

**TUGAS AKHIR – KI141502**

**IMPLEMENTASI KOMPRESI ADAPTIVE MENGGUNAKAN METODE HEATSHRINK UNTUK PENGIRIMAN DATA PADA WIRELESS SENSOR NETWORK BERBASIS ZIGBEE**

**MUHAMAD HENDRI FEBRIANSYAH**

**NRP 051114410000036**

Dosen Pembimbing I

Waskitho Wibisono, S.Kom.,M.Eng.,Ph.D.

Dosen Pembimbing II

Ir. Muchammad Husni, M.Kom.

DEPARTEMEN INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI

INSTITUT TEKNOLOGI SEPULUH NOPEMBER

SURABAYA

2018



**TUGAS AKHIR – KI141502**

**IMPLEMENTASI KOMPRESI ADAPTIVE MENGGUNAKAN METODE HEATSHRINK UNTUK PENGIRIMAN DATA PADA WIRELESS SENSOR NETWORK BERBASIS ZIGBEE**

**MUHAMAD HENDRI FEBRIANSYAH**

**NRP 051114410000036**

**Dosen Pembimbing I**

**Waskitho Wibisono, S.Kom.,M.Eng.,Ph.D.**

**Dosen Pembimbing II**

**Ir. Muchammad Husni, M.Kom.**

**DEPARTEMEN INFORMATIKA**

**FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI**

**INSTITUT TEKNOLOGI SEPULUH NOPEMBER**

**SURABAYA**

**2018**

***(Halaman ini sengaja dikosongkan)***

****

**UNDERGRADUATE THESES – KI141502**

**DESIGN OF MOBILE COMPUTATION OFFLOADING FRAMEWORK FOR IMPROVE PERFORMANCE AND SAVING OF POWER IN SMARTPHONE (LEAF RECOGNITION CASE STUDY)**

**PUTRO SATRIO WIBOWO**

**NRP 5113100130**

**First Advisor**

**Waskitho Wibisono, S.Kom.,M.Eng.,Ph.D.**

**Second Advisor**

Royyana Muslim Ijtihadie, S.Kom.,M.Kom.,Ph.D.

**Department of Informatics**

**Faculty of Information Technology**

**Sepuluh Nopember Institute of Technology**

**Surabaya 2016**

***(Halaman ini sengaja dikosongkan)***

# LEMBAR PENGESAHAN

**RANCANG BANGUN *MOBILE COMPUTATION OFFLOADING FRAMEWORK* UNTUK PENINGKATAN KINERJA DAN PENGHEMATAN DAYA PADA *SMARTPHONE* (STUDI KASUS *LEAF RECOGNITION*)**

**TUGAS AKHIR**

Diajukan Untuk Memenuhi Salah Satu Syarat

Memperoleh Gelar Sarjana Komputer

pada

Bidang Studi Komputasi Berbasis Jaringan

Program Studi S-1 Jurusan Teknik Informatika

Fakultas Teknologi Informasi

Institut Teknologi Sepuluh Nopember

Oleh:

**PUTRO SATRIO WIBOWO**

**NRP: 5113100130**

Disetujui oleh Pembimbing Tugas Akhir:

1. Waskitho Wibisono, S.Kom., M.Eng., Ph.D. .....................

(NIP. 197410222000031001) (Pembimbing 1)

1. Royyana Muslim Ijtihadie, S.Kom.,

M.Kom., Ph.D. ......................

(NIP. 197708242006041001) (Pembimbing 2)

**SURABAYA**

**JUNI, 2017 *(Halaman ini sengaja dikosongkan)***

**RANCANG BANGUN *MOBILE COMPUTATION OFFLOADING FRAMEWORK* UNTUK PENINGKATAN KINERJA DAN PENGHEMATAN DAYA PADA *SMARTPHONE* (STUDI KASUS *LEAF RECOGNITION*)**

|  |  |  |
| --- | --- | --- |
| **Nama Mahasiswa** | **:** | **PUTRO SATRIO WIBOWO** |
| **NRP** | **:** | **5113100130** |
| **Jurusan** | **:** | **Teknik Informatika FTIF-ITS** |
| **Dosen Pembimbing 1** | **:** | **Waskitho Wibisono, S.Kom., M.Eng., Ph.D.** |
| **Dosen Pembimbing 2** | **:** | **Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.** |

# Abstrak

*Offloading adalah suatu metode pengeksekusian sebuah beban kerja pada sebuah perangkat dengan mengirimkan modul berisi beban kerja tersebut kepada perangkat lain yang memiliki sumber daya dan kemampuan komputasi yang lebih baik. Hasil eksekusi dari beban kerja akan diterima kembali oleh perangkat yang telah mengirim modul beban kerja sebelumnya. Teknik ini dianggap sebagai salah satu cara mengatasi keterbatasan perangkat bergerak yang memiliki sumber daya dan kemampuan komputasi yang terbatas. Oleh karena itu, diperlukan adanya penerapan metode offloading dalam mengeksekusi beban kerja pada perangkat bergerak dengan tujuan dapat melakukan penghematan sumber daya dan peningkatan kinerja pada perangkat bergerak.*

*Beban kerja dengan kriteria tertentu yang dapat di eksekusi dengan metode offloading dan terdapat faktor yang dapat menghambat proses offloading sehingga dapat tidak menguntungkan pada sumber daya dan kinerja pada perangkat bergerak. Oleh Karena itu, Tugas Akhir ini mengimplementasikan sebuah mobile framework yang dapat menentukan secara dinamis metode eksekusi yang optimal pada beban kerja yang akan dieksekusi.*

*Pada tugas akhir ini, penggunaan JADE middleware dalam implementasi metode offloading dianggap efektif dan proses image recognition digunakan sebagai beban kerja. Hasil yang didapat dari kedinamisan mobile framework dalam menentukan keputusan metode eksekusi dianggap dapat meminimalisir penggunaan daya dan meningkatkan kinerja pada perangkat bergerak. Dari uji coba didapatkan penghematan memori paling tinggi 0.259%, penghematan penggunaan CPU paling tinggi 40.379%, penghematan level baterai paling tinggi 1%, dan penghematan waktu eksekusi rata – rata paling tinggi 2.428 detik untuk setiap beban kerja.*

***Kata kunci: Offloading, mobile framework, image recognition, JADE middleware, beban kerja.***

**DESIGN OF MOBILE COMPUTATION OFFLOADING FRAMEWORK FOR IMPROVE PERFORMANCE AND SAVING OF POWER IN SMARTPHONE (LEAF RECOGNITION CASE STUDY)**

|  |  |  |
| --- | --- | --- |
| **Student’s Name** | **:** | **PUTRO SATRIO WIBOWO** |
| **Student’s ID** | **:** | **5113100130** |
| **Department** | **:** | **Teknik Informatika FTIF-ITS** |
| **First Advisor** | **:** | **Waskitho Wibisono, S.Kom., M.Eng., Ph.D.** |
| **Second Advisor** | **:** | **Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.** |

# *Abstract*

*Offloading is a method of executing a workload on a device by sending a module containing workload to other device that have better resources and computing capabilities. The execution results of the workload will be received by the device that has sent the previous workload module. This technique is considered as one of the ways to overcoming the limitations of mobile devices that have limited resources and computing capabilities. Therefore, it is necessary to apply the offloading method to executing the workload on a mobile device with purpose of saving the resources and improving performance on mobile devices.*

*Workload with certain criteria that can be executed by offloading method and there are factors that can inhibit the offloading process so it can be unfavorable on the resources and performance of mobile devices. Therefore, this study implements a mobile framework that can dynamically determine the optimal execution method on the workload to be executed.*

*In this undergraduate thesis, the use of JADE middleware in the implementation of offloading method is considered effective and image recognition process is used as workload. The results obtained from the dynamics of the mobile framework in determining the decision of the execution method are considered to minimize the use of power and improve performance on mobile devices. From the experiment, the highest memory saving is 0.259%, the maximum CPU usage is 40.379%, the battery saving rate is 1% high, and the average execution time saving is 2,428 sec for each workload.*

***Keywords : Offloading, mobile framework, image recognition, JADE middleware, workload.***

**KATA PENGANTAR**



Alhamdulillahirabbil’alamin, segala puji bagi Allah Swt yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

