1$ terdapat 2 deteksi dan pada *frame* t ada 3 deteksi. Kemudian dihitung nilai IoU untuk tiap pasangan deteksi antara *frame* $t-1$ dan t dan disimpan dalam matriks 2D. Dari matriks nilai IoU, dilakukan proses *linear assignment* untuk mendapatkan pasangan deteksi yang beririsan paling besar. Dari pasangan deteksi yang didapat akan dilakukan seleksi lagi. Deteksi akan dijadikan objek baru jika dalam *window* ada deteksi sebelumnya yang memiliki nilai IoU dalam ambang batas dengan deteksi sekarang. Ambang batas yang digunakakan sebesar 0,5. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.22. Desain fungsi pembuatan objek baru metode I terdapat pada Pseudocode 3.22. Desain fungsi pembuatan objek baru metode II terdapat pada Pseudocode 3.23a dan 3.23b.



Gambar 3.5 *Sliding Window* antara *Frame t-1* dan *t*

Tabel 3.22 Masukan, Proses, dan Keluaran dari Fungsi NEWOBJECTS

Masukan	Proses	Keluaran
Array $detections$ yaitu kumpulan deteksi	Membuat objek baru dari deteksi yang belum diasosiasikan	*array* $objects$ yaitu kumpulan objek, *array* sc yaitu kumpulan batasan struktural, *array* $object_tracker$ yaitu kumpulan *tracker* objek, dan *array* $sc_tracker$ yaitu kumpulan *tracker* batasan struktural

Pseudocode 3.22: Fungsi NEWOBJECTS Metode I

Input: $detections$
Output: $objects, sc, object_tracker, sc_tracker$
1: $objects \leftarrow$ new object from $detections$
2: $sc \leftarrow$ new SC from $objects$
3: $object_tracker \leftarrow$ new tracker from $objects$
4: $sc_tracker \leftarrow$ new tracker from sc
5: **return** $objects, sc, object_tracker, sc_tracker$

Pseudocode 3.23a: Fungsi NEWOBJECTS Metode II (1)

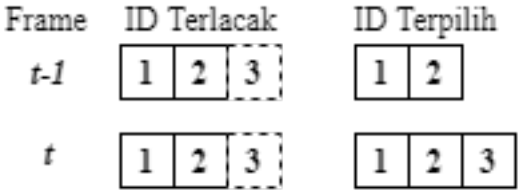
Input: $detections$
Output: $objects, sc, object_tracker, sc_tracker$
1: $iou_scores \leftarrow$ IOU$(detections,\ window)$
2: $matched \leftarrow$ LINEARASSIGNMENT(iou_scores)

Pseudocode 3.23b: Fungsi NEWOBJECTS Metode II (2)

```
3: matched ← THRESHOLD(matched, 0.5)
4: objects ← new object from matched
5: sc ← new SC from objects
6: object_tracker ← new tracker from objects
7: sc_tracker ← new tracker from sc
8: return objects, sc, object_tracker, sc_tracker
```

3.2.23. Desain Fungsi Seleksi Pelacakan Beruntun

Fungsi ini berguna untuk mengurangi tingkat *false negative* karena adanya deteksi yang salah pada *frame*. Fungsi ini digunakan dalam seleksi objek terlacak yang akan digunakan sebagai keluaran akhir proses *object tracking* pada *frame* saat ini. Pada Gambar 3.6 terdapat contoh seleksi objek dengan batas minimal terlacak dua kali beruntun. Pada *frame* $t-1$, ada objek baru yang terlacak yaitu objek dengan ID 3, tetapi objek tersebut tidak termasuk dalam ID terpilih karena baru terlacak satu kali. Kemudian pada *frame* t, objek ID 3 terlacak lagi sehingga masuk ke ID terpilih karena sudah terlacak dua kali secara beruntun. Masukan, proses, dan keluaran dari fungsi terdapat pada Tabel 3.23. Desain fungsi terdapat pada Pseudocode 3.24.



Gambar 3.6 Ilustrasi Seleksi berdasarkan Pelacakan Beruntun dengan Batas Minimal Sebesar Dua

Tabel 3.23 Masukan, Proses, dan Keluaran dari Fungsi FILTERSTREAK

Masukan	Proses	Keluaran
Array $\mathcal{S}$ yaitu kumpulan objek, *array* $\mathcal{I}_w$ yang menyatakan indeks objek terlacak, bilangan bulat min_streak	Memilih objek sesuai syarat min_streak	*Array* $\mathcal{S}_{selected}$ yaitu kumpulan objek terpilih

Pseudocode 3.24: Fungsi FILTERSTREAK

Input: $\mathcal{S}$, $\mathcal{I}_w$, min_streak
Output: $\mathcal{S}_{selected}$

1: $\mathcal{S}_{selected} \leftarrow$ ARRAY()
2: **for** $i \leftarrow 0$ **to** LEN($\mathcal{S}$) $- 1$ **do**
3: **if** $i \in \mathcal{I}_w$ **then**
4: increment $\mathcal{S}[i].hit_streak$
5: **if** $hit_streak \geq min_streak$ **or** $frame \leq min_streak$ **then**
6: insert $\mathcal{S}[i]$ into $\mathcal{S}_{selected}$
7: **end if**
8: **else**
9: set $\mathcal{S}[i].hit_streak$ to 0
10: **end if**
11: **end for**
12: **return** $\mathcal{S}_{selected}$

BAB IV
IMPLEMENTASI

4.1. Lingkungan Implementasi

Lingkungan implementasi dan pengembangan yang dilakukan adalah sebagai berikut.

1. Perangkat Keras
   (a) Prosesor Intel® Core™ i5-6200U CPU @ 2,3GHz (2 CPUs, 4 Threads)
   (b) *Random Access Memory* 8192MB
2. Perangkat Lunak
   (a) Sistem Operasi Windows 10 Home 64-bit
   (b) *Text Editor* Visual Studio Code
   (c) Bahasa Pemrograman Python 3.6.4 64-bit
   (d) Microsoft Visual C++ 2015 (14.0)

4.2. Implementasi Program Utama

Subbab ini menjelaskan implementasi program utama. Program ini digunakan untuk mengolah masukan data deteksi dan menjalankan *object tracking*. Fungsi yang didesain di Bab 3 diimplementasikan dalam beberapa modul dan *class*. Modul yaitu COST, PARTITION, SCEA, dan SCOR. *Class* utama yaitu OBJECTSTATETRACKER, STRUCTURALCONSTRAINTTRACKER, dan TRACKER. *Class* pembantu yaitu OBJECTSTATE, DETECTIONSTATE, dan STRUCTURALCONSTRAINT. Ada fungsi MAIN yang menjadi antarmuka program utama.

4.2.1. Penggunaan Pustaka

Seluruh kode sumber yang dituliskan di bab ini membutuhkan pustaka yang ditunjukkan pada Kode Sumber 4.1 Penjelasan dan fungsi pustaka yang digunakan terdapat pada Tabel 4.1.

```
 1 # external lib
 2 import copy
 3 import math
 4 from typing import (
 5     List, NamedTuple, Optional, Set, Tuple, Union
 6 )
 7 import numpy as np
 8 from dataclasses import asdict, dataclass
 9 from filterpy.kalman import KalmanFilter
10 from k_means_constrained import KMeansConstrained
11 from scipy.linalg import block_diag
12 from scipy.optimize import linear_sum_assignment
13
14 # project modules and classes
15 from sc_tracker import scea
16 from sc_tracker import scor
17 from sc_tracker.cost import (
18     calculate_fs, calculate_structural_constraint,
19     f_a, f_c, f_s,iou,
20 )
21 from sc_tracker.partition import (
22     gating, possible_assignment_generator,
23     subgroup_by_cluster_constrained,
24 )
25 from sc_tracker.state import (
26     DetectionState, ObjectState,
27     StructuralConstraint,
28 )
29 from sc_tracker.tracker.object_state import
       ObjectStateTracker
30 from sc_tracker.tracker.structural_constraint
       import (
31     StructuralConstraintTracker,
32 )
```

Kode Sumber 4.1 Daftar Deklarasi *Import* Program

Tabel 4.1 Nama dan Penjelasan Pustaka

Nama Pustaka	Penjelasan
copy	membantu proses kloning objek
math	perhitungan matematika
typing	memberikan anotasi tipe data
numpy	komputasi ilmiah yang berhubungan dengan *array* dan *matrix*
dataclasses	membuat *boilerplate* kode *class*
filterpy	proses *Kalman filter*
k_means_constrained	melakukan proses K-Means dengan batasan
scipy	komputasi ilmiah dan menyelesaikan *linear assignment*

4.2.2. Implementasi Class DetectionState

Class ini merupakan *class* pembantu yang menyimpan informasi *bounding box* dan histogram dari data hasil *object detection*. Pada *class* ini terdapat beberapa variabel *instance* yang dijelaskan pada Tabel 4.2. Implementasi terdapat pada Kode Sumber 4.2.

Tabel 4.2 Nama dan Penjelasan Variabel Instance dari Class DETECTIONSTATE

Nama Variabel	Penjelasan
x	posisi titik tengah *bounding box* sumbu-x
y	posisi titik tengah *bounding box* sumbu-y
width	lebar deteksi
height	tinggi deteksi
frame_step	menyimpan ID *frame*
histogram	histogram dari citra deteksi

```
@dataclass
class DetectionState:
    x: float
    y: float
    width: float
    height: float
    frame_step: int
    histogram: np.ndarray

    @staticmethod
    def from_bbox(
        left: int,
        top: int,
        width: int,
        height: int,
        frame_step: int,
        full_image: np.ndarray,
        n_bins: int = 8,
    ) -> "DetectionState":
    ...
```

Kode Sumber 4.2 *Class* DETECTIONSTATE

4.2.2.1. Implementasi Fungsi Instansiasi dari Bounding Box

Fungsi ini berguna untuk melakukan instansiasi DETECTIONSTATE dari data *bounding box* dan citra satu *frame*. Implementasi terdapat pada Kode Sumber 4.1a dan 4.1b

```
@staticmethod
def from_bbox(
    left: int,
    top: int,
    width: int,
    height: int,
    frame_step: int,
```

Kode Sumber 4.1a Fungsi Instansiasi dari Bounding Box (1)

```
    full_image: np.ndarray,
    n_bins: int = 8,
) -> "DetectionState":
    left = int(max(left, 0))
    top = int(max(top, 0))
    x = left + width / 2
    y = top + height / 2
    histogram = None
    if full_image is not None:
        image = full_image[
            top : top + int(height),
            left : left + int(width),
        ]
        graysacle = image.mean(axis=2)
        bins = np.arange(n_bins + 1) * (256 //
            n_bins)
        histogram, _ = np.histogram(
            graysacle, bins=bins
        )
        histogram = histogram / histogram.sum()
    return DetectionState(
        x=x,
        y=y,
        width=width,
        height=height,
        frame_step=frame_step,
        histogram=histogram,
    )
```

Kode Sumber 4.1b Fungsi Instansiasi dari Bounding Box (2)

4.2.3. Implementasi Class ObjectState

Class ini merupakan *class* pembantu yang mewakili objek dalam *object tracking*. Pada *class* ini terdapat beberapa variabel *instance* yang dijelaskan pada Tabel 4.3. Implementasi terdapat pada Kode Sumber 4.2.

Tabel 4.3 Nama dan Penjelasan Variabel Instance dari Class OBJECTSTATE

Nama Variabel	Penjelasan
x	posisi titik tengah *bounding box* sumbu-x
y	posisi titik tengah *bounding box* sumbu-y
width	lebar objek
height	tinggi objek
frame_step	menyimpan ID *frame*
histogram	histogram dari citra objek
v_x	kecepatan di sumbu-x
v_y	kecepatan di sumbu-y

```
@dataclass
class ObjectState:
    x: float
    y: float
    width: float
    height: float
    frame_step: int
    histogram: np.ndarray
    v_x: float = 0.0
    v_y: float = 0.0

    @staticmethod
    def from_detection(
        detection: DetectionState,
    ) -> "ObjectState":
    ...
    def update_from_object_state(
        self, object_state: "ObjectState"
    ):
    ...
```

Kode Sumber 4.2 *Class* OBJECTSTATE

4.2.4. Implementasi Class StructuralConstraint

Class ini merupakan *class* pembantu untuk menyimpan batasan struktural dari pasangan OBJECTSTATE. Pada *class* ini terdapat beberapa variabel *instance* yang dijelaskan pada Tabel 4.4 . Implementasi terdapat pada Kode Sumber 4.3.

Tabel 4.4 Nama dan Penjelasan Variabel Instance dari Class STRUCTURALCONSTRAINT

Nama Variabel	Penjelasan
delta_x	selisih posisi di sumbu-x
delta_y	selisih posisi di sumbu-y
delta_v_x	selisih kecepatan di sumbu-x
delta_v_y	selisih kecepatan di sumbu-y

```
 1 @dataclass
 2 class StructuralConstraint:
 3     delta_x: float
 4     delta_y: float
 5     delta_v_x: float
 6     delta_v_y: float
 7
 8     @staticmethod
 9     def create(
10         first_object: ObjectState,
11         second_object: ObjectState,
12     ) -> "StructuralConstraint":
13     ...
14
15     def update_from_sc(self, sc:
           "StructuralConstraint"):
16     ...
```

Kode Sumber 4.3 *Class* STRUCTURALCONSTRAINT

4.2.4.1. Implementasi Fungsi Instansiasi dari Pasangan ObjectState

Fungsi ini berguna untuk membuat StructuralConstraint dari dua ObjectState. Implementasi terdapat pada Kode Sumber 4.4.

```
@staticmethod
def create(
    first_object: ObjectState,
    second_object: ObjectState,
) -> "StructuralConstraint":
    delta_x = first_object.x - second_object.x
    delta_y = first_object.y - second_object.y
    delta_v_x = first_object.v_x -
        second_object.v_x
    delta_v_y = first_object.v_y -
        second_object.v_y

    return StructuralConstraint(
        delta_x=delta_x,
        delta_y=delta_y,
        delta_v_x=delta_v_x,
        delta_v_y=delta_v_y,
    )
```

Kode Sumber 4.4 Fungsi Instansiasi dari Pasangan ObjectState

4.2.5. Implementasi Modul cost

Modul cost digunakan untuk menyimpan fungsi-fungsi yang akan digunakan untuk menghitung *cost* yang dibutuhkan. Berikut adalah penjelasan dari masing-masing fungsi.

4.2.5.1. Implementasi Fungsi Cost Fs

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.8 yang menerima masukan ObjectState dan

DETECTIONSTATE dan mengembalikan hasil perhitungan *cost*. Implementasi terdapat pada Kode Sumber 4.5.

```
def f_s(
    object_state: ObjectState,
    detection_state: DetectionState,
) -> float:
    h_obj = object_state.height
    w_obj = object_state.width
    h_det = detection_state.height
    w_det = detection_state.width
    h = abs(h_obj - h_det) / (2 * (h_obj + h_det))
    w = abs(w_obj - w_det) / (2 * (w_obj + w_det))
    return -math.log(1 - h - w)
```

Kode Sumber 4.5 Fungsi Cost Fs

4.2.5.2. Implementasi Fungsi Cost Fa

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.9 yang menerima masukan OBJECTSTATE dan DETECTIONSTATE dan mengembalikan hasil perhitungan *cost*. Implementasi terdapat pada Kode Sumber 4.6.

```
def f_a(
    object_state: ObjectState,
    detection_state: DetectionState,
) -> float:
    hist_obj = object_state.histogram
    hist_det = detection_state.histogram
    root = np.sqrt(hist_obj * hist_det)
    summation = root.sum()
    return -math.log(summation)
```

Kode Sumber 4.6 Fungsi Cost Fa

4.2.5.3. Implementasi Fungsi Cost Fc

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.10 yang menerima masukan OBJECTSTATE,

DETECTIONSTATE, dan STRUCTURALCONSTRAINT dan mengembalikan hasil perhitungan *cost*. Implementasi terdapat pada Kode Sumber 4.7.

```
def f_c(
    object_state: ObjectState,
    sc_ij: StructuralConstraint,
    detection_k: DetectionState,
    detection_q: DetectionState,
) -> float:
    x = detection_k.x - sc_ij.delta_x -
         object_state.width / 2
    y =  detection_k.y - sc_ij.delta_y -
         object_state.height / 2
    s_jk = [ x, y, x + object_state.width, y +
         object_state.height]
    det_q = [
        detection_q.x - detection_q.width / 2,
        detection_q.y - detection_q.height / 2,
        detection_q.x + detection_q.width / 2,
        detection_q.y + detection_q.height / 2,
    ]
    iou_score = iou(s_jk, det_q)
    return 1 - iou_score
```

Kode Sumber 4.7 Fungsi Cost Fc

4.2.5.4. Implementasi Fungsi Cost Fr

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.15 yang menerima masukan OBJECTSTATE dan DETECTIONSTATE dan mengembalikan hasil perhitungan *cost*. Implementasi terdapat pada Kode Sumber 4.8a dan 4.8b.

```
def f_r(
    object_state: ObjectState,
    detection: DetectionState,
    match_gamma: ObjectState) -> float:
```

Kode Sumber 4.8a Fungsi Cost Fr (1)

```
    sc_object_gamma = StructuralConstraint.create(
        object_state, match_gamma
    )
    x = match_gamma.x + sc_object_gamma.delta_x -
         object_state.width / 2
    y = match_gamma.y + sc_object_gamma.delta_y -
         object_state.height / 2
    s_i_gamma = [x, y, x + object_state.width, y
         + object_state.height]

    det_q = [
        detection.x - detection.width / 2,
        detection.y - detection.height / 2,
        detection.x + detection.width / 2,
        detection.y + detection.height / 2,
    ]

    iou_score = iou(s_i_gamma, det_q)
    return 1 - iou_score
```

Kode Sumber 4.8b Fungsi Cost Fr (2)

4.2.6. Implementasi Modul partition

Modul PARTITION digunakan untuk menyimpan fungsi-fungsi yang akan digunakan dalam proses pembuatan partisi dan pembangkitan kombinasi asosiasi. Berikut adalah penjelasan dari masing-masing fungsi tersebut.