“**RANCANG BANGUN *MOBILE COMPUTATION OFFLOADING FRAMEWORK* UNTUK PENINGKATAN KINERJA DAN PENGHEMATAN DAYA PADA *SMARTPHONE* (STUDI KASUS *LEAF RECOGNITION*)**”

yang merupakan salah satu syarat dalam menempuh ujian sidang guna memperoleh gelar Sarjana Komputer. Selesainya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan beberapa pihak, sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Bapak Wasisto dan Ibu Nanik Wahyuni selaku orang tua penulis yang selalu memberikan dukungan doa, moral, dan material yang tak terhingga kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Waskitho Wibisono, S.Kom., M.Eng., Ph.D. selaku pembimbing I yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini sekaligus dosen wali penulis yang telah memberikan arahan, masukan dan motivasi kepada penulis.
3. Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D. selaku II yang telah membimbing dan memberikan motivasi, nasehat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
4. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku kepala jurusan Teknik Informatika ITS.
5. Bapak Dr. Eng. Radityo Anggoro, S.Kom., M.Sc. selaku koordinator dan sebagai dosen penguji Tugas Akhir penulis.
6. Bapak Tohari Ahmad, S.Kom., MIT., Ph.D. sebagai dosen penguji Tugas Akhir penulis.
7. Seluruh dosen dan karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di ITS.
8. Ibu Eva Mursidah dan Ibu Sri Budiati yang selalu mempermudah penulis dalam peminjaman buku di RBTC.
9. Teman-teman Keluarga Muslim Informatika, yang sudah banyak meluruskan penulis.
10. Teman-teman seperjuangan RMK NCC/KBJ, yang telah menemani dan menyemangati penulis.
11. Teman-teman administrator NCC/KBJ, yang telah menemani dan menyemangati penulis selama penulis menjadi administrator, menjadi rumah kedua penulis selama penulis berkuliah.
12. Teman-teman angkatan 2013, yang sudah mendukung saya selama perkuliahan.
13. Sahabat penulis yang tidak dapat disebutkan satu per satu yang selalu membantu, menghibur, menjadi tempat bertukar ilmu dan berjuang bersama-sama penulis.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan sehingga dengan kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Juni 2017

# DAFTAR ISI

LEMBAR PENGESAHAN vi

Abstrak viii

*Abstract* x

DAFTAR ISI xiv

DAFTAR GAMBAR xviii

DAFTAR TABEL xx

DAFTAR KODE SUMBER xxii

BAB I PENDAHULUAN 1

1.1 Latar Belakang 1

1.2 Rumusan Masalah 3

1.3 Batasan Permasalahan 3

1.4 Tujuan 3

1.5 Manfaat 4

1.6 Metodologi 4

1.6.1 Penyusunan Proposal 4

1.6.2 Studi Literatur 5

1.6.3 Implementasi Perangkat Lunak 5

1.6.4 Pengujian dan Evaluasi 5

1.6.5 Penyusunan Buku 6

1.7 Sistematika Penulisan Laporan 6

BAB II TINJAUAN PUSTAKA 9

2.1 Sistem Operasi Android 9

2.2 *Offloading* 9

2.3 Faktor Penentu Metode *Offloading* 11

2.4 JADE 11

2.5 GSON 12

2.6 Commons-Lang 12

2.7 OpenCV 12

2.8 SURF 13

BAB III PERANCANGAN PERANGKAT LUNAK 15

3.1 Data 15

3.1.1 Data Masukan 15

3.1.2 Data Keluaran 17

3.2 Desain Umum Sistem 18

3.3 JADE *Middleware* 22

3.4 Faktor Penentu Metode *Offloading* 29

3.4.1 Kualitas Koneksi Internet 29

3.4.2 Kondisi Level Baterai Perangkat Bergerak 30

3.4.3 Waktu Eksekusi Proses *Image Recognition* 30

3.5 *Offloading Framework* 31

3.6 Metode SURF 33

3.7 Java *Class Object Serialization* dari *Client* ke *Server* 35

3.8 Java *Class Object Serialization* dari *Server* ke *Client* 36

BAB IV IMPLEMENTASI 37

4.1 Lingkungan Implementasi 37

4.2 Implementasi 38

4.2.1 JADE *Middleware* 39

4.2.1.1 Client 39

4.2.1.2 Server 49

4.2.2 *Offloading Framework* 53

4.2.3 Metode SURF 69

4.2.4 Java *Class Object Serialization* dari *Client* ke *Server* 75

4.2.5 Java *Class Object Serialization* dari *Server* ke *Client* 79

4.2.6 Faktor Penentu Metode *Offloading* 83

4.2.6.1 Kualitas Koneksi Internet 83

4.2.6.2 Kondisi Level Baterai Perangkat Bergerak 86

4.2.6.3 Waktu Eksekusi Proses *Image Recognition* 87

BAB V HASIL UJI COBA DAN EVALUASI 89

5.1 Lingkungan Pengujian 89

5.2 Data Pengujian 90

5.3 *Preprocessing* citra 90

5.4 Skenario Uji Coba 91

5.4.1 Skenario Uji Coba 1 94

5.4.2 Skenario Uji Coba 2 104

5.4.3 Skenario Uji Coba 3 110

5.4.4 Skenario Uji Coba 4 115

5.4.5 Skenario Uji Coba 5 121

5.4.6 Skenario Uji Coba 6 127

5.5 Evaluasi Umum Skenario Uji Coba 132

BAB VI KESIMPULAN DAN SARAN 139

6.1 Kesimpulan 139

6.2 Saran 140

DAFTAR PUSTAKA 141

LAMPIRAN 143

BIODATA PENULIS 145

***(Halaman ini sengaja dikosongkan)***

# DAFTAR GAMBAR

[Gambar 2.1 Alur eksekusi beban kerja pada perangkat bergerak ke perangkat komputasi awan [1] 10](#_Toc487460636)

[Gambar 3.1 Contoh data masukan citra yang digunakan sebagai datatest 16](#_Toc487460637)

[Gambar 3.2 Contoh data masukan citra yang digunakan sebagai dataset 17](#_Toc487460638)

[Gambar 3.3 Proses image recognition menggunakan metode SURF secara garis besar 21](#_Toc487460639)

[Gambar 3.4 Proses image recognition menggunakan metode SURF dengan offloading framework 22](#_Toc487460640)

[Gambar 3.5 Container dan Platform pada JADE middleware [12] 23](#_Toc487460641)

[Gambar 3.6 Mode eksekusi pada runtime JADE [13] 25](#_Toc487460642)

[Gambar 3.7 Langkah eksekusi agent menggunakan thread [12] 27](#_Toc487460643)

[Gambar 3.8 Paradigma pengiriman pesan JADE secara asynchronous [12] 28](#_Toc487460644)

[Gambar 3.9 Contoh pengelompokan kualitas koneksi berdasarkan estimasi buffer data dan buffer data actual [14] 30](#_Toc487460645)

[Gambar 3.10 Proses image recognition pada citra daun 35](#_Toc487460646)

[Gambar 5.1 Bagan alur kerja skenario uji coba 93](#_Toc487460647)

[Gambar 5.2 Grafik performa tanpa penerapan framework pada perangkat Android 97](#_Toc487460648)

[Gambar 5.3 Grafik waktu eksekusi tanpa penerapan framework pada perangkat lunak image recognition 99](#_Toc487460649)

[Gambar 5.4 Grafik performa dengan penerapan framework pada Koneksi A dan Baterai A 101](#_Toc487460650)

[Gambar 5.5 Grafik waktu eksekusi dengan penerapan framework pada Koneksi A dan Baterai A 103](#_Toc487460651)

[Gambar 5.6 Grafik performa dengan penerapan framework pada Koneksi B dan Baterai A 107](#_Toc487460652)

[Gambar 5.7 Grafik waktu eksekusi dengan penerapan framework pada Koneksi B dan Baterai A 109](#_Toc487460653)

[Gambar 5.8 Grafik performa dengan penerapan framework pada Koneksi A dan Baterai B 112](#_Toc487460654)

[Gambar 5.9 Grafik waktu eksekusi dengan penerapan framework pada Koneksi A dan Baterai B 114](#_Toc487460655)

[Gambar 5.10 Grafik performa dengan penerapan framework pada Koneksi B dan Baterai B 118](#_Toc487460656)

[Gambar 5.11 Grafik waktu eksekusi dengan penerapan framework pada Koneksi B dan Baterai B 120](#_Toc487460657)

[Gambar 5.12 Grafik performa dengan penerapan framework pada Koneksi C dan Baterai A 124](#_Toc487460658)

[Gambar 5.13 Grafik waktu eksekusi dengan penerapan framework pada Koneksi C dan Baterai A 126](#_Toc487460659)

[Gambar 5.14 Grafik performa dengan penerapan framework pada Koneksi C dan Baterai B 129](#_Toc487460660)

[Gambar 5.15 Grafik waktu eksekusi dengan penerapan framework pada Koneksi C dan Baterai B 131](#_Toc487460661)

# DAFTAR TABEL

[Tabel 4.1 Lingkungan Implementasi Perangkat Lunak 37](#_Toc487381170)

[Tabel 5.1 Spesifikasi Lingkungan Pengujian 89](#_Toc487381171)

[Tabel 5.2 Pembagian kategori koneksi internet 91](#_Toc487381172)

[Tabel 5.3 Pembagian kategori level baterai 92](#_Toc487381173)

[Tabel 5.4 Performa tanpa penerapan framework pada perangkat Android 95](#_Toc487381174)

[Tabel 5.5 Waktu eksekusi tanpa penerapan framework pada perangkat lunak image recognition 97](#_Toc487381175)

[Tabel 5.6 Performa dengan penerapan framework pada Koneksi A dan Baterai A 99](#_Toc487381176)

[Tabel 5.7 Waktu eksekusi dengan penerapan framework pada Koneksi A dan Baterai A 101](#_Toc487381177)

[Tabel 5.8 Performa dengan penerapan framework pada Koneksi B dan Baterai A 105](#_Toc487381178)

[Tabel 5.9 Waktu eksekusi dengan penerapan framework pada Koneksi B dan Baterai A 107](#_Toc487381179)

[Tabel 5.10 Performa dengan penerapan framework pada Koneksi A dan Baterai B 111](#_Toc487381180)

[Tabel 5.11 Waktu eksekusi dengan penerapan framework pada Koneksi A dan Baterai B 113](#_Toc487381181)

[Tabel 5.12 Performa dengan penerapan framework pada Koneksi B dan Baterai B 116](#_Toc487381182)

[Tabel 5.13 Waktu eksekusi dengan penerapan framework pada Koneksi B Baterai B 118](#_Toc487381183)

[Tabel 5.14 Performa dengan penerapan framework pada Koneksi C dan Baterai A 122](#_Toc487381184)

[Tabel 5.15 Waktu eksekusi dengan penerapan framework pada Koneksi C dan Baterai A 124](#_Toc487381185)

[Tabel 5.16 Performa dengan penerapan framework pada Koneksi C dan Baterai B 128](#_Toc487381186)

[Tabel 5.17 Waktu eksekusi dengan penerapan framework pada Koneksi C dan Baterai B 130](#_Toc487381187)

[Tabel 5.18 Hasil performa dan waktu eksekusi dari 6 skenario uji coba 133](#_Toc487381188)

# DAFTAR KODE SUMBER

[Pseudocode 4.1 Kode program menghubungkan JADE runtime dengan MicroRuntimeService 40](#_Toc487381191)

[Pseudocode 4.2 Kode Program membuat Container 41](#_Toc487381192)

[Pseudocode 4.3 Kode Program membuat Agent 43](#_Toc487381193)

[Pseudocode 4.4 Kode program inisialisasi variabel agent pada client 43](#_Toc487381194)

[Pseudocode 4.5 Kode program metode Setup pada client 44](#_Toc487381195)

[Pseudocode 4.6 Kode program generic behaviour pada client 49](#_Toc487381196)

[Pseudocode 4.7 Kode program inisialisasi variabel agent pada server 49](#_Toc487381197)

[Pseudocode 4.8 Kode program metode Setup pada server 50](#_Toc487381198)

[Pseudocode 4.9 Kode program Ticker behaviour pada server 53](#_Toc487381199)

[Pseudocode 4.10 Kode program inisialisasi perangkat client dalam memulai framework 55](#_Toc487381200)

[Pseudocode 4.11 Kode program inisialisasi penerapan Asynchronous Task pada framework 57](#_Toc487381201)

[Pseudocode 4.12 Kode program pengecekan status eksekusi beban kerja image recognition 58](#_Toc487381202)

[Pseudocode 4.13 Kode program metode eksekusi beban kerja image recognition pada framework 61](#_Toc487381203)

[Pseudocode 4.14 Kode program kode pemantauan eksekusi beban kerja image recognition pada framework 62](#_Toc487381204)

[Pseudocode 4.15 Kode program Decision Maker melakukan pengecekan terhadap ketersediaan koneksi internet dan microRuntimeServiceBinder 64](#_Toc487381205)

[Pseudocode 4.16 Kode program Decision Maker melakukan pengecekan terhadap kualitas koneksi internet 65](#_Toc487381206)

[Pseudocode 4.17 Kode program Decision Maker melakukan pengecekan terhadap kondisi level baterai 65](#_Toc487381207)

[Pseudocode 4.18 Kode program Decision Maker melakukan pembobotan nilai untuk keputusan metode eksekusi 67](#_Toc487381208)

[Pseudocode 4.19 Kode program Decision Maker melakukan metode eksekusi lokal jika tidak tersedianya koneksi internet pada perangkat client 68](#_Toc487381209)

[Pseudocode 4.20 Kode program menyimpan hasil eksekusi beban kerja pada framework 69](#_Toc487381210)

[Pseudocode 4.21 Program kompresi citra datatest 70](#_Toc487381211)

[Pseudocode 4.22 Kode Program mendapatkan descriptor pada datatest 71](#_Toc487381212)

[Pseudocode 4.23 Kode program eksekusi image recognition secara lokal 73](#_Toc487381213)

[Pseudocode 4.24 Kode Program eksekusi image recognition secara offloading 74](#_Toc487381214)

[Pseudocode 4.25 Kode program pengubahan data matriks menjadi string pada perangkat client 76](#_Toc487381215)

[Pseudocode 4.26 Kode Program kelas Java Object Serialization ObjectDataMat pada perangkat client 77](#_Toc487381216)

[Pseudocode 4.27 Kode Program kelas Java Object Serialization ObjectDataMat pada perangkat server 78](#_Toc487381217)

[Pseudocode 4.28 Kode program pengubahan string menjadi Data Matriks pada perangkat server 79](#_Toc487381218)

[Pseudocode 4.29 Kode Program kelas Java Object Serialization data pada perangkat server 81](#_Toc487381219)

[Pseudocode 4.30 Kode Program kelas Java Object Serialization data pada perangkat client 83](#_Toc487381220)

[Pseudocode 4.31 Kode Program inisialisasi pengecekan koneksi internet pada perangkat client 84](#_Toc487381221)

[Pseudocode 4.32 Kode Program kondisi saat koneksi internet mengalami perubahan kualitas dan pengecekan ketersediaan 85](#_Toc487381222)

[Pseudocode 4.33 Kode program inisialisasi penerapan Asynchronous Task pada pengecekan buffer data 86](#_Toc487381223)

[Pseudocode 4.34 Kode program mendapatkan nilai level baterai pada perangkat client 86](#_Toc487381224)

[Pseudocode 4.35 Kode program perhitungan waktu dimulainya beban kerja dieksekusi 87](#_Toc487381225)

[Pseudocode 4.36 Kode program perhitungan waktu selesainya beban kerja dieksekusi secara offloading 88](#_Toc487381226)

[Pseudocode 4.37 Kode program perhitungan waktu selesainya beban kerja dieksekusi secara lokal 88](#_Toc487381227)

***(Halaman ini sengaja dikosongkan)***

# BAB I PENDAHULUAN

## Latar Belakang

Wireless Sensor Network merupakan salah satu teknologi yang paling menjanjikan untuk kebutuhan di masa depan. Hal ini dikarenakan harganya yang murah, mudah dibangun, terdapat sensor cerdas, ukurannya kecil dan multi fungsionalitas sesuai kebutuhan. Berdasarkan research dari IDTechEx, diprediksi bahwa pasar WSN akan mengalami pertumbuhan menjadi $ 1,8 miliar pada tahun 2024. Data ini mengacu pada WSN yang didefinisikan sebagai jaringan mesh nirkabel, yaitu self-healing dan self-organising [1].