4.2.6.1. Implementasi Fungsi Gating

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.4. Parameter fungsi ini adalah *array* 2D *fs_matrix* sebagai matriks *cost fs*, *object_states* dan *detections_states* sebagai kumpulan objek dan deteksi, serta bilangan riil *threshold* sebagai nilai τ seperti pada Persamaan 2.11. Implementasi terdapat pada Kode Sumber 4.9.

```
def gating(
    fs_matrix: np.ndarray,
    object_states: List[ObjectState],
    detection_states: List[DetectionState],
    threshold: float = 0.7,
) -> np.ndarray:
    distance_matrix = __center_distance(
        object_states, detection_states
    )
    diagonal_vector = __diagonal(object_states)
    diagonal_column_vector =
         diagonal_vector.reshape(
        -1, 1
    )
    diagonal_matrix = np.tile(
        diagonal_column_vector,
        (1, len(detection_states))
    )

    mask_center = distance_matrix <
         diagonal_matrix
    mask_fs = np.exp(-fs_matrix) > threshold
    mask =  mask_center & mask_fs
    return mask
```

Kode Sumber 4.9 Fungsi Gating

4.2.6.2. Implementasi Fungsi Partisi

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.5. Parameter fungsi ini adalah *array object_states* sebagai objek dan bilangan bulat *n_member* sebagai batas maksimal anggota per partisi. Implementasi terdapat pada Kode Sumber 4.10.

```
def subgroup_by_cluster_constrained(
    object_states: List[ObjectState],
    n_member: int = 5,
) -> List[List[int]]:
    n_cluster = math.ceil(
        len(object_states) / n_member
    )
    features = [
        [obj.x, obj.y] for obj in object_states
    ]
    kmeans = KMeansConstrained(
        n_clusters=n_cluster,
        size_max=min(
            n_member, len(object_states)
        ),
        random_state=42,
    )
    labels = kmeans.fit_predict(features)
    groups = []
    for label in set(labels):
        indices = np.flatnonzero(labels == label)
        groups.append(indices.tolist())
    return groups
```

Kode Sumber 4.10 Fungsi Partisi

4.2.6.3. Implementasi Fungsi Pembangkit Kombinasi Asosiasi

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.7. Parameter fungsi ini adalah *array* 2D *gated_assignment_matrix* sebagai matriks hasil *gating* dan *array subgroup* sebagai indeks dari objek yang ada dalam suatu partisi. Pembangkitan kombinasi dari *cross product* menggunakan fungsi MESHGRID dari pustaka *numpy* [15] kemudian dihapus kombinasi yang duplikat [16]. Implementasi terdapat pada Kode Sumber 4.11.

```
def possible_assignment_generator(
    gated_assignment_matrix: np.ndarray,
    subgroup: List[int],
) -> List[List[int]]:
    assignment_matrix =
         __assignment_matrix_for_subgroup(
        gated_assignment_matrix, subgroup
    )
    n_object = len(subgroup)
    possible_assignment = [
        [] for _ in range(n_object)
    ]
    rows, cols = assignment_matrix.nonzero()
    for object_index, detection_index in zip(
        rows, cols
    ):
        possible_assignment[object_index].append(
            detection_index
        )
    combination = np.array(
        np.meshgrid(*possible_assignment)
    )
    combination = combination.T.reshape(
        -1, n_object
    )
    combination = __remove_duplicate_row(
        combination, excludes=[0],
    )
    return combination.tolist()
```

Kode Sumber 4.11 Fungsi Pembangkit Kombinasi Asosiasi

4.2.7. Implementasi Modul scea

Modul SCEA digunakan untuk menyimpan fungsi-fungsi dan *class* pembantu yang akan digunakan dalam alur proses SCEA dari perhitungan *cost anchor* hingga agregat hasil asosiasi sesuai desain fungsi pada Subbab 3.2.3. Berikut adalah penjelasan dari masing-masing fungsi tersebut.

4.2.7.1. Implementasi Class ConfigSCEA

Class ini merupakan *class* pembantu yang digunakan untuk menyimpan konfigurasi parameter dalam proses SCEA. Pada *class*ini terdapat beberapa variabel *instance* yang dijelaskan pada Tabel 4.5. Implementasi *class* sebagai *NamedTuple* yang bersifat *immutable* terdapat pada Kode Sumber 4.12.

Tabel 4.5 Nama dan Penjelasan Variabel Instance dari Class CONFIGSCEA

Nama Variabel	Penjelasan
default_cost_d0	nilai *cost* standar untuk kasus deteksi yang salah
num_cluster_member	maksimal banyak objek tiap partisi
gating_threshold	nilai *threshold* dalam proses *gating*

```
1 class ConfigSCEA(NamedTuple):
2     default_cost_d0: float = 4.0
3     num_cluster_member: int = 5
4     gating_threshold: float = 0.7
```

Kode Sumber 4.12 Class CONFIGSCEA

4.2.7.2. Implementasi Fungsi Cost Anchor

Fungsi ini merupakan bagian implementasi dari desain fungsi pada Subbab 3.2.6 untuk menghitung *cost* objek *anchor* dengan deteksi yang diasosiasikan. Implementasi terdapat pada Kode Sumber 4.13.

```
1 def __cost_anchor(
2     object_state: ObjectState,
3     detection_state: DetectionState) -> float:
4     cost_fs = f_s(object_state, detection_state)
5     cost_fa = f_a(object_state, detection_state)
```

```
6     return cost_fs + cost_fa
```

Kode Sumber 4.13 Fungsi Cost Anchor

4.2.7.3. Implementasi Fungsi Cost Non-Anchor

Fungsi ini merupakan bagian implementasi dari desain fungsi pada Subbab 3.2.6 untuk menghitung *cost* pada objek dengan deteksi yang diasosiasikan berdasarkan acuan objek *anchor*. Implementasi terdapat pada Kode Sumber 4.14.

```
1  def __cost_non_anchor(
2      object_state: ObjectState,
3      detection_k: DetectionState,
4      detection_q: DetectionState,
5      sc_ij: StructuralConstraint,
6  ) -> float:
7      cost_fs = f_s(object_state, detection_q)
8      cost_fa = f_a(object_state, detection_q)
9      cost_fc = f_c(
10         object_state,
11         sc_ij,
12         detection_k,
13         detection_q,
14     )
15     return cost_fs + cost_fa + cost_fc
```

Kode Sumber 4.14 Fungsi Cost Non-Anchor

4.2.7.4. Implementasi Fungsi Cost Berdasarkan Anchor dalam Kombinasi

Fungsi ini merupakan bagian implementasi dari desain fungsi pada Subbab 3.2.6 untuk menghitung nilai *cost* dari asosiasi dalam satu kombinasi berdasarkan satu objek *anchor*. Implementasi terdapat pada Kode Sumber 4.15a dan 4.15b.

```
def cost_by_anchor(
    anchor_object_index: int,
    subgroup_member: List[int],
    possible_assignment: List[int],
    structural_constraints: List[
        List[StructuralConstraint]
    ], object_states: List[ObjectState],
    detection_states: List[DetectionState],
    default_cost_d0: float = 4.0,
) -> float:
    object_index = subgroup_member[
        anchor_object_index
    ]
    object_anchor = object_states[object_index]
    detection_index = (
        possible_assignment[anchor_object_index]
        - 1
    )
    detection_anchor = detection_states[
        detection_index
    ]
    cost_value = 0.0
    for obj_index, det_index in enumerate(
        possible_assignment
    ):
        real_obj_index = subgroup_member[
            obj_index
        ]
        real_det_index = det_index - 1

        # object index is equal to anchor index
        if real_obj_index == object_index:
            cost_value += __cost_anchor(
                object_anchor, detection_anchor
            )
```

Kode Sumber 4.15a Fungsi Cost Berdasarkan Anchor dalam Kombinasi (1)

```
36          else:
37              object_state = object_states[
38                  real_obj_index
39              ]
40              detection_state = detection_states[
41                  real_det_index
42              ]
43              # case for d0
44              if real_det_index == -1:
45                  cost_value += default_cost_d0
46              else:
47                  cost_value += __cost_non_anchor(
48                      object_state=object_state,
49                      detection_k=detection_anchor,
50                      detection_q=detection_state,
51                      sc_ij=structural_constraints[
52                          object_index
53                      ][real_obj_index],
54                  )
55      return cost_value
```

Kode Sumber 4.15b Fungsi Cost Berdasarkan Anchor dalam Kombinasi (2)

4.2.7.5. Implementasi Fungsi Perhitungan Cost Asosiasi dari Kombinasi

Fungsi ini merupakan bagian implementasi dari desain fungsi pada Subbab 3.2.6 yang menghitung rata-rata *cost* asosiasi berdasarkan semua objek *anchor* yang ada dalam satu kombinasi asosiasi. Implementasi terdapat pada Kode Sumber 4.16.

```
def cost_by_possible_assignment(
    possible_assignment: List[int],
    subgroup: List[int],
    structural_constraints: List[
        List[StructuralConstraint]],
    object_states: List[ObjectState],
    detection_states: List[DetectionState],
    default_cost_d0: float = 4.0,
) -> float:
    anchor_count = 0
    cost = 0.0
    for (
        object_index,
        detection_index,
    ) in enumerate(possible_assignment):
        if detection_index == 0:
            continue
        anchor_count += 1
        cost += cost_by_anchor(
            anchor_object_index=object_index,
            subgroup_member=subgroup,
            possible_assignment=
                possible_assignment,
            structural_constraints=
                structural_constraints,
            object_states=object_states,
            detection_states=detection_states,
            default_cost_d0=default_cost_d0,
        )
    if anchor_count == 0:
        return np.inf
    cost /= anchor_count
    return cost
```

Kode Sumber 4.16 Fungsi Perhitungan Cost Asosiasi dari Kombinasi

4.2.7.6. Implementasi Fungsi Perhitungan Asosiasi Per Partisi

Fungsi ini merupakan bagian implementasi dari desain fungsi pada Subbab 3.2.6 yang memilih kombinasi asosiasi dengan *cost* terkecil sebagai asosiasi terbaik untuk satu partisi yang diberikan. Keluaran dari fungsi ini adalah matriks asosiasi dan *cost* asosiasinya. Implementasi terdapat pada Kode Sumber 4.17a dan 4.17b.

```
 1 def best_assignment_by_subgroup(
 2     subgroup: List[int],
 3     gated_assignment_matrix: np.ndarray,
 4     object_states: List[ObjectState],
 5     detection_states: List[DetectionState],
 6     structural_constraints: Union[
 7         np.ndarray,
 8         List[List[StructuralConstraint]],
 9     ],
10     default_cost_d0: float = 4.0,
11 ) -> Tuple[np.ndarray, float]:
12     min_cost = np.inf
13     min_assignment_list = []
14     for (
15         possible_assignment
16     ) in possible_assignment_generator(
17         gated_assignment_matrix, subgroup
18     ):
19         cost = cost_by_possible_assignment(
20             possible_assignment=
                    possible_assignment,
21             subgroup=subgroup,
22             structural_constraints=
                    structural_constraints,
23             object_states=object_states,
24             detection_states=detection_states,
25             default_cost_d0=default_cost_d0,
26         )
27         if cost < min_cost:
28             min_cost = cost
```

Kode Sumber 4.17a Fungsi Perhitungan Asosiasi Per Partisi (1)

```
29              min_assignment_list = (
30                  possible_assignment.copy()
31              )
32      best_assignment = np.zeros_like(
33          gated_assignment_matrix, dtype=int
34      )
35      for obj_index, detection_index in enumerate(
36          min_assignment_list
37      ):
38          object_index = subgroup[obj_index]
39          detection_index -= 1  # compensate d0
40          if detection_index < 0:
41              continue
42          best_assignment[object_index][
43              detection_index
44          ] = 1
45      return best_assignment, min_cost
```

Kode Sumber 4.17b Fungsi Perhitungan Asosiasi Per Partisi (2)

4.2.7.7. Implementasi Fungsi Penggabungan Matriks Asosiasi

Fungsi ini merupakan implementasi desain fungsi dari Subbab 3.2.11. Fungsi ini akan memeriksa jika ada asosiasi ke deteksi yang sama dan memilih salah satu dengan *cost* terkecil. Implementasi terdapat pada Kode Sumber 4.18a dan 4.18b.

```
 1 def __aggregate(
 2     assignment_matrices: Union[
 3         np.ndarray, List[List[List[int]]]
 4     ],
 5     assignment_costs: List[float],
 6 ) -> Union[np.ndarray, List[List[int]]]:
 7     n_object = len(assignment_matrices[0])
 8     n_detection = len(assignment_matrices[0][0])
 9     assignment_matrix = np.zeros(
10         (n_object, n_detection), dtype="int")
```

Kode Sumber 4.18a Fungsi Penggabungan Matriks Asosiasi (1)

```
11      assignment_cost = np.empty(
12          (n_object, n_detection)
13      )
14      assignment_cost[:, :] = np.inf
15      for assignment, cost in zip(
16          assignment_matrices, assignment_costs
17      ):
18          mask = assignment == 1
19          mask_column = mask.sum(axis=0).flatten()
20          mask_global_column =
                 assignment_matrix.sum(
21              axis=0).flatten()
22
23          # check for same assignment
24          intersection_column_mask = np.logical_and(
25              mask_column, mask_global_column
26          )
27          if intersection_column_mask.sum() > 0:
28              intersection_index = np.where(
29                  intersection_column_mask == 1
30              )[0]
31              for (
32                  intersection
33              ) in intersection_index:
34                  global_cost = assignment_cost[
35                      :, intersection].min()
36                  global_row = assignment_cost[
37                      :, intersection].argmin()
38                  if global_cost < cost:
39                      mask[:, intersection] = False
40                  else:
41                      assignment_matrix[
42                          global_row, intersection
43                      ] = 0
44          assignment_matrix[mask] = 1
45          assignment_cost[mask] = cost
46      return assignment_matrix
```

Kode Sumber 4.18b Fungsi Penggabungan Matriks Asosiasi (2)

4.2.7.8. Implementasi Fungsi Penentuan Matriks Asosiasi

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.3. Dalam fungsi ini dilakukan proses SCEA mulai dari *gating*, pembagian menjadi partisi, dan pemilihan kombinasi asosiasi terbaik menjadi matriks asosiasi dengan *cost* terkecil. Implementasi terdapat pada Kode Sumber 4.19a dan 4.19b.

```
def best_assignment(
    object_states: List[ObjectState],
    detection_states: List[DetectionState],
    structural_constraints: Union[
        np.ndarray,
        List[List[StructuralConstraint]],
    ],
    default_cost_d0: float = 4.0,
    gating_threshold: float = 0.7,
    num_cluster_member: int = 5,
) -> np.ndarray:
    fs_matrix = calculate_fs(
        object_states, detection_states
    )
    gated_assignment_matrix = gating(
        fs_matrix=fs_matrix,
        object_states=object_states,
        detection_states=detection_states,
        threshold=gating_threshold,
    )
    subgroups = subgroup_by_cluster_constrained(
        object_states, n_member=num_cluster_member
    )
    assignment_matrices = []
    assignment_costs = []
    for subgroup in subgroups:
        (
            current_best_assignment,
            current_cost,
        ) = best_assignment_by_subgroup(
            subgroup=subgroup,
```

Kode Sumber 4.19a Fungsi Penentuan Matriks Asosiasi (1)

```
32              gated_assignment_matrix=
                    gated_assignment_matrix,
33              object_states=object_states,
34              detection_states=detection_states,
35              structural_constraints=
                    structural_constraints,
36              default_cost_d0=default_cost_d0,
37          )
38          assignment_matrices.append(
39              current_best_assignment
40          )
41          assignment_costs.append(current_cost)
42      assignment_matrix = __aggregate(
43          assignment_matrices, assignment_costs
44      )
45      return assignment_matrix
```

Kode Sumber 4.19b Fungsi Penentuan Matriks Asosiasi (2)

4.2.8. Implementasi Modul scor

Modul SCOR digunakan untuk menyimpan fungsi-fungsi dan *class* pembantu yang akan digunakan dalam alur proses SCOR dari penentuan objek *gamma* hingga penentuan asosiasi sesuai desain fungsi pada Subbab 3.2.12. Berikut adalah penjelasan dari masing-masing fungsi tersebut.

4.2.8.1. Implementasi Class ConfigSCOR

Class ini merupakan *class* pembantu yang digunakan untuk menyimpan konfigurasi parameter dalam proses SCOR. Pada *class* ini terdapat variabel *instance* yang dijelaskan pada Tabel 4.6. Implementasi *class* sebagai *NamedTuple* yang bersifat *immutable* terdapat pada Kode Sumber 4.20.