Terdapat banyak permasalahan dan tantangan yang harus dihadapi dalam WSN untuk meningkatkan efisiensi, kelayakan dan manfaat. Tantang tersebut dapat dikategorikan kedalam empat kategori, yaitu efisiensi daya, pengumpulan data, jaringan dan strategi penyebaran [2]. WSN pada dasarnya adalah system yang berbasis event, node sensor akan mendeteksi keadaan di lingkungan sekitarnya untuk dikirim ke sink. Namun, karena node sensor sering mendeteksi fenomena umum, maka kemungkinan ada beberapa redundansi dalam data yang sumbernya beragam berkomunikasi dengan sink tertentu. Untuk itu pemfilteran dan pemrosesan dalam jaringan diperlukan agar dapat menghemat penggunaan energi yang terbatas.

Agregasi data merupakan proses pengumpulan data dari berbagai node untuk menghilangkan redundansi, meminimalkan jumlah transmisi dan memberikan informasi yang ringkas ke simpul utama [3]. Tujuan agregasi data adalah untuk memperpanjang umur jaringan dengan mengurangi transmisi waktu atau ukuran data yang dipancarkan node menggunakan algoritma cerdas. Secara umum, agregasi data dapat dikategorikan menjadi dua subsistem yang berbeda, yaitu protocol jaringan dan penggabungan data. Gagasan protocol agregasi data pada jaringan adalah bekerjasama antar node spasial dan temporal berkorelasi [4] dalam menyebarkan data yang dikumpulkan. Pendekatan ini telah banyak diajukan dalam penelitian seperti LEECH, TEEN, HEED dan PEGASIS. Disisi lain penggabungan data bertujuan untuk mengurangi ukuran data yang ditransmisikan oleh sink atau node itu sendiri. Ukuran data dapat dikurangi menggunakan teknik operasi matematika [5] (median, average, moving average), kompresi, estimasi data dan pemodelan.

Kompresi data merupakan salah satu metode efektif untuk mengurangi penggunaan daya yang terbatas pada WSN. Diasumsikan bahwa beberapa kehilangan presisi atau kedetailan data pada saat kompresi dapat ditolerir jika hal tersebut dapat mengurangi komunikasi. Namun, perlu diperhatikan juga bahwa kualitas kedetailan data harus dipenuhi pada saat kondisi tertentu agar informasi yang didapatkan semakin banyak dan detail.

Disisi lain, menggunakan pendekatan agregasi data akan mempengaruhi perilaku komunikasi pada jaringan. Transmisi data yang harusnya kontinu diubah ke transmisi buffer berdasarkan kemampuan pemrosesan data lokal. Dengan menggabungkan agregasi data dengan komunikasi buffer maka ukuran paket data akan meningkat. Dengan demikian, slot waktu yang dibutuhkan untuk mentransmisikan paket data juga ikut meningkat. Jika terjadi suatu kondisi dimana beberapa node ingin mentransmisikan datanya secara bersamaan, maka node forwarding masing-masing lingkungan multihop akan menjadi hambatan, terutama pada jaringan yang berdaya rendah dan bandwidth yang terbatas. Maka buffer overflows dan rasio paket loss yang meningkat akan menjadi masalah besar [6].

Pada kodisi nyata, jaringan komunikasi pada wireless sensor network memiliki buffer yang sangat kecil. Di nrf24l01+ payload data yang disediakan hanya 32 byte dan di IEEE 802.15.4 payload data sebesar 133 bytes. Belum lagi jika kita menggunakan modul tambahan. Pada Xbee beban buffer yang dialokasikan untuk 802.15.4 dan ZigBee masing-masing adalah 100 dan 72 byte [7].

Oleh karena itu, dibutuhkan suatu managemen pengunaan paket yang baik untuk menghindari terjadinya buffer overflows. Dalam tugas akhir ini metode yang diusulkan adalah implementasi kompresi adaptive menggunakan metode Heatshrink untuk pengiriman data pada wireless sensor network berbasis Zigbee. Algoritma kompresi data yang digunakan adalah Heatshrink. Algoritma ini berbasis pada algoritma Lempel-Ziv-Storer-Szymanski (LZSS) yang merupakan lossless kompresi data yang cocok untuk kompresi data pada embedded system.

## Rumusan Masalah

Tugas akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana cara menentukan level kompresi data?

2. Bagaimana metode yang digunakan dalam pengiriman data pada protocol ZigBee?

3. Bagaimana tingkat efisiensi dari sistem yang dibangun dapat diukur?

## Batasan Permasalahan

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

1. Mengunakan mikrokontroller Arduino Mega.
2. Komunikasi nirkabel menggunakan protocol ZigBee
3. Menggunakan algoritma Heatshrink untuk melakukan kompresi dan dekompresi data*.*
4. Algoritma Heatshrink dibuat dalam bahasa pemrograman C.

## Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Membangun sebuah *offloading computation* pada *mobile framework* yang dapat menentukan secara dinamis dan optimal suatu beban kerjadiproses secara *offloading computation* ke *server* atau secara lokal berdasarkan kondisi faktor – faktor pendukung yang dialami saat itu oleh perangkat bergerak di sisi *client*.
2. Melakukan implementasi metode *offloading* untuk penghematan daya dan peningkatan kinerja perangkat bergerak.

## Manfaat

Dengan dibuatnya tugas akhir ini diharapkan dapat memberikan manfaat untuk menghasilkan sebuah *Offloading computation mobile framework* yang dapat menentukan secara dinamis pemrosesan suatu beban kerja yang optimal dengan tujuan mengurangi penggunaan sumber daya baterai dan memaksimalkan performa komputasi agar memiliki waktu eksekusi seminimal mungkin.

Sedangkan bagi penulis, tugas akhir ini bermanfaat sebagai sarana untuk mengimplementasikan ilmu dan algoritma *image recognition* serta pemrosesannya yang dilakukan secara *thread processing* yang telah dipelajari selama kuliah agar berguna bagi masyarakat.

## Metodologi

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

### Penyusunan Proposal

Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan terdiri atas hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu, dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Sub bab metodologi berisi penjelasan tahapan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal ginggu penyususunan buku tugas akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

### Studi Literatur

Tahap studi literatur merupakan tahap pembelajaran dan pengumpulan informasi yang digunakan untuk mengimplementasikan Tugas Akhir. Tahap ini diawali dengan pengumpulan literatur, diskusi, eksplorasi teknologi, dan pustaka, serta pemahaman dasar teori yang digunakan pada topik tugas akhir. Literatur-literatur yang dimaksud disebutkan sebagai berikut:

### Analisis dan Desain Perangkat Lunak

Pada tahap ini akan dilakukan analisa, perancangan, dan pendefinisian kebutuhan system untuk mengetahui permasalahan yang akan dihadapi pada tahap implementasi. Kemudian akan dijabarkan kebutuhan-kebutuhan tersebut ke dalam perancangan fitur sistem. Berikut langkah yang akan dilakukan perancangan proses perangkat lunak:

1. Perancangan rangkaian node yang akan dibuat
2. Uji coba komunikasi menggunakan protocol ZigBee pada rangkaian node
3. Implementasi kompresi dan dekompresi pada node
4. Implementasi kompresi adaptive pada node

### Implementasi Perangkat Lunak

Implementasi merupakan tahap untuk membangun metode-metode yang sudah diajukan pada proposal Tugas Akhir. Sistem dibangun menggunakan Arduino IDE (Integrated Development Environment), dengan bahasa pemrograman C dan algoritma Heatshrink.

### Pengujian dan Evaluasi

Tahap pengujian dan evaluasi berisi pengujian aplikasi dan evaluasi berdasarkan hasil pengujian. Pada tahap ini dilakukan pengujian dari fungsionalitas dan performa system WSN yang mana nanatinya akan dijalankan scenario yang sudah ditentukan. Hasil evaluasi menjabarkan tentang rangkuman hasil pengujian yang telah dilakukan. Pengujian fungsionalitas meliputi uji coba setiap bagian perangkat keras yang dirangkai pada Arduino dan juga uji coba keseluruhan sistem. Pengujian performa meliputi tingkat akurasi hasil kompresi data dan efisiensi data yang dapat di hemat.

### Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencangkup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

## Sistematika Penulisan Laporan

Sistematika penulisan laporan tugas akhir adalah sebagai berikut:

* + 1. Bab I. Pendahuluan

Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.

* + 1. Bab II. Tinjauan Pustaka

Bab ini memaparkan hasil studi literatur yang digunakan sebagai dasar untuk menyelesaikan tugas akhir ini, terdiri atas deskripsi mengenai wireless sensor network, mikrokontroller, protokol ZigBee, bahasa pemrograman C, dan algoritma Heatshrink*.*

* + 1. Bab III. Perancangan Perangkat Lunak

Bab ini berisi pembahasan mengenai perancangan dari *offloading computation framework* yang diterapkan pada perangkat bergerak Android untuk menentukan dilakukannya metode *offloading* pada suatu beban kerja saat melakukan *image recognition* berdasarkan faktor – faktor tertentu.

* + 1. Bab IV. Implementasi

Bab ini menjelaskan implementasi yang berbentuk *Pseudocode* yang berupa *Pseudocode* dari *offloading computation framework* dari sistem perangkat lunak *image recognition* beserta penerapan JADE *middleware* padaclient (perangkatbergerakAndroid) danserver (PC)*.*

* + 1. Bab V. Hasil Uji Coba dan Evaluasi

Bab ini berisikan hasil uji coba dari *offloading computation framework* yang digunakan untuk menentukan dilakukannya metode *offloading* pada suatu beban kerja dari sistem perangkat lunak *image recognition* yang sudah diimplementasikan pada *Pseudocode*. Uji coba dilakukan dengan menggunakan datatest dan dataset citra yang memiliki kualitas rendah*.* Hasil evaluasi mencakup kedinamisan dari kemampuan *offloading computation framework* dalam menentukan dilakukannya metode *offloading* pada beban kerja berdasarkan faktor – faktor penentu yang di alami perangkat client dan server saat itu.

* + 1. Bab VI. Kesimpulan dan Saran

Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.

* + 1. Daftar Pustaka

Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.

* + 1. Lampiran

Dalam lampiran terdapat tabel-tabel data hasil uji coba dan *Pseudocode* program secara keseluruhan.

***(Halaman ini sengaja dikosongkan)***

# BAB II TINJAUAN PUSTAKA

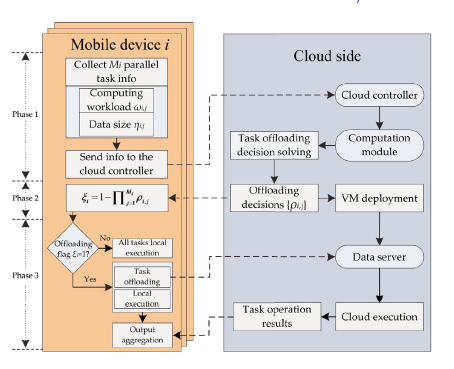
Bab ini berisi pembahasan mengenai teori-teori dasar yang digunakan dalam tugas akhir. Teori-teori tersebut diantaranya adalah sistem operasi Android sebagai OS, metode *offloading*, metode SURF untuk *image recognition,* JADE sebagai *middleware*,modul pengiriman data antara *client* dan *server* dan beberapa teori lain yang mendukung pembuatan tugas akhir.

## Sistem Operasi Android

Android adalah sistem operasi berbasis Linux yang dirancang untuk perangkat bergerak layar sentuh seperti telepon pintar dan komputer tablet [4]. Android awalnya dikembangkan oleh Android, Inc., dengan dukungan finansial dari Google, yang kemudian membelinya pada tahun 2005 [4]. Android bersifat *open source* yaitu memberikan *developers* untuk mengembangkan versi pribadi mereka dari sistem operasi Android [5]. *Smartphone* berbasis Android umumnya memiliki daya tahan baterai lebih pendek dibandingkan iOS dari iPhone Apple dengan baterai yang lebih besar dan kontrol perangkat lunak pihak ketiga yang ketat. Oleh karena itu, untuk mengurangi tingkat konsumsi daya melalui *background services* merupakan masalah penting untuk komputasi perangkat *mobile* Android. Hal tersebut adalah salah satu alasan Google menerbitkan sebagian besar *source code* untuk Android [5] Dari fakta tersebut, sistem operasi Android sangat cocok untuk digunakan untuk membangun *mobile computation offloading framework*.

## *Offloading*

Metode *Offloading* telah menjadi teknik yang menjanjikan untuk memecahkan masalah yang dihadapi perangkat *smartphone*, yaitu dengan memungkinkan *smartphone* melakukan metode *offload* atau mengirimkan suatu beban kerja komputasi yang intensif ke server [5]. Berbagai upaya telah dilakukan untuk menerapkan metode *offload* pada aplikasi Java ke server untuk memperoleh keuntungan dari kapabilitas *crossplatform* *bytecode* Java [5]. Eksekusi secara *remote* pada server dapat menjadi solusi untuk keterbatasan perangkat *mobile* dan dapat dilihat sebagai cara memperpanjang daya tahan baterai yang merupakan tujuan penerapan *offloading* [6]. Contoh alur eksekusi beban kerja pada penerapan metode *offloading* yang diimplementasikan pada perangkat bergerak dapat dilihat pada Gambar 2.1.



Gambar 2.1 Alur eksekusi beban kerja pada perangkat bergerak ke perangkat komputasi awan [1]

## Faktor Penentu Metode *Offloading*

Keputusan dalam menentukan dilakukannya metode *Offloading* adalah tantangan. Hal ini bergantung pada volume data dan transmisi *bandwidth*, konteks yang di eksekusi dari beban kerja komputasi, perbandingan dalam kemampuan eksekusi antara SoC (*system on chip*) *smartphone* dan *processor* server serta seberapa efek keuntungan jika melakukan metode *Offloading* [5]. Membuat data rinci sebuah beban kerja komputasi dan pemantauan jaringan *quality-of-service* (QoS) sangat penting dalam *framework*. Selain itu, karena komputasi dan karakteristik komunikasi adalah dinamis, sebuah *daemon* yang secara periodik mengukur kapabilitas komputasi, performa komunikasi, konsumsi daya dan secara dinamis membuat keputusan dilakukannya metode *offloading* untuk setiap *task* yang memenuhi syarat untuk dilakukannya *offloading*, hal ini dibutuhkan dalam *framework* [5].

## JADE

JADE adalah singkatan dari *Java Agent DEvelopment Framework*. JADE adalah perangkat lunak *framework* yang sepenuhnya diimplementasikan dalam bahasa Java. JADE mempermudah penggunanya dalam mengimplementasikan sistem *multi-agent* melalui sebuah *middleware* yang sesuai dengan spesifikasi FIPA dan seperangkat alat tampilan grafis yang mendukung fase *debugging* dan *deployment*. Sistem berbasis JADE dapat didistribusikan ke sebagian besar perangkat (meskipun memiliki Sistem Operasi yang berbeda) dan konfigurasi dapat dikendalikan melalui *remote* GUI. Konfigurasi dapat diubah pada saat *runtime* dengan memindahkan agen dari satu perangkat ke perangkat lainnya sesuai kebutuhan. JADE secara penuh diimplementasikan pada Bahasa Java dengan persyaratan minimal sistem adalah versi 5 Java (*runtime* *environment* atau JDK) [7].

## GSON

GSON *library* awalnya dikembangkan untuk keperluan internal Google. GSON adalah *library* Java yang bisa digunakan untuk mengonversi Objek Java menjadi versi representasi berbentuk JSON. GSON juga dapat digunakan untuk mengubah *string* JSON menjadi Objek Java yang ekuivalen. GSON dapat bekerja Objek Java yang bermacam – macam [8].

## Commons-Lang

Commons-Lang adalah Java *library* yang menyediakan metode untuk memanipulasi kelas inti Java. Apache Commons-Lang menyediakan metode tambahan yaitu sejumlah utilitas pembantu untuk API java.lang, terutama metode manipulasi String, metode numerik dasar, refleksi objek, konkurensi, pembuatan dan serialisasi. Selain itu, perangkat ini berisi perangkat tambahan dasar untuk java.util.Date dan serangkaian utilitas yang didedikasikan untuk menjalankan beberapa metode, seperti *hashCode*, *toString* dan *equals* [9].

## OpenCV

*Open*CV (*Open Source Computer Vision*) adalah *library* yang utamanya digunakan untuk pemrosesan visi komputer. *Open*CV adalah *library* gratis yang dapat digunakan di berbagai *platform*, sepertiGNU/Linux maupun Windows. *Open*CV mulanya ditulis dalam bahasa pemrograman C++, namun saat ini *Open*CV dapat digunakan pada berbagai bahasa seperti Python, Java, atau MATLAB [10].