Tabel 4.6 Nama dan Penjelasan Variabel Instance dari Class CONFIGSCOR

Nama Variabel	Penjelasan
default_cost_d0	nilai *cost* standar untuk kasus deteksi yang salah

```
1 class ConfigSCOR(NamedTuple):
2     default_cost_d0: float = 4.0
```

Kode Sumber 4.20 *Class* CONFIGSCOR

4.2.8.2. Implementasi Fungsi Penentuan Objek Gamma

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.13. Objek *gamma* dipilih berdasarkan kemiripan pergerakan yang dihitung dari resultan kecepatan antar objek. Implementasi terdapat ada Kode Sumber 4.21.

```
 1 def matching_s_gamma(
 2     updated_objects: List[ObjectState],
 3     missing_objects: List[ObjectState],
 4 ) -> List[ObjectState]:
 5     s_gamma = []
 6     for missing in missing_objects:
 7         min_resultant = math.inf
 8         min_state = None
 9         for well_tracked in updated_objects:
10             resultant = velocity_resultant(
11                 missing, well_tracked
12             )
13             if resultant < min_resultant:
14                 min_resultant = resultant
15                 min_state = well_tracked
16         s_gamma.append(min_state)
17     return s_gamma
```

Kode Sumber 4.21 Fungsi Penentuan Objek Gamma

4.2.8.3. Implementasi Fungsi Perhitungan Matriks Cost

Fungsi ini merupakan implementasi desain fungsi pada Subbab 3.2.14 yang menghitung matriks nilai *cost* asosiasi deteksi ke objek yang menghilang berdasarkan pasangan objek gamma-nya. Kemudian diimplementasikan juga sebagian desain fungsi Subbab 3.2.16 yang menambahkan matriks *cost* untuk kasus d_0. Implementasi terdapat pada Kode Sumber 4.22a dan 4.22b.

```
def calculate_assignment_cost_matrix(
    missing_objects: List[ObjectState],
    unassigned_detections: List[DetectionState],
    updated_objects: List[ObjectState],
    default_cost_d0: float = 4.0,
) -> np.ndarray:
    match_s_gamma = matching_s_gamma(
        updated_objects=updated_objects,
        missing_objects=missing_objects,
    )
    costs = []
    for (
        obj_idx,
        (missing_object, match_s),
    ) in enumerate(
        zip(missing_objects, match_s_gamma)
    ):
        row = []
        for (
            det_idx,
            unassigned_detection,
        ) in enumerate(unassigned_detections):
            value = assignment_cost(
                missing_object,
                unassigned_detection,
                match_s,
            )
            row.append(value)
        costs.append(row)
    costs = np.array(costs)
```

Kode Sumber 4.22a Fungsi Perhitungan Matriks Cost (1)

```
31      n_objects = len(missing_objects)
32      costs_d0 = np.eye(n_objects)
33      costs_d0[costs_d0 == 0.0] = np.inf
34      costs_d0[costs_d0 == 1.0] = default_cost_d0
35      cost_augmented = np.hstack((costs, costs_d0))
36      return cost_augmented
```

Kode Sumber 4.22b Fungsi Perhitungan Matriks Cost (2)

4.2.8.4. Implementasi Fungsi Penentuan Asosiasi

Fungsi ini merupakan implementasi dari desain fungsi Subbab 3.2.12 dan 3.2.16. Dalam fungsi ini diiimplementasikan proses SCOR secara keseluruhan dan penentuan pasangan asosiasi menggunakan *linear sum assignment*. Implementasi terdapat pada Kode Sumber 4.23a dan 4.23b.

```
 1 def best_assignment(
 2     missing_objects: List[ObjectState],
 3     unassigned_detections: List[DetectionState],
 4     updated_objects: List[ObjectState],
 5     default_cost_d0: float = 4.0,
 6 ) -> np.ndarray:
 7     cost_matrix =
           calculate_assignment_cost_matrix(
 8         missing_objects=missing_objects,
 9         unassigned_detections=
               unassigned_detections,
10         updated_objects=updated_objects,
11         default_cost_d0=default_cost_d0,
12     )
13     max_val = np.max(
14         cost_matrix[cost_matrix != np.inf]
15     )
16     cost_matrix[cost_matrix == np.inf] = (
17         max_val * 2
18     )
```

Kode Sumber 4.23a Fungsi Penentuan Asosiasi (1)

```
    rows, cols = linear_sum_assignment(
        cost_matrix
    )
    n_unassigned_det = len(unassigned_detections)
    n_missing_objects = len(missing_objects)
    assignment_matrix = np.zeros(
        (
            n_missing_objects,
            n_unassigned_det + n_missing_objects,
        ),
        dtype="int",
    )
    assignment_matrix[rows, cols] = 1

    # exclude column for d0
    assignment_matrix = assignment_matrix[
        :, :n_unassigned_det
    ]
    return assignment_matrix
```

Kode Sumber 4.23b Fungsi Penentuan Asosiasi (2)

4.2.9. Implementasi Class ObjectStateTracker

Class OBJECTSTATETRACKER merupakan *class* yang menjadi bagian dari implementasi desain fungsi Subbab 3.2.17 untuk bagian *Kalman filter* dan pembaruan histogram. *Class* ini memiliki satu *class* variabel yaitu bilangan bulat *counter* yang nantinya digunakan untuk mengisi *id* pada tiap *instance*. Juga terdapat beberapa variabel *instance* yang dijelaskan pada Tabel 4.7. Implementasi terdapat pada Kode Sumber 4.24.

Tabel 4.7 Nama dan Penjelasan Variabel Instance dari Class OBJECTSTATETRACKER

Nama Variabel	Penjelasan
id	nomor unik identitas objek
kf	*instance* KALMANFILTER dari pustaka *filterpy* untuk menjalankan proses *Kalman filter*
frame_step	ID *frame* terkini
histogram	nilai histogram objek
hist_alpha	tingkat perubahan histogram
__state	*instance* OBJECTSTATE yang menyimpan nilai terbaru *state* objek

```
 1 class ObjectStateTracker:
 2     counter = 0
 3     def __init__(self, state: ObjectState):
 4     ...
 5     def update(self, detection: DetectionState):
 6     ...
 7     def __update_state(
 8         self, detection: DetectionState
 9     ):
10     ...
11     def __update_histogram(
12         self, detection: DetectionState
13     ):
14     ...
15     def predict(self):
16     ...
17     def __set_state(self):
18     ...
19     @property
20     def state(self) -> ObjectState:
21     ...
```

Kode Sumber 4.24 *Class* OBJECTSTATETRACKER

4.2.9.1. Implementasi Fungsi Constructor ObjectStateTracker

Fungsi ini berguna untuk membuat *instance* OBJECTSTATETRACKER baru berdasarkan satu OBJECTSTATE. Dalam fungsi ini juga dilakukan inisialisasi variabel *instance*. Dalam fungsi ini juga dilakukan inisialisasi nilai matriks *Kalman filter* sesuai Subbab 3.2.17. Implementasi terdapat pada Kode Sumber 4.25a dan 4.25b.

```
def __init__(self, state: ObjectState):
    self.id = ObjectStateTracker.counter
    ObjectStateTracker.counter += 1

    self.kf = KalmanFilter(
        dim_x=DIM_X, dim_z=DIM_Z
    )

    F = np.eye(DIM_X)
    F[[0, 1], [2, 3]] = 1
    self.kf.F = F

    H = np.zeros((DIM_Z, DIM_X))
    H[[0, 1, 2, 3], [0, 1, 4, 5]] = 1
    self.kf.H = H

    quart_q = 0.25 * DEV_Q
    half_q = 0.5 * DEV_Q
    self.kf.Q = np.array(
        [
            [quart_q, 0, half_q, 0, 0, 0],
            [0, quart_q, 0, half_q, 0, 0],
            [half_q, 0, DEV_Q, 0, 0, 0],
            [0, half_q, 0, DEV_Q, 0, 0],
            [0, 0, 0, 0, DEV_S, 0],
            [0, 0, 0, 0, 0, DEV_S],
        ]
    )
```

Kode Sumber 4.25a Fungsi Constructor ObjectStateTracker (1)

```
29      self.kf.R = block_diag(
30          DEV_X, DEV_Y, DEV_W, DEV_H
31      )
32      self.kf.x = np.array(
33          [
34              state.x,
35              state.y,
36              state.v_x,
37              state.v_y,
38              state.width,
39              state.height,
40          ]
41      )
42
43      self.frame_step = state.frame_step
44      self.histogram = state.histogram.copy()
45      self.hist_alpha = 0.1
46      self.__state = None
47      self.__set_state()
```

Kode Sumber 4.25b Fungsi Constructor ObjectStateTracker (2)

4.2.9.2. Implementasi Fungsi Pembaruan State Objek

Fungsi ini merupakan implementasi proses pembaruan nilai *state* objek dari deteksi yang diasosiasikan dengan menggunakan *Kalman filter*. Nilai pengukuran yang menjadi masukan *Kalman filter* sesuai dengan Subbab 3.2.17. Fungsi ini dipanggil oleh fungsi UPDATE yang kemudian memanggil __SET_STATE untuk mengisikan variabel *instance* *__state* dengan nilai terbaru dari objek hasil prediksi *Kalman filter*. Implementasi terdapat pada Kode Sumber 4.26.

```
def __update_state(
    self, detection: DetectionState
):
    measurement = np.array(
        [
            detection.x,
            detection.y,
            detection.width,
            detection.height,
        ]
    )
    self.kf.update(measurement)
```

Kode Sumber 4.26 Fungsi Pembaruan State Objek

4.2.9.3. Implementasi Fungsi Pembaruan Histogram Objek

Fungsi ini merupakan implementasi tahap pembaruan histogram objek sesuai desain fungsi pada Subbab 3.2.17 yang menggunakan Persamaan 2.25 untuk memperbarui histogram objek berdasarkan histogram deteksi. Implementasi terdapat pada Kode Sumber 4.27.

```
def __update_histogram(
    self, detection: DetectionState
):
    old_value = (
        1 - self.hist_alpha
    ) * self.histogram
    current_value = (
        self.hist_alpha * detection.histogram)
    self.histogram = old_value + current_value
```

Kode Sumber 4.27 Fungsi Pembaruan Histogram Objek

4.2.9.4. Implementasi Fungsi Prediksi State Objek

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.19 yang melakukan prediksi nilai *state* objek dari hasil

pembaruan sebelumnya. Implementasi terdapat pada Kode Sumber 4.28.

```
1 def predict(self):
2     self.kf.predict()
3     self.__set_state()
```

Kode Sumber 4.28 Fungsi Prediksi State Objek

4.2.10. Implementasi Class StructuralConstraintTracker

Class STRUCTURALCONSTRAINTTRACKER merupakan *class* yang menjadi implementasi dari tahapan pembaruan batasan struktural dari Subbab 3.2.18. Terdapat variabel *instance* yang dijelaskan pada Tabel 4.8. Implementasi ada pada Kode Sumber 4.29a dan 4.29b.

Tabel 4.8 Nama dan Penjelasan Variabel Instance dari Class STRUCTURALCONSTRAINTTRACKER

Nama Variabel	Penjelasan
kf	*instance* KALMANFILTER dari pustaka *filterpy* untuk menjalankan proses *Kalman filter*
__state	*instance* OBJECTSTATE yang menyimpan nilai terbaru *state* objek
__is_tracked	*flag* pasangan objek terlacak

```
1 class StructuralConstraintTracker:
2     def __init__(
3         self,
4         initial_state: StructuralConstraint
5     ):
6     ...
```

Kode Sumber 4.29a *Class* STRUCTURALCONSTRAINTTRACKER (1)

```
 7      def update(
 8          self,
 9          sc: Optional[StructuralConstraint] = None,
10      ):
11      ...
12      def __update_well_tracked(
13          self, sc: StructuralConstraint
14      ):
15      ...
16      def __update_missing(self):
17      ...
18      def predict(self):
19      ...
20      def __set_state(self):
21      ...
22      @property
23      def state(self) -> StructuralConstraint:
24      ...
```

Kode Sumber 4.29b *Class* STRUCTURALCONSTRAINTTRACKER (2)

4.2.10.1. Implementasi Fungsi Constructor StructuralConstraintTracker

Fungsi ini berguna untuk membuat *instance* STRUCTURALCONSTRAINTTRACKER baru berdasarkan satu STRUCTURALCONSTRAINT. Dalam fungsi ini dilakukan inisialisasi variabel *instance*. Dalam fungsi ini juga dilakukan inisialisasi nilai matriks *Kalman filter* sesuai Subbab 3.2.18. Implementasi terdapat pada Kode Sumber 4.30.

```
def __init__(
    self, initial_state: StructuralConstraint
):
    self.kf = KalmanFilter(
        dim_x=DIM_X, dim_z=DIM_Z
    )

    F = np.eye(DIM_X)
    F[[0, 1], [2, 3]] = 1
    self.kf.F = F

    H = np.zeros((DIM_Z, DIM_X))
    H[[0, 1], [0, 1]] = 1
    self.kf.H = H
    quart_sc = 0.25 * DEV_SC
    half_sc = 0.5 * DEV_SC
    self.kf.Q = np.array(
        [
            [quart_sc, 0, half_sc, 0],
            [0, quart_sc, 0, half_sc],
            [half_sc, 0, DEV_SC, 0],
            [0, half_sc, 0, DEV_SC],
        ]
    )

    self.kf.R = block_diag(DEV_X, DEV_Y)
    self.kf.x = np.array(
        [
            initial_state.delta_x,
            initial_state.delta_y,
            initial_state.delta_v_x,
            initial_state.delta_v_y,
        ]
    )
    self.__is_tracked = True
    self.__state = None
    self.__set_state()
```

Kode Sumber 4.30 Fungsi Constructor StructuralConstraintTracker

4.2.10.2. Implementasi Fungsi Pembaruan State Batasan Struktural

Fungsi ini merupakan bagian dari implementasi Subbab 3.2.18 untuk memperbarui nilai *state* batasan struktural menggunakan *Kalman filter* untuk yang berasal dari pasangan objek terlacak. Untuk batasan struktural yang salah satu atau kedua objeknya gagal terlacak akan ditandai dengan *flag __is_tracked*. Implementasi terdapat pada Kode Sumber 4.31.

```
 1 def update(
 2     self,
 3     sc: Optional[StructuralConstraint] = None,
 4 ):
 5     if sc:
 6         self.__update_well_tracked(sc)
 7     else:
 8         self.__update_missing()
 9
10     self.__set_state()
11
12 def __update_well_tracked(
13     self,
14     sc: StructuralConstraint,
15 ):
16     measurement = np.array(
17         [
18             sc.delta_x,
19             sc.delta_y
20         ]
21     )
22     self.kf.update(measurement)
23     self.__is_tracked = True
24
25 def __update_missing(self):
26     self.__is_tracked = False
```

Kode Sumber 4.31 Fungsi Pembaruan State Batasan Struktural

4.2.10.3. Implementasi Fungsi Prediksi State Batasan Struktural

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.20 yang menghitung prediksi nilai batasan struktural terkini sesuai *flag __is_tracked*. Implementasi terdapat pada Kode Sumber 4.32.

```
 1 def predict(self):
 2     if self.__is_tracked:
 3         self.kf.predict()
 4     else:
 5         column_vector_x = self.kf.x.reshape(
 6             (-1, 1)
 7         )
 8         self.kf.x = np.dot(
 9             self.kf.F, column_vector_x
10         )
11     self.__set_state()
```

Kode Sumber 4.32 Fungsi Prediksi State Batasan Struktural

4.2.11. Implementasi Class Tracker

Class TRACKER merupakan *class* yang menjadi implementasi dari desain fungsi *object tracker* pada Subbab 3.2.2. *Class* ini mengimplementasikan alur data antara proses SCEA dan SCOR, pembaruan menggunakan OBJECTSTATETRACKER dan STRUCTURALCOSNTRAINTTRACKER, eliminasi objek, dan pembuatan objek baru dari deteksi. *Class* ini memiliki beberapa variabel *instance* yang dijelaskan pada Tabel 4.9. Implementasi terdapat pada Kode Sumber 4.33a, 4.33b dan 4.33c.