## SURF

Dalam visi computer khususnya *image recognition*, *Speeded Up Robust Features* (SURF) adalah *local feature detector* dan *descriptor* yang dipatenkan. Metode ini dapat digunakan untuk pengenalan objek dan gambar, klasifikasi dan rekonstruksi 3D. Metode ini sebagian terinspirasi oleh *Scale Invariant* *feature transform* (SIFT) *descriptor*. Versi standar SURF beberapa kali lebih cepat dari SIFT dan diklaim oleh penemunya untuk menjadi lebih tangguh terhadap transformasi citra yang berbeda daripada SIFT. Untuk mendeteksi titik – titik penting fitur pada citra, SURF menggunakan pendekatan bilangan bulat dari determinan *Hessian blob detector*. Deskriptor fitur didasarkan pada jumlah respon *Haar wavelet* di sekitar titik – titik penting fitur. Deskriptor SURF telah digunakan untuk menemukan dan mengenali benda, orang atau wajah, untuk merekonstruksi adegan 3D, untuk melacak objek dan untuk mengekstrak titik penting fitur citra [11].

***(Halaman ini sengaja dikosongkan)***

# BAB III PERANCANGAN PERANGKAT LUNAK

Bab ini membahas mengenai perancangan dan pembuatan sistem *framework* pada perangkat lunak Android. Sistem *framework* pada perangkat lunak Android yang dibuat pada tugas akhir ini adalah menentukan keputusan dilakukannya metode *offloading* berdasarkan faktor – faktor penentu yang mendukung metode *offloading.*

## Data

Pada sub bab ini akan dijelaskan mengenai data yang digunakan sebagai masukan pada perangkat lunak Android untuk selanjutnya diolah dan dilakukan pengujian secara optimal sehingga menghasilkan data keluaran yang diharapkan dengan waktu pemrosesan dan penggunaan sumber daya seminimal mungkin.

### Data Masukan

Data masukan adalah data yang digunakan sebagai masukan awal dari sistem. Data yang digunakan dalam perangkat lunak Android terdiri dari dua jenis data masukan. Data masukan jenis pertama adalah sebuah datatest berupa sebuah citra daun yang dianggap belum diketahui jenisnya. Data masukan jenis pertama diperoleh dengan menggunakan kamera yang terpasang di perangkat Android dan dilakukan pengambilan gambar saat menjalankan perangkat lunak Android. Data masukan jenis pertama yang belum diketahui jenisnya tersebut digunakan sebagai datatest untuk uji coba prediksi pada *image recognition* menggunakan metode SURF untuk ekstraksi fitur.

Sedangkan data masukan jenis kedua adalah sebuah dataset terdiri dari 40 buah citra daun yang dapat dibagi 10 citra daun berdasarkan jenis daunnya sehingga setiap jenis daun memiliki 4 citra daun. Data masukan jenis kedua telah dilakukan pemrosesan *preprocessing* guna memaksimalkan citra daun untuk proses *image recognition*. Hal yang dilakukan pada tahap *preprocessing* secara umum yaitu mengurangi *noise*, *blurring*, *resizing*, konversi citra ke *grayscale* atau representasi warna lainnya. Data masukan jenis kedua yang berjumlah 40 tersebut digunakan sebagai data masukan untuk uji coba pencocokan dengan datatest pada *image recognition* menggunakan metode SURF untuk ekstraksi fitur.

Contoh citra sebagai data masukan datatest dan dataset ditunjukkan pada Gambar 3.1 dan Gambar 3.2.



Gambar 3.1 Contoh data masukan citra yang digunakan sebagai datatest



Gambar 3.2 Contoh data masukan citra yang digunakan sebagai dataset

### Data Keluaran

Data masukan akan dilakukan *image recognition* dengan menggunakan metode SURF sebagai ekstraksi fitur*.* Terdapat dua buah data masukan yaitu data citra yang digunakan sebagai datatest dan data citra yang digunakan sebagai dataset.Hasil dari proses *image recognition* antara datatest dengan masing – masing dataset adalah suatu fitur atau ciri – ciri berupa titik – titik pada citra yang dapat digunakan sebagai pembeda pada setiap jenis daun. Dari ciri – ciri tersebut dapat dihitung jarak kedekatan dan kemiripan antara fitur yang dimiliki datatest dan fitur yang dimiliki masing – masing dataset.

## Desain Umum Sistem

Rancangan sistem *framework* pada perangkat lunak *image recognition* menggunakan metode SURF berbasis Android, dimulai dengan mengambil foto gambar citra daun menggunakan kamera yang terpasang pada perangkat bergerak Android. Foto gambar citra daun yang didapat selanjutnya akan digunakan sebagai datatest dan di kompresi agar memiliki kualitas yang sama dengan gambar daun yang digunakan sebagai dataset, kemudian dikonversi menggunakan warna *grayscale*. Tahap selanjutnya adalah membandingkan fitur citra daun datatest dan fitur masing – masing citra daun dataset dengan *image recognition*, yang berarti fitur – fitur yang berada pada satu citra daun datatest akan dibandingkan dengan fitur – fitur yang berada pada setiap citra daun dataset. Metode SURF pada *image recognition* digunakan untuk menghitung jarak kedekatan atau kemiripan antara fitur pada sebuah citra daun datatest dan fitur pada sebuah citra daun dataset. Semakin kecil/mendekati nol nilai jarak kedekatan atau kemiripan antara citra daun datatest dan citra daun dataset maka semakin mirip dan identik kedua citra daun tersebut. Diagram alir langkah kerja secara garis besar *image recognition* pada citra daun datatest dan citra daun dataset menggunakan metode SURF, ditunjukkan pada Gambar 3.3.

Langkah kerja proses *image recognition* pada perangkat lunak ini dimodifikasi dengan menambahkan metode *offloading* dengan tujuan meminimalisir penggunaan sumber daya dengan waktu eksekusi yang optimal. Metode *offloading* dapat meminimalisir penggunaan sumber daya dan mempercepat waktu eksekusi dengan cara mengirimkan beban kerja proses pada perangkat *client* ke *server* yang memiliki kemampuan komputasi jauh lebih baik dan sumber daya tak terbatas*,* hasil yang didapatkan akan dikirimkan kembali ke perangkat client. Untuk menggabungkan langkah kerja proses *image recognition* dengan metode *offloading*, dibutuhkan sebuah *framework* untuk mengurutkan setiap langkah kerja secara runtut serta menentukan beban kerja yang akan di *offload* ke *server*. Selain itu, *framework* juga menentukan eksekusi suatu beban kerja dilakukan secara lokal atau *offloading* sebab metode *offloading* memiliki beberapa persyaratan agar mengeksekusi beban kerja secara *offloading* ke *server* lebih optimal dan menghemat sumber daya daripada eksekusi secara lokal pada perangkat *client*.

Sebelum tahap proses *image recognition* menggunakan metode SURFdilakukan, *framework* melakukan pengambilan data inisialisasi yang digunakan sebagai faktor penentu keputusan dilakukannya metode *offloading* pada saat perangkat lunak melakukan pengambilan foto gambar citra daun sebagai datatest. Metode *offloading* diperlukan sebab proses *image recognition* menggunakan metode SURF membutuhkan sumber daya yang cukup besar sedangkan perangkat bergerak Android memiliki sumber daya baterai dan kemampuan komputasi yang terbatas. Data yang diambil oleh *framework* untuk dijadikan faktor penentu keputusan dilakukannya metode *offloading* pada perangkat lunakdiantaranya terdiri dari kecepatan *bandwidth* internet, kondisi level baterai pada perangkat bergerak sebagai inisialisasi awal serta waktu eksekusi proses *image recognition* baik eksekusi secara lokal maupun *offloading* untuk proses *image recognition* selanjutnya.

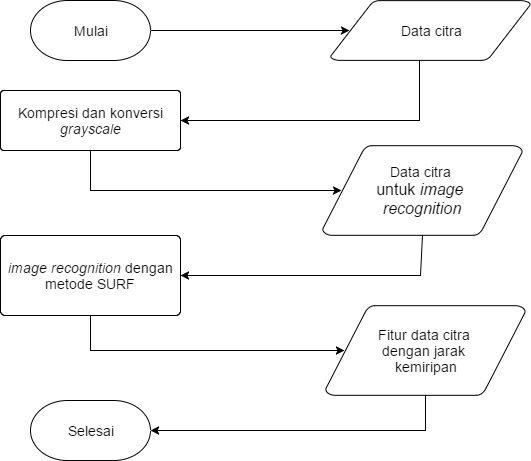
Data – data yang dijadikan faktor penentu keputusan dilakukannya metode *offloading* pada *framework* adalah kecepatan *bandwidth* internet, kondisi level baterai perangkat bergerak dan waktu eksekusi proses *image recognition*.