Tabel 4.9 Nama dan Penjelasan Variabel Instance dari Class TRACKER

Nama Variabel	Penjelasan
__max_age	batas lama objek menghilang
__scea_config	parameter untuk SCEA
__scor_config	parameter untuk SCOR
object_states	*array* semua objek
__structural_constraints	*array* semua batasan struktural
__object_trackers	*array* semua *tracker* objek
__sc_trackers	*array* semua *tracker* batasan struktural
__well_tracked_indexes	*array* indeks objek terlacak
__missing_indexes	*array* indeks objek menghilang
__last_updates	*array* ID *frame* terakhir objek terlacak
__unassigned_detections	*sliding window* untuk inisialisasi objek baru
__first	*flag* untuk inisialisasi saat *update*
__hit_streak	menyimpan *counter* objek terlacak secara beruntun
__frame_count	menyimpan ID *frame* terkini
__min_streak	batas minimal terlacak secara beruntun

```
1 class Tracker:
2     def __init__(
3         self,
4         max_age: int,
5         min_streak: int,
6         scea_config: scea.ConfigSCEA,
7         scor_config: scor.ConfigSCOR,): ...
```

Kode Sumber 4.33a *Class* TRACKER (1)

```
    def update(
        self,
        detections: List[DetectionState],
        current_frame_step: int,
    ): ...
    def __update_first_time(
        self,
        detections: List[DetectionState],
        current_frame_step: int,
    ): ...
    def __update_routine(
        self,
        detections: List[DetectionState],
        current_frame_step: int,
    ): ...
    def __process_scea(
        self, detections: List[DetectionState]
    ) -> Tuple[List[int], List[int], List[int]]:
    ...
    def __process_scor(
        self,
        unassigned_detection_index: List[int],
        detections: List[DetectionState],
    ) -> Tuple[List[int], List[int], List[int]]:
    ...
    def __update_object_from_assigned_detection(
        self,
        detections: List[DetectionState],
        tracked_object_indexes: List[int],
        assigned_detection_indexes: List[int],
    ): ...
    def __update_object_trackers(
        self,
        tracked_object_indexes: List[int],
        assigned_detection_indexes: List[int],
        detections: List[DetectionState],
    ): ...
```

Kode Sumber 4.33b *Class* TRACKER (2)

```
46      def __update_sc_trackers(
47          self,
48          tracked_object_indexes: List[int],
49          assigned_detection_indexes: List[int],
50          detections: List[DetectionState],
51      ): ...
52      def __predict_object_trackers(self): ...
53      def __predict_sc_trackers(self): ...
54      def __update_last_updates(
55          self, current_frame_step: int
56      ): ...
57      def __remove_missing_objects(
58          self, current_frame_step: int
59      ): ...
60      def __select_detections(
61          self,
62          unassigned_detections: List[
63              DetectionState
64          ],
65      ) -> List[DetectionState]: ...
66      def __create_new_well_tracked_objects(
67          self,
68          unassigned_detections: List[
69              DetectionState
70          ],
71          current_frame_step: int,
72      ): ...
```

Kode Sumber 4.33c *Class* TRACKER (3)

4.2.11.1. Implementasi Fungsi Constructor Tracker

Fungsi ini berguna untuk membuat *instance* TRACKER baru. Dalam fungsi ini juga dilakukan inisialisasi untuk beberapa variabel *instance*. Implementasi terdapat pada Kode Sumber 4.34.

```
def __init__(
    self,
    max_age: int,
    min_streak: int,
    scea_config: scea.ConfigSCEA,
    scor_config: scor.ConfigSCOR,
):
    self.__max_age = max_age
    self.__scea_config = scea_config
    self.__scor_config = scor_config
    self.__object_states: List[
        ObjectState
    ] = []
    self.__structural_constraints: List[
        List[StructuralConstraint]
    ] = [[]]
    self.__object_trackers: List[
        ObjectStateTracker
    ] = []
    self.__sc_trackers: List[
        List[StructuralConstraintTracker]
    ] = []
    self.__well_tracked_indexes: List[
        int
    ] = []
    self.__missing_indexes: List[int] = []
    self.__last_updates: List[int] = []
    self.__unassigned_detections: List[
        List[DetectionState]
    ] = [[]]
    self.__first = True
    self.__hit_streak: List[int] = []
    self.__frame_count = 0
    self.__min_streak = min_streak
```

Kode Sumber 4.34 Fungsi Constructor Tracker

4.2.11.2. Implementasi Fungsi Pemroses SCEA

Fungsi ini merupakan bagian dari implementasi desain fungsi pada Subbab 3.2.3 yang mengintegrasikan proses SCEA dengan *class* TRACKER. Dalam fungsi ini dilakukan persiapan masukan SCEA, pengolahan hasil matriks asosiasi, pembaruan sementara S_w untuk digunakan di SCOR, dan pembaruan nilai indeks untuk objek terlacak dan menghilang dari hasil SCEA. Implementasi terdapat pada Kode Sumber 4.35a, 4.35b, dan 4.35c.

```
def __process_scea(
    self, detections: List[DetectionState]
) -> Tuple[List[int], List[int], List[int]]:
    if len(self.__well_tracked_indexes) == 0:
        return (np.array([]), np.array([]),
            np.arange(len(detections)))
    object_states = self.__object_states[
        self.__well_tracked_indexes
    ]
    structural_constraints =
         self.__structural_constraints[
        self.__well_tracked_indexes, :
    ]
    structural_constraints =
         structural_constraints[
        :, self.__well_tracked_indexes
    ]
    assignment_matrix = scea.best_assignment(
        object_states=object_states,
        detection_states=detections,
        structural_constraints=
             structural_constraints,
        default_cost_d0=
             self.__scea_config.default_cost_d0,
        gating_threshold=
             self.__scea_config.gating_threshold,
        num_cluster_member=
             self.__scea_config.num_cluster_member,
    )
```

Kode Sumber 4.35a Fungsi Pemroses SCEA (1)

```
24      (
25          tracked_object_indexes,
26          assigned_detection_indexes,
27      ) = np.where(assignment_matrix == 1)
28
29      self.__update_object_from_assigned_detection(
30          detections=detections,
31          tracked_object_indexes=
                tracked_object_indexes,
32          assigned_detection_indexes=
                assigned_detection_indexes,
33      )
34      actual_tracked_object_index =
             self.__well_tracked_indexes[
35          tracked_object_indexes
36      ]
37
38      unassigned_detections_index =
             scea.get_missing_detections(
39          assignment_matrix
40      )
41      relative_missing_objects_index =
             scea.get_missing_objects(
42          assignment_matrix
43      )
44      missing_object_index =
             self.__well_tracked_indexes[
45          relative_missing_objects_index
46      ]
47
48      # update index well-tracked object
49      self.__missing_indexes = np.concatenate(
50          (
51              self.__missing_indexes,
52              missing_object_index,
53          )
54      )
```

Kode Sumber 4.35b Fungsi Pemroses SCEA (2)

```
55      self.__missing_indexes =
             self.__missing_indexes.astype(
56          "int32"
57      )
58      self.__well_tracked_indexes = (
59          actual_tracked_object_index
60      )
61
62      return (
63          actual_tracked_object_index,
64          assigned_detection_indexes,
65          unassigned_detections_index,
66      )
```

Kode Sumber 4.35c Fungsi Pemroses SCEA (3)

4.2.11.3. Implementasi Fungsi Pemroses SCOR

Fungsi ini merupakan bagian dari implementasi desain fungsi pada Subbab 3.2.12 yang mengintegrasikan proses SCOR dengan *class* TRACKER. Dalam fungsi ini dilakukan persiapan masukan SCOR, pengolahan hasil matriks asosiasi, dan pembaruan nilai indeks untuk objek terlacak dan menghilang dari hasil SCOR. Implementasi terdapat pada Kode Sumber 4.36a, 4.36b, dan 4.36c.

```
 1 def __process_scor(
 2     self,
 3     unassigned_detection_index: List[int],
 4     detections: List[DetectionState],
 5 ) -> Tuple[List[int], List[int], List[int]]:
 6     if (
 7         len(self.__missing_indexes) == 0
 8         or len(self.__well_tracked_indexes)
 9         == 0
10         or len(unassigned_detection_index)
11         == 0
12     ):
```

Kode Sumber 4.36a Fungsi Pemroses SCOR (1)

```
13          return (
14              np.array([]),
15              np.array([]),
16              unassigned_detection_index,
17          )
18      unassigned_detections = detections[
19          unassigned_detection_index
20      ]
21      missing_objects = self.__object_states[
22          self.__missing_indexes
23      ]
24      updated_objects = self.__object_states[
25          self.__well_tracked_indexes
26      ]
27      assignment_matrix = scor.best_assignment(
28          missing_objects=missing_objects,
29          unassigned_detections=
                unassigned_detections,
30          updated_objects=updated_objects,
31          default_cost_d0=
                self.__scor_config.default_cost_d0,
32      )
33      (
34          tracked_object_indexes,
35          relative_assigned_detection_indexes,
36      ) = np.where(assignment_matrix == 1)
37      assigned_detection_indexes =
            unassigned_detection_index[
38          relative_assigned_detection_indexes
39      ]
40
41      actual_tracked_object_index =
            self.__missing_indexes[
42          tracked_object_indexes
43      ]
```

Kode Sumber 4.36b Fungsi Pemroses SCOR (2)

```
44      relative_unassigned_detections_index =
             scea.get_missing_detections(
45          assignment_matrix
46      )
47      unassigned_detection_index =
             unassigned_detection_index[
48          relative_unassigned_detections_index
49      ]
50
51      relative_missing_objects_index =
             scea.get_missing_objects(
52          assignment_matrix
53      )
54      missing_object_index = self.__missing_indexes[
55          relative_missing_objects_index
56      ]
57      # update index well-tracked object
58      self.__well_tracked_indexes = np.concatenate(
59          (
60              self.__well_tracked_indexes,
61              actual_tracked_object_index,
62          )
63      )
64      self.__well_tracked_indexes =
             self.__well_tracked_indexes.astype(
65          "int32"
66      )
67      self.__missing_indexes = (
68          missing_object_index
69      )
70
71      return (
72          actual_tracked_object_index,
73          assigned_detection_indexes,
74          unassigned_detection_index,
75      )
```

Kode Sumber 4.36c Fungsi Pemroses SCOR (3)

4.2.11.4. Implementasi Fungsi Update Objek Terlacak dengan Deteksi

Fungsi ini merupakan bagian dari implementasi desain fungsi pada Subbab 3.2.3 yang memperbarui objek terlacak dengan deteksinya pada akhir proses SCEA. Implementasi terdapat pada Kode Sumber 4.37.

```
def __update_object_from_assigned_detection(
    self,
    detections: List[DetectionState],
    tracked_object_indexes: List[int],
    assigned_detection_indexes: List[int],
):
    object_states = self.__object_states[
        self.__well_tracked_indexes
    ]
    for object_idx, detection_idx in zip(
        tracked_object_indexes,
        assigned_detection_indexes,
    ):
        detection_state = detections[
            detection_idx
        ]
        object_state = object_states[
            object_idx
        ]
        object_state.update_from_detection(
            detection_state
        )
```

Kode Sumber 4.37 Fungsi Update Objek Terlacak dengan Deteksi

4.2.11.5. Implementasi Fungsi Update ObjectTracker

Fungsi ini merupakan bagian dari implementasi desain fungsi pada Subbab 3.2.17. Dalam fungsi ini dilakukan pembaruan pada *tracker* dan nilai *state* objek sesuai hasil *Kalman filter*. Implementasi terdapat pada Kode Sumber 4.38.

```
def __update_object_trackers(
    self,
    tracked_object_indexes: List[int],
    assigned_detection_indexes: List[int],
    detections: List[DetectionState]
):
    for object_index, detection_index in zip(
        tracked_object_indexes,
        assigned_detection_indexes,
    ):
        detection = detections[
            detection_index
        ]
        tracker = self.__object_trackers[
            object_index
        ]
        tracker.update(detection)
        object_state = self.__object_states[
            object_index
        ]
        object_state.update_from_object_state(
            tracker.state
        )
```

Kode Sumber 4.38 Fungsi Update ObjectTracker

4.2.11.6. Implementasi Fungsi Update StructuralConstraint-Tracker

Fungsi ini merupakan bagian dari implementasi desain fungsi pada Subbab 3.2.18. Dalam fungsi ini dilakukan pembaruan batasan struktural menggunakan *Kalman filter*. Implementasi terdapat pada Kode Sumber 4.39a dan 4.39b.

```
def __update_sc_trackers(
    self, tracked_object_indexes: List[int],
    assigned_detection_indexes: List[int],
    detections: List[DetectionState],
):
    objects_from_detections = [
        ObjectState.from_detection(
            detections[i]
        )
        for i in assigned_detection_indexes
    ]
    for i, object_index_i in enumerate(
        tracked_object_indexes
    ):
        for j, object_index_j in enumerate(
            tracked_object_indexes
        ):
            if i == j:
                continue
            object_i = objects_from_detections[i]
            object_j = objects_from_detections[j]
            sc = StructuralConstraint.create(
                object_i, object_j
            )
            tracker = self.__sc_trackers[
                object_index_i
            ][object_index_j]
            tracker.update(sc)
            saved_sc =
                 self.__structural_constraints[
                object_index_i
            ][object_index_j]
            saved_sc.update_from_sc(
                tracker.state
            )
```

Kode Sumber 4.39a Fungsi Update StructuralConstraintTracker (1)

```
35      # update one or both missing
36      mask = np.ix_(
37          tracked_object_indexes,
38          tracked_object_indexes,
39      )
40      bool_mask = np.ones_like(
41          self.__sc_trackers, dtype="bool")
42      bool_mask[mask] = 0
43      missing_sc_trackers = self.__sc_trackers[
44          bool_mask]
45      missing_sc = self.__structural_constraints[
46          bool_mask]
47
48      for tracker, sc in zip(
49          missing_sc_trackers, missing_sc
50      ):
51          if tracker:
52              tracker.update()
53              sc.update_from_sc(tracker.state)
```

Kode Sumber 4.39b Fungsi Update StructuralConstraintTracker (2)

4.2.11.7. Implementasi Fungsi Update Counter Frame Terakhir Terlacak dan Terlacak Beruntun

Fungsi ini merupakan bagian dari implementasi desain fungsi pada Subbab 3.2.21 dan 3.2.23 yang memperbarui nilai *counter frame* terakhir terlacak untuk objek yang berhasil dilacak dan *counter* terlacak secara beruntun. Implementasi terdapat pada Kode Sumber 4.40.

```
def __update_last_updates(
    self, current_frame_step: int
):
    self.__last_updates[
        self.__well_tracked_indexes
    ] = current_frame_step
    self.__hit_streak[
        self.__well_tracked_indexes
    ] += 1
    self.__hit_streak[
        self.__missing_indexes
    ] = 0
```

Kode Sumber 4.40 Fungsi Update Counter Frame Terakhir Terlacak dan Terlacak Beruntun

4.2.11.8. Implementasi Fungsi Eliminasi Objek

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.21. Dari semua objek, dihitung selisih *frame* terakhir berhasil dilacak dengan *frame* sekarang dan indeks objek yang melebihi batas *max_age* ditandai dalam *mask*. Implementasi terdapat pada Kode Sumber 4.41a dan 4.41b.

```
def __remove_missing_objects(
    self, current_frame_step: int
):
    diff = (
        current_frame_step
        - self.__last_updates
    )
    removed_index = np.where(
        diff > self.__max_age
    )[0].flatten()
    if len(removed_index) == 0:
        return
```

Kode Sumber 4.41a Fungsi Eliminasi Objek (1)

```
13      mask = np.ones(
14          self.__object_states.shape, bool
15      )
16      mask[removed_index] = 0
17      self.__object_states = self.__object_states[
18          mask
19      ]
20      self.__object_trackers =
             self.__object_trackers[
21          mask
22      ]
23      self.__last_updates = self.__last_updates[
24          mask
25      ]
26      self.__hit_streak = self.__hit_streak[
27          mask
28      ]
29      self.__structural_constraints =
             self.__structural_constraints[
30          mask, :
31      ]
32      self.__structural_constraints =
             self.__structural_constraints[
33          :, mask
34      ]
35      self.__sc_trackers = self.__sc_trackers[
36          mask, :
37      ]
38      self.__sc_trackers = self.__sc_trackers[
39          :, mask
40      ]
41      self.__well_tracked_indexes = reset_index(
42          self.__well_tracked_indexes,
43          removed_index,
44      )
45      self.__missing_indexes = reset_index(
46          self.__missing_indexes, removed_index
47      )
```

Kode Sumber 4.41b Fungsi Eliminasi Objek (2)

4.2.11.9. Implementasi Fungsi Prediksi ObjectStateTracker

Fungsi ini merupakan bagian dari implementasi desain fungsi pada Subbab 3.2.19 yang memprediksi nilai objek dari hasil pembaruan sebelumnya. Implementasi terdapat pada Kode Sumber 4.42.

```
def __predict_object_trackers(self):
    for (track, obj) in zip(
        self.__object_trackers,
        self.__object_states,
    ):
        track.predict()
        obj.update_from_object_state(
            track.state
        )
```

Kode Sumber 4.42 Fungsi Prediksi ObjectStateTracker

4.2.11.10. Implementasi Fungsi Prediksi StructuralConstraintTracker

Fungsi ini merupakan bagian dari implementasi desain fungsi pada Subbab 3.2.20 yang memprediksi nilai batasan struktural dari hasil pembaruan sebelumnya. Implementasi terdapat pada Kode Sumber 4.43.

```
def __predict_sc_trackers(self):
    for (track_row, sc_row) in zip(
        self.__sc_trackers,
        self.__structural_constraints,
    ):
        for (track, sc) in zip(
            track_row, sc_row
        ):
            if track is None:
                continue
            track.predict()
            sc.update_from_sc(track.state)
```

Kode Sumber 4.43 Fungsi Prediksi StructuralConstraintTracker

4.2.11.11. Implementasi Fungsi Pembuatan Objek Metode I

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.22 untuk metode I. Dalam fungsi ini, objek baru dibuat berdasarkan deteksi yang belum diasosiasikan. Dibuat juga *tracker* untuk objek dan batasan struktural baru. Implementasi terdapat pada Kode Sumber 4.44a dan 4.44b.

```
def __create_new_well_tracked_objects(
    self,
    unassigned_detections: List[
        DetectionState
    ],
    current_frame_step: int,
):
    selected_detections = (
        unassigned_detections)
    n_new_objects = len(selected_detections)
    if n_new_objects == 0:
        return
    new_object_states = [
        ObjectState.from_detection(detection)
        for detection in selected_detections
    ]
    new_object_trackers = [
        ObjectStateTracker(object_state)
        for object_state in new_object_states
    ]
    new_object_index = np.arange(
        n_new_objects
    ) + len(self.__object_states)
    self.__well_tracked_indexes = np.concatenate(
        (self.__well_tracked_indexes,
        new_object_index))
    new_last_update = [
        current_frame_step] * n_new_objects
    self.__last_updates = np.concatenate(
        (self.__last_updates, new_last_update))
```

Kode Sumber 4.44a Fungsi Pembuatan Objek Metode I (1)

```
31      new_hit_streak = [1] * n_new_objects
32      self.__hit_streak =
             np.concatenate((self.__hit_streak,
             new_hit_streak)).astype(int)
33
34      sc_new = create_sc(
35          self.__object_states,
36          new_object_states,
37          self.__structural_constraints,
38      )
39      sc_tracker_new = create_sc_tracker(sc_new)
40
41      self.__structural_constraints = sc_new
42      self.__sc_trackers = sc_tracker_new
43
44      self.__object_states = np.concatenate(
45          (self.__object_states,
46              new_object_states))
47      self.__object_trackers = np.concatenate(
48          (self.__object_trackers,
49              new_object_trackers))
```

Kode Sumber 4.44b Fungsi Pembuatan Objek Metode I (2)

4.2.11.12. Implementasi Fungsi Pembuatan Objek Metode II

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.22 untuk metode II yang menggunakan *sliding window*. Dari *sliding window* di variabel *__unassigned_detections* dihitung IoU kemudian dicari pasangannya dengan *linear sum assignment*. Implementasi terdapat pada Kode Sumber 4.45a dan 4.45b.

```
1 ...
2 time_window = len(self.__unassigned_detections)
3 if time_window < 2:
4     return selected_detections
5
6 iou_scores = []
```

Kode Sumber 4.45a Fungsi Pembuatan Objek Metode II (1)

```
 7 for (
 8     old_detection
 9 ) in self.__unassigned_detections[0]:
10     row = []
11     for (
12         new_detection
13     ) in self.__unassigned_detections[1]:
14         bb_old = state_to_bb(old_detection)
15         bb_new = state_to_bb(new_detection)
16         iou_score = iou(bb_old, bb_new)
17         row.append(iou_score)
18     iou_scores.append(row)
19
20 iou_scores = np.array(iou_scores)
21
22 min_iou = 0.5
23 olds, news = linear_sum_assignment(
24     -iou_scores)
25
26 selected_index = []
27 for old_index, new_index in zip(
28     olds, news
29 ):
30     if (
31         iou_scores[old_index][new_index]
32         < min_iou
33     ):
34         continue
35     selected_index.append(new_index)
36 selected_detections = unassigned_detections[
37     selected_index].tolist()
38 ...
```

Kode Sumber 4.45b Fungsi Pembuatan Objek Metode II (2)

4.2.11.13. Implementasi Fungsi Update Tracker

Fungsi ini merupakan implementasi dari desain fungsi pada Subbab 3.2.2 yang menjalankan alur *object tracking* seca-

ra keseluruhan dari proses SCEA, SCOR, hingga seleksi dengan min_streak. Fungsi ini yang menjadi antarmuka antara program MAIN dengan program *object tracking*. Implementasi terdapat pada Kode Sumber 4.46a, 4.46b, 4.46c, 4.46d, 4.46e, dan 4.46f.

```
def update(
    self,
    detections: List[DetectionState],
    current_frame_step: int,
):
    if not isinstance(detections, np.ndarray):
        detections = np.array(detections)
    self.__frame_count += 1
    if self.__first:
        return self.__update_first_time(
            detections, current_frame_step
        )
    return self.__update_routine(
        detections, current_frame_step
    )

def __update_first_time(
    self,
    detections: List[DetectionState],
    current_frame_step: int,
):
    self.__object_states = np.array(
        [
            ObjectState.from_detection(
                detection
            )
            for detection in detections
        ],
        dtype=object,
    )
    self.__object_trackers = np.array(
        [
            ObjectStateTracker(object_state)
```

Kode Sumber 4.46a Fungsi Update Tracker (1)

```
34              for object_state in
                     self.__object_states
35          ],
36          dtype=object,
37      )
38
39      structural_constraints =
             calculate_structural_constraint(
40          self.__object_states
41      )
42      if structural_constraints == []:
43          self.__structural_constraints = np.array(
44              [[]], dtype=object
45          )
46      else:
47          self.__structural_constraints = np.array(
48              structural_constraints,
49              dtype=object,
50          )
51      sc_trackers = [
52          [
53              None
54              for _ in range(
55                  len(self.__object_states)
56              )
57          ]
58          for _ in range(
59              len(self.__object_states)
60          )
61      ]
62      for i, row in enumerate(
63          self.__structural_constraints
64      ):
65          for (
66              j,
67              structural_constraint,
68          ) in enumerate(row):
69              if i == j:
```

Kode Sumber 4.46b Fungsi Update Tracker (2)

```
                    continue
                sc_trackers[i][
                    j
                ] = StructuralConstraintTracker(
                    structural_constraint
                )
        if structural_constraints == []:
            self.__sc_trackers = np.array(
                [[]], dtype=object
            )
        else:
            self.__sc_trackers = np.array(
                sc_trackers, dtype=object
            )

        self.__well_tracked_indexes = np.arange(
            len(self.__object_states), dtype=int
        )
        self.__missing_indexes = np.array(
            [], dtype=int
        )
        self.__last_updates = np.array(
            [current_frame_step]
            * len(detections),
            dtype=int,
        )
        self.__hit_streak = np.array(
            [1] * len(detections), dtype=int
        )
        self.__first = False
        return self.__object_trackers

    def __update_routine(
        self,
        detections: List[DetectionState],
        current_frame_step: int,
    ):
        self.__predict_object_trackers()
```

Kode Sumber 4.46c Fungsi Update Tracker (3)

```
108     self.__predict_sc_trackers()
109     if len(detections) == 0:
110         self.__missing_indexes = np.concatenate(
111             (
112                 self.__missing_indexes,
113                 self.__well_tracked_indexes,
114             )
115         )
116         self.__well_tracked_indexes = np.array(
117             [], dtype=int
118         )
119         self.__update_last_updates(
120             current_frame_step=current_frame_step
121         )
122         self.__remove_missing_objects(
123             current_frame_step=current_frame_step
124         )
125         return
126     (
127         scea_tracked_object_index,
128         scea_assigned_detection_index,
129         scea_unassigned_detections_index,
130     ) = self.__process_scea(detections)
131
132     (
133         scor_tracked_object_index,
134         scor_assigned_detection_index,
135         scor_unassigned_detections_index,
136     ) = self.__process_scor(
137         scea_unassigned_detections_index,
138         detections,
139     )
140
141     tracked_object_indexes = np.concatenate(
142         (
143             scea_tracked_object_index,
144             scor_tracked_object_index,
145         )).astype(int)
```

Kode Sumber 4.46d Fungsi Update Tracker (4)

```
146     assigned_detection_indexes = np.concatenate(
147         (
148             scea_assigned_detection_index,
149             scor_assigned_detection_index,
150         )
151     ).astype(
152         int
153     )
154
155     self.__update_object_trackers(
156         tracked_object_indexes=
                tracked_object_indexes,
157         assigned_detection_indexes=
                assigned_detection_indexes,
158         detections=detections,
159     )
160
161     self.__update_sc_trackers(
162         tracked_object_indexes=
                tracked_object_indexes,
163         assigned_detection_indexes=
                assigned_detection_indexes,
164         detections=detections,
165     )
166
167     self.__update_last_updates(
168         current_frame_step=current_frame_step
169     )
170     self.__remove_missing_objects(
171         current_frame_step=current_frame_step
172     )
173
174     if len(self.__unassigned_detections):
175         if len(
176             self.__unassigned_detections[0]
177         ):
178             last_saved =
                    self.__unassigned_detections[
```

Kode Sumber 4.46e Fungsi Update Tracker (5)

```
179                     0
180             ][
181                 0
182             ].frame_step
183             if (
184                 current_frame_step
185                 - last_saved
186                 > 1
187             ):
188                 del self.__unassigned_detections[
189                     0
190                 ]
191     self.__unassigned_detections =
             self.__unassigned_detections[
192         -1:
193     ]
194     self.__create_new_well_tracked_objects(
195         detections[
196             scor_unassigned_detections_index
197         ],
198         current_frame_step=current_frame_step,
199     )
200
201     mask = (
202         self.__last_updates
203         == current_frame_step
204     ) & (
205         (self.__hit_streak >= self.__min_streak)
206         | (self.__frame_count <=
                self.__min_streak)
207     )
208     return self.__object_trackers[mask]
```

Kode Sumber 4.46f Fungsi Update Tracker (6)

4.2.12. Implementasi Fungsi Main

Fungsi MAIN merupakan implementasi dari desain fungsi pada Subbab 3.2.1 yang menjadi antarmuka dari penggunaan program

object tracking. Fungsi ini akan melakukan iterasi dari *frame* ke *frame* untuk mendapatkan semua data *bounding box* dan citra satu *frame*. Kemudian diubah menjadi *instance* DETECTIONSTATE dan menjadi masukan bagi fungsi UPDATE dari TRACKER. Setelah menjalankan *object tracking* dari satu *frame*, data *state* semua objek yang terlacak dicetak ke konsol untuk nantinya menjadi masukan untuk tahap pengujian. Implementasi terdapat pada Kode Sumber 4.47a, 4.47b, dan 4.47c.

```
def main(phase: str, sequence: str):
    DATA_DIR = f"dataset/2DMOT2015/{phase}"
    BASE_IMAGE_DIR = f"{DATA_DIR}/{sequence}/img1"
    images = np.load(
        f"{DATA_DIR}/{sequence}/img.npy",
        mmap_mode="r",
    )

    if phase == "test":
        gt_filename = (
            f"{DATA_DIR}/{SEQUENCE}/det/det.csv"
        )
        min_conf = -1
    else:
        gt_filename = (
            f"{DATA_DIR}/{SEQUENCE}/det/det.csv"
        )
        min_conf = -1

    df = pd.read_csv(gt_filename)
    df = df[df["conf"] >= min_conf]
    MIN_FRAME = df["frame"].min()
    MAX_FRAME = df["frame"].max()

    config_scea = ConfigSCEA(
        gating_threshold=0.7, default_cost_d0=1.0
    )
    config_scor = ConfigSCOR(default_cost_d0=1.0)
```

Kode Sumber 4.47a Fungsi MAIN (1)

```
29      tracker = Tracker(2, config_scea, config_scor)
30      BBOX_COL = [
31          "frame",
32          "left",
33          "top",
34          "width",
35          "height",
36      ]
37      position_col = [
38          "left",
39          "top",
40          "width",
41          "height",
42      ]
43      df[position_col] = df[position_col] + 1
44      for frame_step in range(
45          max(1, MIN_FRAME), MAX_FRAME + 1
46      ):
47          bboxes = df.loc[
48              df["frame"] == frame_step, BBOX_COL
49          ]
50          detections = []
51          image = images[frame_step - 1]
52          for num, detection in enumerate(
53              bboxes.itertuples()
54          ):
55              left = int(max(0, detection.left))
56              top = int(max(0, detection.top))
57              det = DetectionState.from_bbox(
58                  left=detection.left,
59                  top=detection.top,
60                  width=detection.width,
61                  height=detection.height,
62                  frame_step=detection.frame,
63                  full_image=image,
64                  n_bins=64,
65              )
66              detections.append(det)
```

Kode Sumber 4.47b Fungsi MAIN (2)

```
67          tracker.update(detections, frame_step)
68          for (
69              obj_tracker
70          ) in tracker.well_tracked_objects:
71              state = obj_tracker.state
72              left = state.x - state.width / 2
73              top = state.y - state.height / 2
74              print(
75                  f"{state.frame_step},"
76                  f"{obj_tracker.id},"
77                  f"{left:.3f},{top:.3f},"
78                  f"{state.width:.3f},"
79                  f"{state.height:.3f},"
80                  "1,-1,-1,-1"
81              )
```

Kode Sumber 4.47c Fungsi MAIN (3)

[*Halaman ini sengaja dikosongkan*]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini dijelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan pada tugas akhir ini.

5.1. Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi sebagai berikut:

1. Perangkat Keras
   (a) Prosesor Intel® Core™ i5-6200U CPU @ 2,3GHz (2 CPUs, 4 Threads)
   (b) *Random Access Memory* 8192MB
2. Perangkat Lunak
   (a) Sistem Operasi Windows 10 Home 64-bit
   (b) Bahasa Pemrograman Python 3.6.4 64-bit
   (c) Microsoft Visual C++ 2015 (14.0)

5.2. Dataset

Pada tugas akhir ini, data yang digunakan sebagai masukan sistem *object tracking* adalah data MOTChallenge 2015 dalam format 2D atau disebut juga 2DMOT15. 2DMOT15 adalah dataset yang berisikan *frame* dari video deteksi objek pejalan kaki dengan anotasi ID lintasan hasil *tracking*. Dalam dataset ini terdapat data latih dan data uji, dengan data latih dilengkapi dengan label anotasi ID lintasan sedangkan data uji tidak disediakan label anotasi ID. Label untuk data uji disimpan di *website* dataset [4] dan hanya diberikan akses untuk evaluasi hasil *tracking*.

Pada data latih 2DMOT15, terdapat 11 video dengan total

5503 *frame*. Data anotasi tiap video berupa berkas dengan ekstensi .txt dan nilai yang dipisahkan dengan koma. Data anotasi memiliki kolom seperti pada Tabel 5.1. Detail dataset latih dapat dilihat pada Tabel 5.2a dan 5.2b. Detail dataset uji dapat dilihat pada Tabel 5.3a dan 5.3b.

Tabel 5.1 Detail Data Anotasi

Kolom	Deskripsi
FrameID	Nomor urut *frame*, mulai dari 1
ID	Nomor unik untuk tiap *track*
Left	Posisi *bounding box* kiri atas di sumbu X 2D
Top	Posisi *bounding box* kiri atas di sumbu Y 2D
Width	Lebar *bounding box*
Height	Tinggi *bounding box*
Confidence	Probabilitas klasifikasi kelas pejalan kaki
X	Posisi *bounding box* sumbu X 3D
Y	Posisi *bounding box* sumbu Y 3D
Z	Posisi *bounding box* sumbu Z 3D

Tabel 5.2a Spesifikasi Data Latih dari 2DMOT15 (1)

Nama	FPS	Resolusi	Durasi (detik)	Track	Kamera
TUD-Stadtmitte	25	640x480	179	10	statis
TUD-Campus	25	640x480	71	8	statis
PETS09-S2L1	7	768x576	795	19	statis
ETH-Bahnhof	14	640x480	1000	171	gerak
ETH-Sunnyday	14	640x480	354	30	gerak
ETH-Pedcross2	14	640x480	840	133	gerak

Tabel 5.2b Spesifikasi Data Latih dari 2DMOT15 (2)

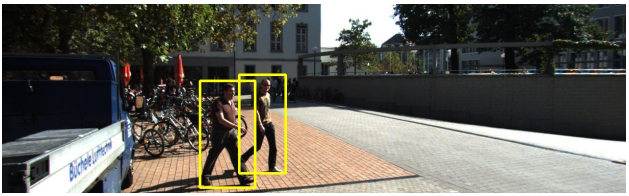
Nama	FPS	Resolusi	Durasi (detik)	Track	Kamera
ADL-Rundle-6	30	1920x1080	525	24	statis
ADL-Rundle-8	30	1920x1080	654	28	gerak
KITTI-13	10	1242x375	340	42	gerak
KITTI-17	10	1242x375	145	9	statis
Venice-2	30	1920x1080	600	26	statis

Tabel 5.3a Spesifikasi Data Uji dari 2DMOT15 (1)

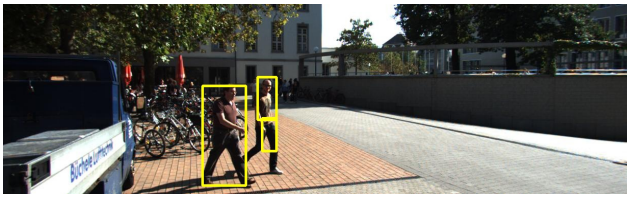
Nama	FPS	Resolusi	Durasi (detik)	Track	Kamera
TUD-Crossing	25	640x480	201	13	statis
PETS09-S2L2	7	768x576	436	42	statis
ETH-Jelmoli	14	640x480	440	45	gerak
ETH-Linthescher	14	640x480	1194	197	gerak
ETH-Crossing	14	640x480	219	26	gerak
AVG-TownCentre	2,5	1920x1080	450	133	statis
ADL-Rundle-1	30	1920x1080	500	32	gerak
ADL-Rundle-3	30	1920x1080	625	44	statis
KITTI-16	10	1242x375	209	4217	statis

Tabel 5.3b Spesifikasi Data Uji dari 2DMOT15 (2)

Nama	FPS	Resolusi	Durasi (detik)	Track	Kamera
KITTI-19	10	1242x375	1059	62	gerak
Venice-1	30	1920x1080	450	17	statis



(a)



(b)

Gambar 5.1 Perbandingan *Bounding Box* pada Video KITTI-17 *Frame* ke-1: (a) data *gt* dan (b) data *det*

Data latih 2DMOT15 memiliki dua macam data anotasi yaitu *gt* (*ground truth*) dan *det* (*raw detection*). Pada data *gt*, ID lintasan dan nilai *bounding box* yang digunakan berasal dari data *ground truth* untuk *object detection*. Pada data *det*, tidak terdapat ID lintasan yang ditandai dengan nilai negatif satu dan nilai *bounding box* yang digunakan berasal dari detektor berbasis *aggregated channel features* (ACF) [17] yang memiliki skor *precision* 60,2% dan *recall*

52,1% pada data latih dan skor *precision* 59,5% dan *recall* 45,9% pada data uji. Pada Gambar 5.1, dapat dilihat pada data *gt* (Gambar 5.1a) terdapat 2 objek sedangkan pada data *det* (Gambar 5.1b) terdapat 3 objek dan *bounding box* tidak menutupi objek secara menyeluruh. Untuk data uji 2DMOT15 jenis *bounding box* yang ada adalah data *det* saja.