*Framework* melakukan pengujian terhadap data - data yang dijadikan sebagai faktor penentu seperti yang telah disebutkan di atas, pengujian dilakukan dengan menggunakan pembobotan nilai terhadap data – data tersebut. Hasil pengujian berupa keputusan pemrosesan *image recognition* pada perangkat bergerak *client* yang optimal baik itu dilakukan secara lokal maupun *offloading*.

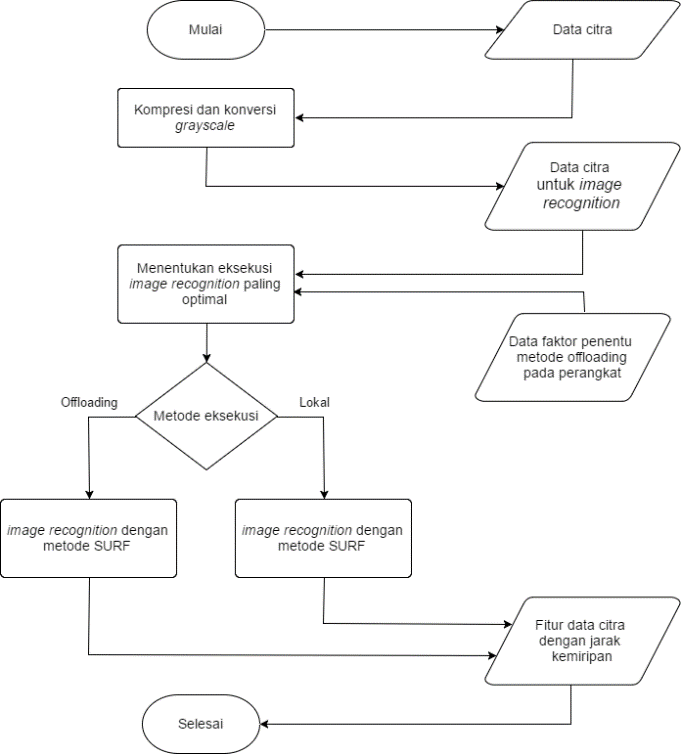
Pada pemrosesan secara lokal, datatest citra daun yang telah diambil dilakukan *image recognition* secara bertahap mulai dari pengubahan datatest citra daun menjadi bentuk tipe data matriks, pengubahan dataset masing – masing citra daun pada perangkat bergerak *client* menjadi bentuk tipe data matriks, pengambilan *descriptor* menggunakan metode SURF dari masing – masing tipe data matriks datatest maupun dataset dan menghitung jarak kedekatan dan kemiripan dari masing – masing *descriptor* tersebut.

Berbeda halnya dengan pemrosesan secara *offloading*, pada perangkat bergerak *client* dilakukan pengubahan datatest citra daun menjadi bentuk tipe data matriks terlebih dahulu sama seperti pemrosesan secara lokal. Akan tetapi, data matriks tersebut kemudian dikonversi menjadi data *string*, hal ini ditujukan untuk pembuatan modul pengiriman data berbentuk *object serialization* Java yang dikirimkan ke *server*. Konversi dari tipe data matriks menjadi data *string* dilakukan menggunakan *library* GSON. Proses modul pengiriman data pada *server* di awali dengan pembuatan *Agent* JADE pada perangkat bergerak *client* dan *server*. Selanjutnya, perangkat bergerak *client* akan melakukan permintaan dengan mengirimkan pesan *Agent* JADE menuju *server*. *Agent* JADE bertugas untuk mengirimkan dan menerima pesan antara *client* dan *server*. Secara bertahap, *Agent* JADE disisipkan pesan berupa modul pengiriman data yang telah dibuat dan selanjutnya akan dikirimkan pada *server*. *Agent* JADEpada *server* secara responsif akan menerima dan membaca pesan jika ada pesan yang dikirimkan oleh *Agent* JADE *client*. Pesan yang diterima *Agent* JADE *server* yaitu modul pengiriman data yang berbentuk *object serialization* Java. Data *string* dari matriks datatest didalam modul pengiriman data tersebutakan dikonversikan kembali menjadi tipe data matriks menggunakan GSON pada *server*. Data matriks datatest tersebut akan dilakukan pengambilan *descriptor* menggunakan metode SURF untuk dibandingkan dengan *descriptor* dari masing – masing dataset pada *server* yang selanjutnya akan dihitung jarak kedekatan dan kemiripan dari masing – masing *descriptor* tersebut. Data jarak kedekatan antara matriks datatest dan dataset tersebut akan dikirimkan kembali menggunakan modul pengiriman data yang sama yaitu berbentuk *object serialization* Java ke perangkat bergerak *client* yang melakukan permintaandan perangkat bergerak *client* tersebut akan mengambil data yang diperlukan dari modul pengiriman tersebut.

Data hasil dari proses *image recognition* antara datatest dan dataset baik diperoleh dari pemrosesan lokal maupun *offloading* selanjutnya akan diseleksi menurut nilai jarak kedekatannya sesuai dengan kriteria yang ditentukan, hal ini dilakukan untuk mengetahui citra daun dataset yang paling mirip dengan citra daun datatest. Keterangan citra daun dataset yang paling mirip dengan citra daun datatest selanjutnya akan ditampilkan pada tampilan perangkat lunak perangkat bergerak *client*. Pada Gambar 3.4 dijelaskan secara garis besar kolaborasi antara proses *image recognition* dengan *offloading* *framework*.



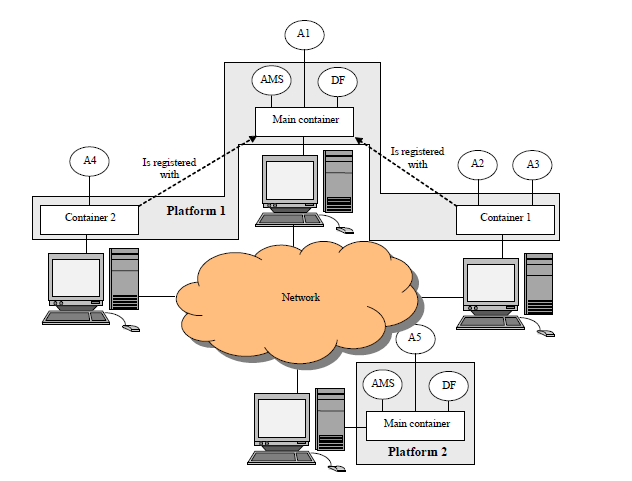
Gambar 3.3 Proses *image recognition* menggunakan metode SURF secara garis besar



Gambar 3.4 Proses *image recognition* menggunakan metode SURF dengan *offloading* *framework*

## JADE *Middleware*

JADEmerupakan *open source multi agent* *middleware* dasar yang diimplementasikan dalam Bahasa Java. JADE dapat bertindak sebagai *middleware* yang memfasilitasi pembuatan sistem *multi* *agent*. Fitur – fitur yang disediakan JADE diantaranya sebuah *runtime environment* dimana JADE *agent* dapat “hidup” dan diaktifkan dengan memberikan alamat *host* sebelum *agent* tersebut dapat dieksekusi, seperangkat kelas *library* yang dibutuhkan dan digunakan *programmer* atau *developer* untuk pengembangan dan pengaturan *agent* yang telah dibuat, serta seperangkat alat grafis yang memungkinkan pengaturan dan pemantauan aktifitas *agent* yang sedang berjalan.