5.3. Skenario Uji Coba

Subbab ini akan menjelaskan desain pengujian program untuk mendapatkan parameter yang menghasilkan nilai uji paling optimal. Hasil terbaik dari suatu skenario uji coba akan digunakan untuk skenario uji coba berikutnya. Skenario uji yang dijalankan secara lokal diterapkan pada data latih 2DMOT15 dengan menjalankan program pada data *det* dan dievaluasi dengan data *gt*. Khusus untuk skenario uji luar dengan data uji akan dievaluasi melalui pengumpulan pada *website* MOTChallenge. Ada empat macam skenario uji lokal yang akan dijalankan yang terbagi menjadi dua jenis, yaitu skenario yang menjalankan sebagian komponen dari program dan skenario yang menjalankan program *object tracking* secara keseluruhan. Tambahan satu skenario uji luar menggunakan data uji yang dievaluasi melalui *website*. Skenario uji yang akan dilakukan adalah sebagai berikut:

1. Uji coba performa asosiasi data berdasarkan fungsi *cost* dan nilai $default_cost_d0$
2. Uji coba performa prediksi *Kalman filter* dengan modifikasi matriks
3. Uji coba metode inisialisasi objek baru
4. Uji coba batas minimal deteksi beruntun
5. Uji coba luar pada data uji

Parameter dasar yang digunakan terdapat pada Tabel 5.4 dan dapat berubah di setiap uji coba yang dilakukan. Pada setiap uji co-

ba akan ditetapkan nilai parameter yang memberikan hasil terbaik. Pada *toolkit* yang digunakan untuk menghitung metrik MOT, nilai batas minimal IoU yang digunakan untuk menentukan pasangan *ground truth* dengan hipotesis adalah 0,5.

Tabel 5.4 Parameter Dasar pada Sistem *Object Tracking*

Keterangan	Parameter
n_bins histogram	64
SCEA $num_cluster_member$	5
SCEA $gating_threshold$	0,7
Tracker max_age	2

5.3.1. Uji Coba Performa Asosiasi Data

Uji coba performa asosiasi data dilakukan untuk menguji ketepatan asosiasi data yang hanya dijalankan dalam proses SCEA dan SCOR yang menggunakan data *gt* pada data latih. Masukan $\mathcal{S}_w$ untuk SCEA akan diambil dari data deteksi pada *frame* $t-1$ dan $\mathcal{D}$ diambil dari data deteksi pada *frame* t. Kemudian $\mathcal{S}_m$ dan $\tilde{\mathcal{D}}$ diambil dari objek dan deteksi yang belum diasosiasikan di SCEA untuk menjadi masukan proses SCOR. Sebelum mengolah data *frame* berikutnya, kecepatan objek diperbarui berdasarkan deteksi dengan ID yang sama pada *frame* saat ini untuk mempertahankan nilai batasan struktural.

Evaluasi dilakukan dengan membandingkan ketepatan hasil asosiasi berdasarkan ID. Pasangan asosiasi objek (*frame* $t-1$) dengan deteksi (*frame* t) yang menjadi *ground truth* terdiri dari pasangan dengan ID yang sama, pasangan dengan ID objek kosong karena merupakan objek yang baru muncul, dan pasangan dengan ID deteksi kosong karena merupakan objek yang hilang pada *frame* tersebut.

Pengelompokkan dalam evaluasi hasil asosiasi divisualisa-

sikan pada Tabel 5.5. Hasil prediksi asosiasi yang sesuai dengan *ground truth* akan dihitung sebagai *true positive* (TP) seperti baris pertama Tabel 5.5b dengan baris pertama Tabel 5.5a. Prediksi asosiasi yang tidak ada di *ground truth* dihitung sebagai *false positive* (FP) seperti beris kedua Tabel 5.5b. Asosiasi di *ground truth* yang gagal diprediksi dihitung sebagai *false negative* (FN) seperti baris kedua dan keenam Tabel 5.5a. Nilai *precision* dan *recall* dihitung untuk tiap video yang ada dalam data latih.

Tabel 5.5 Perbandingan Pasangan ID Hasil Asosiasi: (a) *Ground Truth* dan (b) Prediksi

(a)

No	ID $t-1$	ID t
1	1	1
2	2	-
3	3	3
4	4	4
5	5	5
6	-	6

(b)

No	ID $t-1$	ID t
1	1	1
2	2	6
3	3	3
4	4	4
5	5	5

Parameter SCEA dan SCOR seperti pada Tabel 5.4 dengan parameter asosiasi data yang divariasikan antara lain:

1. Fungsi *cost* F_c dan F_r
2. Nilai $default_cost_d0$

Fungsi *cost* F_r dan F_c yang digunakan ada dua macam, yaitu versi orisinal sesuai Persamaan 2.16 dan 2.21 dengan yang sudah dimodifikasi pada Subbab 3.2.10 dan 3.2.15.

Untuk nilai $default_cost_d0$ akan dicoba beberapa nilai yang besarannya disesuaikan dengan versi fungsi *cost* yang digunakan. Nilai ini ditentukan dengan mengikuti besar nilai *cost*

yang dihasilkan untuk nilai IoU tertentu, sehingga dapat menjadi nilai ambang batas IoU untuk menjadi sebuah asosiasi yang valid. Nilai IoU yang dijadikan ambang batas adalah 0,0, 0,1, 0,2, dan 0,3. Pada fungsi *cost* orisinal nilai IoU tersebut dipetakan ke $default_cost_d0$ dengan menghitung $-log(IoU)$. Pada fungsi *cost* modifikasi nilai IoU dipetakan dengan menghitung $1 - IoU$. Pemetaan terdapat pada Tabel 5.6 dengan nilai pada fungsi orisinal mengalami pembulatan. Pengecualian pada nilai IoU 0,0 di fungsi orisinal menggunakan nilai log dari 0,03 karena log dari 0,0 tidak terdefinisi.

Tabel 5.6 Pemetaan Nilai IoU ke $default_case_d0$ pada Fungsi *Cost* Orisinal dan Modifikasi

IoU	Orisinal	Modifikasi
0,0	3,5	1,0
0,1	2,3	0,9
0,2	1,6	0,8
0,3	1,2	0,7

Hasil uji coba terdapat pada Tabel 5.7. Nilai *precision* dan *recall* yang ditampilkan merupakan hasil rata-rata dari semua video dengan bobot berupa total asosiasi yang ada pada video tersebut. Hasil uji dengan detail tiap video dapat dilihat pada Lampiran A.

5.3.2. Uji Coba Prediksi Kalman Filter

Uji coba prediksi *Kalman filter* dilakukan untuk menguji performa *Kalman filter* yang diimplementasikan dalam melakukan prediksi *bounding box*. Uji coba ini dijalankan pada data latih *gt*. Ketepatan prediksi *Kalman filter* diperlukan agar dapat menghasilkan irisan *bounding box* yang lebih luas saat proses asosiasi data.

Pengujian dilakukan dengan menjalankan OBJECTSTATETRACKER pada tiap objek yang ada di semua

Tabel 5.7 Nilai *Weighted Precision* dan *Recall* pada Uji Asosiasi Data

Fungsi Cost	IoU	Weighted Precision ↑	Weighted Recall ↑
Orisinal	0,3	99,373	99,503
Orisinal	0,2	99,414	99,505
Orisinal	0,1	99,440	99,493
Orisinal	0,0	99,435	99,446
Modifikasi	0,3	99,348	99,522
Modifikasi	0,2	99,453	99,574
Modifikasi	0,1	99,511	99,602
Modifikasi	0,0	**99,581**	**99,624**

video. Proses *tracking* dijalankan dari *frame* awal munculnya objek hingga *frame* terakhir objek muncul. Ketepatan prediksi diukur dari skor *Root Mean Square Error* (RMSE) hasil prediksi *bounding box* pada *frame* t berdasarkan proses *update* yang dijalankan pada *frame* $t-1$ dibandingkan dengan *bounding box ground truth* pada *frame* t.

Matriks F, R, dan H yang digunakan dalam *Kalman filter* sesuai dengan desain pada Subbab 3.2.17. Variasi diterapkan pada matriks Q yang menggunakan versi orisinal dari Persamaan 2.24 dan hasil modifikasi pada Subbab 3.2.17.

Tabel 5.8 Nilai Rata-Rata RMSE per Komponen Pengukuran pada *Kalman Filter* untuk Objek

Matriks	$RMSE_x \downarrow$	$RMSE_y \downarrow$	$RMSE_w \downarrow$	$RMSE_h \downarrow$
Orisinal	40,900	20,437	5,931	5,002
Modifikasi	**4,209**	**2,396**	5,931	5,002

Hasil uji coba terdapat pada Tabel 5.8. Nilai RMSE yang ditampilkan merupakan rata-rata nilai RMSE dari semua video pada data latih *gt*. $RMSE_x$ untuk komponen posisi di sumbu-x. $RMSE_y$ untuk komponen posisi di sumbu-y. $RMSE_w$ untuk komponen dimensi lebar. $RMSE_h$ untuk komponen dimensi tinggi. Hasil uji detail tiap video dapat dilihat pada Lampiran B.

5.3.3. Uji Coba Metode Inisialisasi Objek Baru

Uji coba metode inisialisasi objek baru dilakukan untuk menguji pengaruh metode inisialisasi objek baru terhadap nilai metrik *multi object tracking*. Uji coba dijalankan pada data latih *det* yang dievaluasi dengan data latih *gt*.

Pengujian dilakukan dengan menjalankan program *object tracking* pada data latih *det*. Hasil keluaran dalam format data anotasi 2DMOT15 dievaluasi dengan data latih *gt* menggunakan *toolkit* yang disediakan dari dataset. Metrik yang digunakan adalah *precision*, *recall*, *Multiple Object Tracking Accuracy* (MOTA), dan *Multiple Object Tracking Precision* (MOTP).

Parameter yang digunakan dalam program *object tracking* sesuai dengan Tabel 5.4. Parameter lain mengikuti parameter optimal sesuai hasil uji coba yang sudah dilakukan. Fungsi *cost* yang digunakan adalah versi modifikasi dengan nilai $default_cost_d0$ sebesar 1,0. Untuk *Kalman filter* menggunakan matriks Q yang sudah dimodifikasi. Metode inisialisasi objek yang diujikan ada dua seperti pada Subbab 3.2.22, yaitu metode I dan metode II yang menggunakan *sliding window*.

Hasil uji coba metode inisialisasi objek baru terdapat pada Tabel 5.9. Nilai *precision*, *recall*, MOTA dan MOTP yang ditampilkan merupakan rata-rata dari semua data video. Hasil uji coba detail tiap video terdapat pada Lampiran C.

Tabel 5.9 Nilai Hasil Uji Metode Inisialisasi Objek Baru

Inisialisasi Objek	Precision ↑	Recall ↑	MOTA ↑	MOTP ↑
Metode I	57,7%	**54,0%**	6,0%	72,1%
Metode II	**65,9%**	47,8%	**17,9%**	**72,3%**

5.3.4. Uji Coba Batas Minimal Deteksi Beruntun

Uji coba batas minimal deteksi beruntun dilakukan untuk menguji pengaruh nilai batas minimal deteksi beruntun terhadap nilai metrik *multi object tracking*. Uji coba dijalankan pada data latih *det* yang dievaluasi dengan data latih *gt*.

Pengujian dilakukan dengan menjalankan program *object tracking* pada data latih *det*. Hasil keluaran dalam format data anotasi 2DMOT15 dievaluasi dengan data latih *gt* menggunakan *toolkit* yang disediakan dari dataset. Metrik yang digunakan adalah *precision*, *recall*, *Multiple Object Tracking Accuracy* (MOTA), dan *Multiple Object Tracking Precision* (MOTP).

Parameter yang digunakan dalam program *object tracking* sesuai dengan Tabel 5.4. Parameter lain mengikuti parameter optimal sesuai hasil uji coba yang sudah dilakukan. Fungsi *cost* yang digunakan adalah versi modifikasi dengan nilai $default_cost_d0$ sebesar 1,0. Untuk *Kalman filter* menggunakan matriks Q yang sudah dimodifikasi. Inisialisasi objek baru menggunakan metode II. Paramater deteksi beruntun min_streak yang diujikan bernilai 2, 3, dan 4.

Hasil uji coba nilai batas minimal deteksi beruntun terdapat pada Tabel 5.10. Nilai *precision*, *recall*, MOTA dan MOTP yang ditampilkan merupakan rata-rata dari semua data video. Hasil uji yang detail untuk tiap video terdapat pada Lampiran D.

Tabel 5.10 Nilai Hasil Uji Coba Batas Minimal Deteksi Beruntun

Min Streak	Precision ↑	Recall ↑	MOTA ↑	MOTP ↑
2	70,4%	**42,2%**	20,4%	72,6%
3	73,1%	37,9%	**20,7%**	72,7%
4	**75,0%**	34,4%	20,3%	**72,8%**

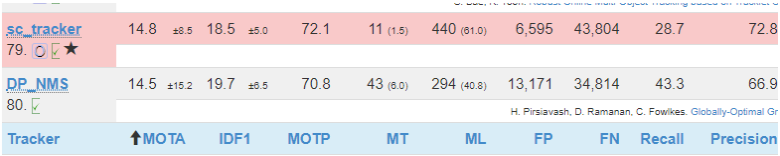
5.3.5. Uji Coba Luar pada Data Uji

Uji coba luar menggunakan data uji dilakukan untuk menguji performa program *object tracking* yang dibuat pada data uji yang tidak diketahui *ground truth*-nya. Program *object tracking* dijalankan pada data uji dan keluarannya dikirim ke *website* MOTChallenge untuk dievaluasi.

Parameter yang digunakan dalam *object tracking* sesuai dengan Tabel 5.4 dengan parameter lain mengikuti parameter optimal dari uji coba yang telah dilakukan sebelumnya. Hasil evaluasi terdapat pada Tabel 5.11. Hasil uji yang lebih lengkap terdapat pada Lampiran E. Hasil pengumpulan pada *website* mendapatkan peringkat ke-79 dari 81 seperti terdapat pada Gambar 5.2.

Tabel 5.11 Nilai Hasil Uji Coba Luar pada Data Uji

Metrik	Nilai
FP	6595
FN	43804
Switch	1962
MOTA	14,78%
MOTP	72,14%
Recall	28,70%
Precision	72,78%



Tracker	↑MOTA	IDF1	MOTP	MT	ML	FP	FN	Recall	Precision
sc_tracker 79.	14.8 ±8.5	18.5 ±5.0	72.1	11 (1.5)	440 (61.0)	6,595	43,804	28.7	72.8
DP_NMS 80.	14.5 ±15.2	19.7 ±6.5	70.8	43 (6.0)	294 (40.8)	13,171	34,814	43.3	66.9

H. Pirsiavash, D. Ramanan, C. Fowlkes. Globally-Optimal Gr

Gambar 5.2 Peringkat Hasil Program *Object Tracking* pada *Website* MOTChallenge

5.4. Hasil dan Analisis

Pada uji coba performa asosiasi data, dapat diketahui bahwa penggunaan fungsi *cost* F_r dan F_c yang sudah dimodifikasi dengan nilai $default_cost_d0$ sebesar 1,0 dapat menghasilkan *precision* 99,581% dan *recall* 99,624% yang lebih besar dari kombinasi parameter lainnya. Hal ini dapat terjadi karena asosiasi objek dan deteksi yang nilai IoU-nya mendekati nol tidak langsung dikategorikan sebagai d_0 karena masih dapat terbantu dengan hasil *cost* pada objek *anchor* yang lain yang kemudian diagregasi.

Pada uji coba prediksi *Kalman filter*, diperoleh skor RMSE terbaik secara keseluruhan dari penggunaan matriks Q yang sudah dimodifikasi dengan nilai $RMSE_x$ 4,209 dan $RMSE_y$ 2,396 yang lebih kecil daripada nilai RMSE yang dihasilkan dari matriks Q orisinal. Hal ini merupakan hasil dari perhitungan model pergerakan dan kovarians pada Subbab 3.2.17 yang menyesuaikan matriks kovarians Q dengan model pergerakan objek yang dibuat. Modifikasi yang dilakukan hanya pada matriks Q tanpa mengubah bagian lain dari [3] dapat mengurangi kesalahan prediksi tanpa memberikan kompleksitas tambahan karena perhitungannya tetap konstan atau $O(1)$.

Pada uji coba inisialisasi objek baru, diperoleh skor *precision*, MOTA, dan MOTP pada metode II lebih tinggi dengan nilai masing-masing sebesar 65,9%, 17,9%, dan 72,3%. Skor MOTA meningkat karena pada metode II sebuah deteksi baru akan diinisialisasi seba-

Tabel 5.12 Parameter Optimal untuk Sistem

Keterangan	Parameter Optimal
Fungsi *cost*	modifikasi
Nilai $default_cost_d0$	1,0
Matriks *Q Kalman filter*	modifikasi
Metode inisialisasi objek baru	metode II (dengan *sliding window*)
Nilai *min streak*	2

gai objek terlacak jika berhasil dipasangkan dua kali berturut-turut, sehingga dapat mengurangi kesalahan *tracking* akibat adanya deteksi *false positive* pada data latih *det*. Batas IoU sebesar 0,5 yang digunakan untuk menentukan pasangan dalam *sliding window* juga cukup selektif dalam memasangkan deteksi antar dua *frame*.

Pada uji coba batas minimal deteksi beruntun, tiga variasi parameter *min streak* yang digunakan menghasilkan keunggulan skor yang berbeda-beda. Pada *min streak* sebesar 4, skor *recall* yang didapat jauh lebih kecil daripada yang lain. Hal ini terjadi dikarenakan semakin tinggi nilai *min streak*, maka sistem *object tracking* akan semakin selektif dalam mengembalikan objek yang terlacak sehingga semakin banyak objek yang dianggap tidak terlacak dan menurunkan *recall*. Pada perbandingan *min streak* sebesar 2 dan 3, peningkatan *precision* pada *min streak* sebesar 3 lebih kecil dibandingkan peningkatan *recall* pada *min streak* sebesar 2, skor MOTA dan MOTP juga tidak berbeda jauh. Sehingga parameter *min streak* yang optimal adalah sebesar 2 dengan *precision* 70,4%, *recall* 42,2%, MOTA 20,4%, dan MOTP 72,6%.

Dari keempat uji coba untuk mendapatkan parameter optimal yang sudah dilakukan, ditetapkan parameter optimal pada sistem *object tracking* yang terdapat pada Tabel 5.12.

Berdasarkan nilai parameter SCEA $num_cluster_member$

Tabel 5.13 Perbandingan Rata-Rata Banyak Pasangan Asosiasi per *Frame* di Data Latih *gt*

Video	Kepadatan Objek	Tanpa Mask F_s	Dengan Mask F_s
ADL-Rundle-6	9,5	92,29	56,67
ADL-Rundle-8	10,4	109,39	67,56
ETH-Bahnhof	5,4	26,33	17,15
ETH-Pedcross2	7,5	44,90	21,11
ETH-Sunnyday	5,2	23,25	16,66
KITTI-13	2,2	6,18	4,06
KITTI-17	4,7	20,92	11,90
PETS09-S2L1	5,6	29,10	22,64
TUD-Campus	5,1	21,59	15,33
TUD-Stadtmitte	6,5	29,08	18,37
Venice-2	11,9	141,72	70,43

dan $gating_threshold$ pada Tabel 5.4, dianalisis pengaruhnya terhadap reduksi ruang pencarian yang harus diproses oleh sistem. Pertama dari nilai $gating_threshold$ yang digunakan dalam proses *gating*. Dalam proses *gating* ada dua *mask* yang digunakan untuk seleksi yaitu berdasarkan jarak titik pusat dibanding diagonal dan berdasarkan batas nilai *cost* F_s terhadap $gating_threshold$. Perbandingan rata-rata banyaknya pasangan asosiasi yang didapat terdapat pada Tabel 5.13. Dari tabel tersebut dapat dilihat bahwa seleksi berdasarkan batas nilai *cost* F_s terhadap $gating_threshold$ sebesar 0,7 membantu mengurangi pasangan asosiasi objek dengan deteksi yang harus dievaluasi. Nilai $gating_threshold$ berpengaruh terhadap banyaknya pasangan asosiasi yang berhasil diseleksi, jika terlalu banyak yang diseleksi maka pasangan yang seharusnya benar dapat terseleksi dan tidak dievaluasi.

Dianalisis juga pengaruh proses partisi menggunakan pa-

Tabel 5.14 Perbandingan Rata-Rata Banyak Kombinasi Asosiasi yang Dievaluasi pada Data Latih gt

Video	Kepadatan Objek	Tanpa Partisi	Dengan Partisi
ADL-Rundle-6	9,5	547051,72	939,22
ADL-Rundle-8	10,4	247233,63	221,48
ETH-Bahnhof	5,4	1160,57	52,72
ETH-Pedcross2	7,5	5932,70	140,07
ETH-Sunnyday	5,2	145,06	65,10
KITTI-13	2,2	6,02	6,02
KITTI-17	4,7	168,17	40,20
PETS09-S2L1	5,6	71,76	31,31
TUD-Campus	5,1	88,44	73,00
TUD-Stadtmitte	6,5	73,26	52,81
Venice-2	11,9	469931,56	302,97

rameter $num_cluster_member$ sebesar lima terhadap kombinasi asosiasi yang harus dievaluasi yang terdapat pada Tabel 5.14. Pada tabel tersebut dapat dilihat bahwa proses partisi objek menjadi *cluster* dengan maksimal anggota sebanyak lima dapat mengurangi banyaknya kombinasi asosiasi yang harus dievaluasi secara signifikan terutama pada video dengan tingkat kepadatan objek yang tinggi. Nilai $num_cluster_member$ yang semakin besar dapat menghasilkan partisi yang lebih sedikit dengan banyak kombinasi asosiasi yang meningkat.

Pada uji coba luar dengan menggunakan data uji, diperoleh nilai metrik yang lebih rendah dari hasil uji pada data latih *det* dan peringkat yang tidak bagus. Menurunnya performa ini juga sejalan dengan menurunnya performa antara uji asosiasi data yang menggunakan data latih *gt* dengan uji parameter keseluruhan sistem yang menggunakan data latih *det*. oleh karena itu, dilakukan analisis le-

Tabel 5.15 Perbandingan Banyak Data *Bounding Box* pada Data Latih *gt* dan *det*

Video	Banyak data gt	Banyak data det
ADL-Rundle-6	5009	4902
ADL-Rundle-8	6783	7114
ETH-Bahnhof	5415	6817
ETH-Pedcross2	6263	1438
ETH-Sunnyday	1858	1123
KITTI-13	762	1040
KITTI-17	683	552
PETS09-S2L1	4476	5578
TUD-Campus	359	322
TUD-Stadtmitte	1156	1129
Venice-2	7141	7313

bih lanjut mengenai penyebab menurunnya performa program *object tracking*.

Dari segi kualitas *bounding box* yang ada pada data *det*, terdapat perbedaan yang dapat berdampak negatif. Pertama, terdapat perbedaan banyaknya *bounding box* antara data latih *gt* dengan *det* seperti pada Tabel 5.15. Enam dari dua belas video memiliki data *gt* yang lebih banyak daripada data *det*, dengan selisih terbesar pada data video ETH-Pedcross2. Data *gt* yang lebih banyak daripada data *det* akan meningkatkan nilai FN pada hasil evaluasi karena semakin banyak objek pada *ground truth* yang tidak dapat dipasangkan sesuai cara kerja evaluasi MOT pada Subbab 2.7. Sedangkan pada data video lain yang data *det* lebih banyak daripada data *gt* dapat meningkatkan nilai FP karena semakin banyak hipotesis yang tidak mendapat pasangan objek di *ground truth*.

Seperti yang dapat dilihat pada Gambar 5.3, pada data *det* terdapat beberapa *frame* yang tidak memiliki data *bounding box*. Pada

data *gt*, terdapat baris data untuk *frame* 13 hingga 15,sedangkan pada data *det* tidak ada. Kosongnya data antara *frame* seperti ini dapat menyebabkan FN maupun *switch* karena objek di *ground truth* tidak mendapat pasangan hipotesis atau muncul hipotesis dengan ID yang baru akibat melebihi nilai parameter max_age.

Untuk mendapatkan analisis pengaruh algoritma yang diimplementasikan pada program *object tracking* terhadap hasil evaluasi, pemeriksaan hasil dilakukan pada keluaran program yang dijalankan pada data latih *gt*. Pada uji coba metode inisialisasi objek, dilihat dari detail pengujian pada Lampiran C, terdapat peningkatan FN yang menunjukkan semakin banyak objek *ground truth* yang gagal dilacak.

Dapat dilihat pada Gambar 5.4, ilustrasi *sliding window* dari video KITTI-13 pada data latih *gt*. Kotak berwarna hijau menandakan *bounding box* dari deteksi pada *frame 71* sedangkan warna kuning untuk deteksi pada *frame* ke 72. Pada *frame* 71, deteksi yang tidak terasosiasi terdiri dari ID 8, 10, 11, dan 12. Sedangkan, pada *frame* 72, deteksi yang tidak terasosiasi terdiri dari ID 8, 9, 10, 11, dan 12. Dari hasil perhitungan IoU antar pasangan deteksi dari *frame* 71 dan 72, yang nilainya bukan nol terdapat pada Tabel 5.16. Diketahui bahwa semua deteksi dengan ID yang sama terhitung beririsan karena nilai IoU-nya lebih dari nol, tetapi dikarenakan batas IoU yang digunakan sebesar 0,5 maka deteksi-deteksi tersebut tidak diinisialisasi sebagai objek yang terlacak dan menyebabkan nilai FN meningkat.

Tabel 5.16 Daftar Nilai IoU dari Pasangan Deteksi pada *Frame* 71 dan 72

ID frame 71	ID frame 72	IoU
8	8	0,430
10	10	0,265
11	11	0,405
12	12	0,382

```
11,1,794,164,19.667,54.959,0,-1,-1,-1
12,0,773,160,23.128,59.487,1,-1,-1,-1
12,1,801,163,20.302,57.116,0,-1,-1,-1
13,0,779,160,25.333,61.888,1,-1,-1,-1
13,1,808,163,20.0,59.449,1,-1,-1,-1
13,2,201,165,18.0,57.389,0,-1,-1,-1
14,0,786,161,27.325,64.492,1,-1,-1,-1
14,1,817,163,21.524,61.983,1,-1,-1,-1
14,2,187,167,20.4,58.974,0,-1,-1,-1
15,0,793,162,28.0,67.324,1,-1,-1,-1
15,1,826,165,22.0,64.744,1,-1,-1,-1
15,2,172,168,22.333,60.648,1,-1,-1,-1
16,0,801,164,31.673,70.441,1,-1,-1,-1
16,1,836,166,25.478,67.762,1,-1,-1,-1
16,2,156,170,23.22,62.42,1,-1,-1,-1
17,0,809,165,33.667,73.861,1,-1,-1,-1
```

(a)

```
9,-1,17,99,39.966,90.691,13.437,-1,-1,-1
12,-1,710,148,30.821,69.941,12.326,-1,-1,-1
16,-1,801,171,28.33,64.287,76.165,-1,-1,-1
16,-1,831,171,28.33,64.287,51.771,-1,-1,-1
17,-1,844,175,28.33,64.287,47.844,-1,-1,-1
```

(b)

Gambar 5.3 Perbandingan Data pada Video KITTI-13 *Frame* ke-12 hingga 16: (a) data *gt* dan (b) data *det*

Gambar 5.4 Ilustrasi *Sliding Window* Deteksi yang Tidak Terasosiasi pada Video KITTI-13 *gt Frame* 71 dan 72

[*Halaman ini sengaja dikosongkan*]

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan serta saran-saran tentang pengembangan yang dapat dilakukan terhadap tugas akhir ini di masa yang akan datang.

6.1. Kesimpulan

Berdasarkan penjabaran di bab-bab sebelumnya, dapat disimpulkan beberapa poin terkait implementasi sistem.

1. Konsep batasan struktural untuk mendapatkan posisi relatif objek dapat diimplementasikan dalam asosiasi data di sistem *object tracking*
2. Penyesuaian matriks dalam *Kalman filter* dengan pemodelan pergerakan objek dapat menghasilkan prediksi yang lebih akurat.
3. Masalah hasil deteksi *false positive* dapat diatasi dengan menerapkan skema seleksi dalam menginisialisasi deteksi menjadi objek memanfaatkan *sliding window* dan *linear assignment*.
4. Penggunaan nilai batas minimal IoU yang cukup besar pada perhitungan dalam *sliding window* dapat meningkatkan *false negative*.
5. Kualitas hasil dari detektor yang kurang baik berdampak buruk terhadap performa program *object tracking*.
6. Sistem *object tracking* yang memanfaatkan batasan struktural telah berhasil diimplementasikan dengan nilai MOTA terbaik sebesar 20,4% yang didapatkan dari penggunaan fungsi *cost* yang dimodifikasi, matriks *Q Kalman filter* yang disesuaikan, sistem *sliding window* untuk inisialisasi objek dan seleksi

berdasarkan terlacak secara beruntun dua *frame*.

6.2. Saran

Pada tugas akhir kali ini tentunya terdapat kekurangan serta nilai-nilai yang dapat penulis ambil. Berikut adalah saran-saran yang dapat digunakan untuk pengembangan di masa yang akan datang. Saran-saran ini didasarkan pada hasil desain, implementasi, dan uji coba yang telah dilakukan.

1. Sumber data *object detection* dari detektor yang lebih baik karena kualitas deteksi memengaruhi performa *object tracker*.
2. Desain fungsi *cost* lain untuk komponen dimensi dan penampilan objek yang bisa membantu menentukan asosiasi yang tepat antara objek dan deteksi.
3. Pemetaan nilai konstanta dan pengaruh positif-negatif terhadap performa sistem untuk kemudian dioptimasi dengan kompromi sebaik mungkin.
4. Implementasi sistem *sliding window* dengan ukuran yang lebih besar sehingga dapat melakukan seleksi deteksi *false positive* dengan lebih handal.
5. Pemodelan nilai *state* dalam *Kalman filter* untuk menggunakan komponen pengukuran yang lain seperti proporsi dimensi objek.

DAFTAR PUSTAKA

[1] B. Yang dan R. Nevatia, "Multi-Target Tracking by Online Learning a CRF Model of Appearance and Motion Patterns", *International Journal of Computer Vision*, vol 107, halaman 203–217, 2014. doi: 10.1007/s11263-013-0666-4.

[2] W. Choi, "Near-Online Multi-target Tracking with Aggregated Local Flow Descriptor", *CoRR*, vol abs/1504.02340, 2015. arXiv: 1504.02340.

[3] J. H. Yoon, C.-R. Lee, M.-H. Yang, dan K.-J. Yoon, "Structural Constraint Data Association for Online Multi-object Tracking", *International Journal of Computer Vision*, vol 127, halaman 1–21, 2019. doi: 10.1007/s11263-018-1087-1.

[4] L. Leal-Taixé, A. Milan, I. D. Reid, S. Roth, dan K. Schindler, "MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking", *CoRR*, vol abs/1504.01942, 2015. arXiv: 1504.01942.

[5] R. Verschae dan J. Ruiz-del Solar, "Object Detection: Current and Future Directions", *Frontiers in Robotics and AI*, vol 2, halaman 29, 2015, issn: 2296-9144. doi: 10.3389/frobt.2015.00029.

[6] R. Szeliski, *Computer Vision*. Springer London, 2011. doi: 10.1007/978-1-84882-935-0.

[7] R. Collins, *Introduction to Data Association*, 2012. [Online]. Available: http://www.cse.psu.edu/~rtc12/CSE598C/.

[8] D. A. Forsyth dan J. Ponce, *Computer Vision - A Modern Approach, Second Edition.* Prentice Hall Professional Technical Reference, 2012, isbn: 0130851981.

[9] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, X. Zhao, dan T.-K. Kim, *Multiple Object Tracking: A Literature Review*, 2014. arXiv: 1409.7618 [cs.CV].

[10] D. Arthur dan S. Vassilvitskii, "K-Means++: The Advantages of Careful Seeding", dalam *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, USA: Society for Industrial dan Applied Mathematics, 2007, halaman 1027–1035, isbn: 9780898716245.

[11] K. Bennett, P. Bradley, dan A. Demiriz, "Constrained K-Means Clustering", Tech. Rep. MSR-TR-2000-65, May 2000, halaman 8.

[12] G. Welch dan G. Bishop, "An Introduction to the Kalman Filter", USA, Tech. Rep., 1995.

[13] K. Bernardin dan R. Stiefelhagen, "Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics", *J. Image Video Process.*, vol 2008, Jan. 2008, issn: 1687-5176. doi: 10.1155/2008/246309.

[14] R. Sadli, *Object Tracking: 2-D Object Tracking using Kalman Filter in Python*, 2020. [Online]. Available: https://machinelearningspace.com/2d-object-tracking-using-kalman-filter/.

[15] C. Z. (https://stackoverflow.com/users/2487184/ct zhu), *Using numpy to build an array of all combinations of two arrays*, Stack Overflow. [Online]. Available: https://stackoverflow.com/revisions/35608701/1.

[16] D. (https://stackoverflow.com/users/3293881/divakar), *Removing rows with duplicates in a NumPy array*, Stack Overflow. [Online]. Available: https://stackoverflow.com/revisions/45136720/2.

[17] P. Dollar, R. Appel, S. Belongie, dan P. Perona, “Fast Feature Pyramids for Object Detection”, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol 36, no. 8, halaman 1532–1545, Aug. 2014. doi: 10.1109/TPAMI.2014.2300479.

[*Halaman ini sengaja dikosongkan*]

LAMPIRAN A: Detail Hasil Uji Coba Performa Asosiasi Data

Tabel A.1 Hasil Uji Asosiasi dengan Fungsi *Cost* Orisinal dan $default_cost_d0$ 1,2

Video	TP	FN	FP	Precision	Recall
ADL-Rundle-6	5020	0	0	100,00	100,00
ADL-Rundle-8	6796	0	0	100,00	100,00
ETH-Bahnhof	5660	38	32	99,44	99,33
ETH-Pedcross2	6368	18	20	99,69	99,72
ETH-Sunnyday	1882	4	6	99,68	99,79
KITTI-13	682	117	182	78,94	85,36
KITTI-17	674	14	15	97,82	97,97
PETS09-S2L1	4483	4	4	99,91	99,91
TUD-Campus	355	2	2	99,44	99,44
TUD-Stadtmitte	1153	0	0	100,00	100,00
Venice-2	7142	4	6	99,92	99,94

Tabel A.2 Hasil Uji Asosiasi dengan Fungsi *Cost* Orisinal dan $default_cost_d0$ 1,6

Video	TP	FN	FP	Precision	Recall
ADL-Rundle-6	5020	0	0	100,00	100,00
ADL-Rundle-8	6796	0	0	100,00	100,00
ETH-Bahnhof	5655	43	36	99,37	99,25
ETH-Pedcross2	6364	22	23	99,64	99,66
ETH-Sunnyday	1877	9	11	99,42	99,52
KITTI-13	694	105	155	81,74	86,86
KITTI-17	675	13	13	98,11	98,11
PETS09-S2L1	4483	4	4	99,91	99,91
TUD-Campus	353	4	4	98,88	98,88
TUD-Stadtmitte	1153	0	0	100,00	100,00
Venice-2	7146	0	0	100,00	100,00

Tabel A.3 Hasil Uji Asosiasi dengan Fungsi *Cost* Orisinal dan $default_cost_d0$ 2,3

Video	TP	FN	FP	Precision	Recall
ADL-Rundle-6	5020	0	0	100,00	100,00
ADL-Rundle-8	6796	0	0	100,00	100,00
ETH-Bahnhof	5636	62	49	99,14	98,91
ETH-Pedcross2	6364	22	23	99,64	99,66
ETH-Sunnyday	1877	9	11	99,42	99,52
KITTI-13	707	92	129	84,57	88,49
KITTI-17	676	12	12	98,26	98,26
PETS09-S2L1	4483	4	4	99,91	99,91
TUD-Campus	353	4	4	98,88	98,88
TUD-Stadtmitte	1153	0	0	100,00	100,00
Venice-2	7146	0	0	100,00	100,00

Tabel A.4 Hasil Uji Asosiasi dengan Fungsi *Cost* Orisinal dan $default_cost_d0$ 3,5

Video	TP	FN	FP	Precision	Recall
ADL-Rundle-6	5020	0	0	100,00	100,00
ADL-Rundle-8	6796	0	0	100,00	100,00
ETH-Bahnhof	5610	88	72	98,73	98,46
ETH-Pedcross2	6347	39	41	99,36	99,39
ETH-Sunnyday	1868	18	20	98,94	99,05
KITTI-13	740	59	77	90,58	92,62
KITTI-17	676	12	12	98,26	98,26
PETS09-S2L1	4483	4	4	99,91	99,91
TUD-Campus	353	4	4	98,88	98,88
TUD-Stadtmitte	1153	0	0	100,00	100,00
Venice-2	7146	0	0	100,00	100,00

Tabel A.5 Hasil Uji Asosiasi dengan Fungsi *Cost* Modifikasi dan $default_cost_d0$ 1,0

Video	TP	FN	FP	Precision	Recall
ADL-Rundle-6	5020	0	0	100,00	100,00
ADL-Rundle-8	6796	0	0	100,00	100,00
ETH-Bahnhof	5652	46	35	99,39	99,19
ETH-Pedcross2	6370	16	18	99,72	99,75
ETH-Sunnyday	1882	4	6	99,68	99,79
KITTI-13	731	68	93	88,71	91,49
KITTI-17	678	10	10	98,55	98,55
PETS09-S2L1	4487	0	0	100,00	100,00
TUD-Campus	353	4	4	98,88	98,88
TUD-Stadtmitte	1153	0	0	100,00	100,00
Venice-2	7142	4	6	99,92	99,94

Tabel A.6 Hasil Uji Asosiasi dengan Fungsi *Cost* Modifikasi dan $default_cost_d0$ 0,9

Video	TP	FN	FP	Precision	Recall
ADL-Rundle-6	5020	0	0	100,00	100,00
ADL-Rundle-8	6796	0	0	100,00	100,00
ETH-Bahnhof	5663	35	29	99,49	99,39
ETH-Pedcross2	6370	16	18	99,72	99,75
ETH-Sunnyday	1882	4	6	99,68	99,79
KITTI-13	710	89	131	84,42	88,86
KITTI-17	677	11	12	98,26	98,40
PETS09-S2L1	4487	0	0	100,00	100,00
TUD-Campus	355	2	2	99,44	99,44
TUD-Stadtmitte	1153	0	0	100,00	100,00
Venice-2	7142	4	6	99,92	99,94

Tabel A.7 Hasil Uji Asosiasi dengan Fungsi *Cost* Modifikasi dan $default_cost_d0$ 0,8

Video	TP	FN	FP	Precision	Recall
ADL-Rundle-6	5020	0	0	100,00	100,00
ADL-Rundle-8	6796	0	0	100,00	100,00
ETH-Bahnhof	5667	31	27	99,53	99,46
ETH-Pedcross2	6370	16	20	99,69	99,75
ETH-Sunnyday	1882	4	6	99,68	99,79
KITTI-13	695	104	158	81,48	86,98
KITTI-17	677	11	12	98,26	98,40
PETS09-S2L1	4487	0	0	100,00	100,00
TUD-Campus	355	2	2	99,44	99,44
TUD-Stadtmitte	1153	0	0	100,00	100,00
Venice-2	7142	4	6	99,92	99,94

Tabel A.8 Hasil Uji Asosiasi dengan Fungsi *Cost* Modifikasi dan $default_cost_d$0 0,7

Video	TP	FN	FP	Precision	Recall
ADL-Rundle-6	5020	0	0	100,00	100,00
ADL-Rundle-8	6796	0	0	100,00	100,00
ETH-Bahnhof	5671	27	25	99,56	99,53
ETH-Pedcross2	6367	19	25	99,61	99,70
ETH-Sunnyday	1882	4	6	99,68	99,79
KITTI-13	675	124	203	76,88	84,48
KITTI-17	676	12	13	98,11	98,26
PETS09-S2L1	4487	0	0	100,00	100,00
TUD-Campus	355	2	2	99,44	99,44
TUD-Stadtmitte	1153	0	0	100,00	100,00
Venice-2	7141	5	8	99,89	99,93

[*Halaman ini sengaja dikosongkan*]

LAMPIRAN B: Detail Hasil Uji Coba Prediksi Kalman Filter

Tabel B.1 Hasil Uji Prediksi Kalman Filter dengan Matriks Q Orisinal

Video	$RMSE_x$	$RMSE_y$	$RMSE_w$	$RMSE_h$
ADL-Rundle-6	1,5427	1,0564	3,434	3,7206
ADL-Rundle-8	1,8303	1,4023	1,8244	2,7925
ETH-Bahnhof	5,3738	4,4301	5,7566	8,2782
ETH-Pedcross2	2,7676	2,1568	6,0619	8,4072
ETH-Sunnyday	6,0077	5,1358	5,4102	8,819
KITTI-13	8,2918	2,3262	18,2855	11,6892
KITTI-17	6,7142	1,9824	12,2543	3,1728
PETS09-S2L1	4,8599	2,0008	0,4664	0,5943
TUD-Campus	6,7034	4,6976	7,8425	5,3656
TUD-Stadtmitte	0,8513	0,4054	1,1361	0,516
Venice-2	1,3536	0,763	2,7723	1,6618

Tabel B.2 Hasil Uji Prediksi Kalman Filter dengan Matriks Q Modifikasi

Video	$RMSE_x$	$RMSE_y$	$RMSE_w$	$RMSE_h$
ADL-Rundle-6	94,1171	47,0379	3,434	3,7206
ADL-Rundle-8	74,3237	37,1837	1,8244	2,7925
ETH-Bahnhof	16,6604	8,5816	5,7566	8,2782
ETH-Pedcross2	35,1155	17,639	6,0619	8,4072
ETH-Sunnyday	21,0704	10,763	5,4102	8,819
KITTI-13	36,7402	16,921	18,2855	11,6892
KITTI-17	35,7059	17,8511	12,2543	3,1728
PETS09-S2L1	73,3704	36,7189	0,4664	0,5943
TUD-Campus	6,9089	4,1827	7,8425	5,3656
TUD-Stadtmitte	10,0829	5,0371	1,1361	0,516
Venice-2	45,8008	22,8947	2,7723	1,6618

LAMPIRAN C: Detail Hasil Uji Coba Metode Inisialisasi Objek Baru

Table C.1 Hasil Uji Inisialisasi Objek Baru dengan Metode I

Video	Recall	Precision	FP	FN	Switch	MOTA	MOTP
ADL-Rundle-6	54,9%	56,1%	2151	2258	523	1,5%	72,1%
ADL-Rundle-8	47,5%	45,3%	3889	3558	396	-15,6%	72,4%
ETH-Bahnhof	69,8%	55,5%	3035	1633	655	1,7%	73,2%
ETH-Pedcross2	15,9%	69,1%	444	5269	249	4,8%	70,2%
ETH-Sunnyday	48,5%	80,3%	221	956	152	28,5%	76,6%
KITTI-13	58,9%	43,2%	591	313	164	-40,2%	71,9%
KITTI-17	63,4%	78,4%	119	250	72	35,4%	72,0%
PETS09-S2L1	91,5%	73,4%	1484	382	478	47,6%	71,1%
TUD-Campus	64,1%	71,4%	92	129	58	22,3%	70,9%
TUD-Stadtmitte	71,8%	73,5%	299	326	95	37,7%	64,7%
Venice-2	54,1%	52,9%	3448	3276	530	-1,6%	72,8%

Tabel C.2 Hasil Uji Inisialisasi Objek Baru dengan Metode II

Video	Recall	Precision	FP	FN	Switch	MOTA	MOTP
ADL-Rundle-6	48,9%	65,7%	1280	2562	360	16,1%	72,5%
ADL-Rundle-8	41,9%	52,8%	2541	3938	244	0,9%	72,6%
ETH-Bahnhof	60,9%	63,1%	1927	2116	429	17,4%	73,5%
ETH-Pedcross2	11,7%	77,0%	218	5533	140	5,9%	70,9%
ETH-Sunnyday	40,4%	85,7%	125	1107	87	29,0%	76,6%
KITTI-13	29,5%	57,7%	165	537	54	0,8%	72,2%
KITTI-17	52,7%	92,3%	30	323	41	42,3%	72,1%
PETS09-S2L1	86,5%	83,3%	777	605	287	62,7%	71,3%
TUD-Campus	52,4%	85,8%	31	171	34	34,3%	72,5%
TUD-Stadtmitte	68,2%	82,4%	168	368	63	48,2%	64,9%
Venice-2	49,8%	57,8%	2595	3583	309	9,2%	72,9%

LAMPIRAN D: Detail Hasil Uji Coba
Batas Minimal Deteksi Beruntun

Table D.1 Hasil Uji Coba Batas Minimal Deteksi Beruntun Sebesar 2

Video	Recall	Precision	FP	FN	Switch	MOTA	MOTP
ADL-Rundle-6	43,0%	71,5%	859	2853	301	19,9%	72,9%
ADL-Rundle-8	36,4%	58,7%	1736	4315	194	7,9%	73,1%
ETH-Bahnhof	52,2%	67,2%	1381	2586	337	20,5%	73,9%
ETH-Pedcross2	9,2%	81,7%	129	5689	99	5,5%	71,2%
ETH-Sunnyday	33,1%	88,4%	81	1243	66	25,2%	76,8%
KITTI-13	21,1%	64,1%	90	601	27	5,8%	73,1%
KITTI-17	44,8%	93,6%	21	377	29	37,5%	72,4%
PETS09-S2L1	80,9%	87,0%	543	856	239	63,4%	71,4%
TUD-Campus	41,2%	90,8%	15	211	24	30,4%	73,2%
TUD-Stadtmitte	64,0%	87,8%	103	416	51	50,7%	64,8%
Venice-2	45,2%	60,3%	2125	3913	265	11,7%	73,2%

Tabel D.2 Hasil Uji Coba Batas Minimal Deteksi Beruntun Sebesar 3

Video	Recall	Precision	FP	FN	Switch	MOTA	MOTP
ADL-Rundle-6	38,3%	74,9%	641	3092	251	20,5%	73,1%
ADL-Rundle-8	32,4%	63,6%	1261	4582	149	11,7%	73,3%
ETH-Bahnhof	45,2%	69,4%	1078	2969	263	20,4%	74,2%
ETH-Pedcross2	7,3%	84,1%	87	5804	66	4,9%	71,4%
ETH-Sunnyday	28,1%	90,2%	57	1336	52	22,2%	77,1%
KITTI-13	15,7%	71,0%	49	642	10	8,0%	73,7%
KITTI-17	38,7%	92,6%	21	419	24	32,1%	72,4%
PETS09-S2L1	75,8%	88,4%	445	1085	200	61,3%	71,5%
TUD-Campus	33,7%	90,3%	13	238	18	25,1%	73,2%
TUD-Stadtmitte	60,7%	90,9%	70	454	50	50,3%	64,6%
Venice-2	41,5%	61,8%	1832	4176	221	12,8%	73,3%

Tabel D.3 Hasil Uji Coba Batas Minimal Deteksi Beruntun Sebesar 4

Video	Recall	Precision	FP	FN	Switch	MOTA	MOTP
ADL-Rundle-6	34,6%	78,0%	489	3275	212	20,6%	73,3%
ADL-Rundle-8	29,5%	67,5%	963	4784	121	13,5%	73,5%
ETH-Bahnhof	39,6%	70,6%	894	3272	180	19,7%	74,5%
ETH-Pedcross2	6,0%	85,0%	66	5888	50	4,1%	71,8%
ETH-Sunnyday	24,1%	91,2%	43	1410	42	19,5%	77,4%
KITTI-13	12,1%	77,3%	27	670	8	7,5%	74,5%
KITTI-17	33,7%	92,0%	20	453	20	27,8%	72,7%
PETS09-S2L1	71,2%	89,1%	390	1288	172	58,7%	71,5%
TUD-Campus	27,9%	91,7%	9	259	11	22,3%	72,4%
TUD-Stadtmitte	57,6%	93,1%	49	490	45	49,5%	64,6%
Venice-2	38,5%	62,9%	1624	4391	195	13,0%	73,3%

[*Halaman ini sengaja dikosongkan*]

LAMPIRAN E: Detail Hasil Uji Luar dengan Data Uji

Table E.1 Hasil Uji Coba Luar

Video	Recall	Precision	FP	FN	Switch	MOTA	MOTP
ADL-Rundle-1	40,33%	56,31%	2912	5553	302	5,79%	71,70%
ADL-Rundle-3	39,03%	70,24%	1681	6198	281	19,73%	72,67%
AVG-TownCentre	14,70%	78,14%	294	6097	108	9,08%	72,24%
ETH-Crossing	15,95%	94,67%	9	843	18	13,26%	76,76%
ETH-Jelmoli	34,61%	84,59%	160	1659	108	24,04%	74,52%
ETH-Linthescher	11,86%	95,58%	49	7871	129	9,87%	75,44%
KITTI-16	23,81%	81,16%	94	1296	118	11,35%	71,36%
KITTI-19	27,53%	72,25%	565	3872	234	12,58%	67,89%
PETS09-S2L2	33,83%	92,70%	257	6379	462	26,38%	71,53%
TUD-Crossing	43,47%	94,66%	27	623	69	34,75%	73,31%
Venice-1	25,20%	67,77%	547	3413	133	10,30%	73,17%

[*Halaman ini sengaja dikosongkan*]

BIODATA PENULIS

Penulis bernama Dandy Naufaldi, putra kedua dari dua bersaudara yang lahir di Malang pada tanggal 31 Oktober 1998. Penulis telah menjalani masa pendidikan di SMA Negeri 1 Probolinggo pada tahun 2013 hingga 2016. Pada masa penulisan, penulis sedang menempuh masa studi S1 semester kedelapan di Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember.

Selama studi S1, penulis memiliki ketertarikan pada *Machine Learning* dan *Software Engineering*. Penulis pernah menjadi asisten dosen pada mata kuliah Dasar Pemrograman, Struktur Data, Matematika Informatika, Sistem Operasi, dan Kecerdasan Buatan. Untuk mengembangkan kemampuan, penulis aktif mengikuti kompetisi di bidang *Data Mining*. Prestasi yang pernah dicapai yaitu menjadi finalis pada Gemastik 2017 dan 2018 pada bidang *Data Mining* serta pada *Data Science Competition* MCF ITB 2019. Penulis juga memiliki pengalaman magang sebagai AI *research scientist* di PT Nodeflux Teknologi Indonesia dan *software engineer* di PT Tokopedia.

Penulis juga cukup aktif berorganisasi di dalam kampus dengan menjadi staf ahli departemen Teknologi Himpunan Mahasiswa Teknik Computer-Informatika 2019/2020. Penulis aktif sebagai administrator di Laboratorium Komputasi Cerdas dan Visi dan menjadi koordinator pada 2019. Penulis juga pernah membantu dalam kepanitiaan dalam jurusan, yaitu staff *National Logic Competition* Schematics 2017-2018. Penulis dapat dihubungi melalui surel: dandy.naufaldi@gmail.com.