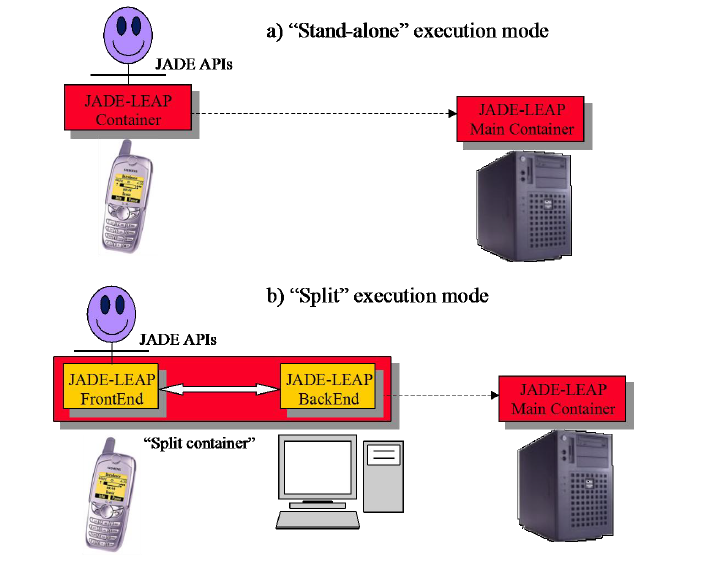
Setiap *runtime environment* JADE yang berjalan disebut *Container* yang dapat berisi beberapa *agent*. Kumpulan dari *Container* yang aktif disebut *Platform*. Sebuah *Main Container* harus selalu diaktifkan terlebih dahulu dan semua *Container* lain yang terdapat pada sebuah *Platform* harus terdaftar dan dikenali oleh *Main Container* sebelum memulai pengaktifan. Dapat disimpulkan bahwa *Container* pertama yang dibuat oleh *Platform* haruslah sebuah *Main Container* sementara *Container* lain yang bersifat “normal” atau bukan *Main Container* harus “di beri tahu” dimana untuk mencari (*host* dan *port*) *Main Container* dari *Container* tersebut. Jika *Main Container* sebuah *Platform* di aktifkan pada tempat tertentu dalam suatu jaringan, *Platform* lain yang berada dalam satu jaringan dan memiliki *Container* baru bersifat “normal” dapat dimungkinkan untuk dikenali oleh *Main Container* pada *Platform* pertama. Untuk lebih jelasnya dapat dilihat pada Gambar 3.5 di bawah ini.

Gambar 3.5 *Container* dan Platform pada JADE *middleware* [12]

Selain kemampuan untuk mengenal *agent* dari *container* lain, *main container* memiliki perbedaan dengan *container* normal sebab memiliki dua *agent* khusus yang otomatis muncul ketika *main container* dibuat, yaitu AMS (*Agent Management System*) dan DF (*Directory Facilitator*) seperti pada gambar.

AMS menyediakan layanan penamaan *agent* dimana setiap *agent* di dalam *platform* harus memiliki nama yang unik serta mewakili otoritas di dalam *platform* dalam melakukan pembuatan dan penghapusan *agent* pada *remote* *container*. Pada tugas akhir ini, penulis mengikuti pengaturan dasar dan tidak melakukan perubahan pada AMS.

DF menyediakan layanan *Yellow Pages* yang artinya *agent* dapat mencari *agent* lain yang menyediakan layanan yang ia butuhkan untuk mencapai tujuannya di dalam satu *platform*. Pada tugas akhir ini, penulis tidak menggunakan DF sebab komunikasi yang dibutuhkan antar *agent* hanya berlangsung dua arah pada setiap *client* dan *server*.

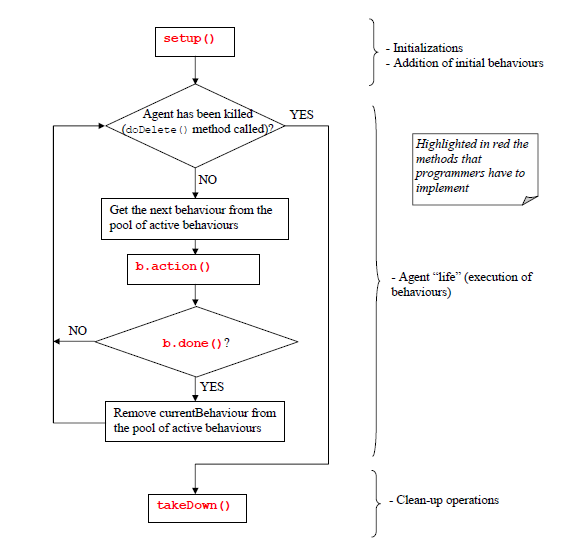
JADE *runtime environment* dapat di eksekusi pada dua mode yang berbeda. Dikatakan melakukan secara mode eksekusi normal *Stand – alone* jika *container* yang lengkap (*main container* dan *container* normal lainnya) di eksekusi pada perangkat atau *host* dimana JADE *runtime* diaktifkan. Dikatakan melakukan secara mode eksekusi *Split* jika *container* di pisahkan antara *Front – End* (proses yang berjalan pada perangkat / *host* dimana JADE *runtime* diaktifkan) dan *Back – End* (proses yang diaktifkan pada *remote server*). Pada mode eksekusi *Split,* perangkat *Front – End* dan *Back – End* dihubungkan dengan koneksi internet dan JADE *container* harus diaktifkan terlebih dahulu pada *host* sesuai dengan dimana *Back – End* telah dibuat. *Container* baru yang dibuat antara Front – End dan Back – End disebut *mediator*. Untuk lebih jelasnya dapat dilihat pada Gambar 3.6 di bawah ini.

Gambar 3.6 Mode eksekusi pada *runtime* JADE [13]

Dalam pengerjaan pada tugas akhir ini, pembuatan JADE *Main* *container* dilakukan pada satu *platform*, yaitu pada perangkat PC (*server*). JADE *runtime environment* dilakukan pada mode eksekusi *Split* dimana perangkat bergerak (*client*) bertindak sebagai *Front – End* dan PC (*server*) bertindak sebagai *Back – End*, sehingga perangkat *client* membuat JADE *container* dengan menggunakan *Main container* pada *server*. Proses pada JADEdimulai dengan pengaktifan JADE *runtime* dari PC *server*, dilanjutkan pada Android *Activity* pada perangkat bergerakyang dihubungkan dengan *MicroRuntimeService*. Hasilnya, objek JADE *MicroRuntimeServiceBinder* diaktifkan sedemikian rupa sehingga memungkinkan untuk melakukan semua operasi manajemen JADE, salah satunya yaitu membuat *container* dengan penerapan mode eksekusi *Split*. Hal yang harus diketahui dalam pembuatan *Split container* adalah alamat *host* dan *port* letak *main container* yangsedang berjalanpada *server*. Setelah JADE *runtime* terhubung dan berjalan, hal ini memungkinkan *client* dan *server* untuk berkomunikasi serta saling mengirim dan menerima pesan.

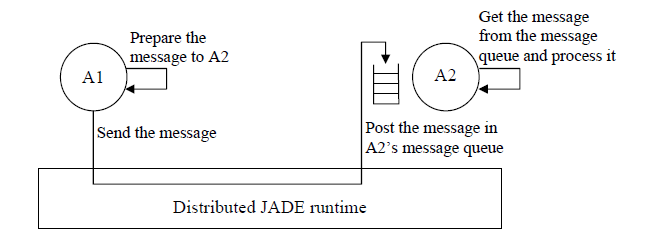
Langkah pertama yang dilakukan setelah JADE *runtime* yang terhubung adalah melakukan pengaturan pada *agent* yang telah dibuat yaitu menginisialisasi identitas *agent* lain yang menjadi tujuan pengiriman pesan seperti nama *agent*, nama *platform* dan alamat *host* menggunakan *AID*. Begitu pun sebaliknya pada JADE agent *server*, pengaturan *AID* dilakukan dengan menggunakan informasi identitas dari *agent* yang melakukan pengiriman sebelumnya.

Langkah kedua, setiap operasi yang akan dikerjakan oleh JADE *agent* dapat dilakukan dengan menggunakan *behaviour*. *Behaviour* dapat ditambahkan secara fleksibel baik ketika pembuatan *agent* (metode *setup*)maupun didalam *behaviour* yang lain. Setiap *behaviour* terdapat metode *action* yang mendefinisikan operasi yang dijalankan ketika *behaviour* dieksekusi dan metode *done* yang menentukan akhir pengeksekusian suatu *behaviour* serta melakukan penghapusan *agent* (metode *dodelete*) dari kumpulan *behaviour* lain dari suatu *agent*. *Agent* dapat mengeksekusi *behaviour* secara bersamaan serta eksekusi akan dimulai saat metode *action* dipanggil dan berjalan sampai mendapatkan hasil dari operasi pada *behaviour*. Untuk lebih jelasnya dapat dilihat pada Gambar 3.7 di bawah ini.

Gambar 3.7 Langkah eksekusi *agent* menggunakan *thread* [12]

Terdapat beberapa macam tipe *behaviour* yang disediakan oleh JADE *agent*. Berdasarkan eksekusi operasi, *behaviour* dibagi menjadi 3 jenis, diantaranya *one – shot* *behaviour*, *cyclic behaviour*, dan *generic behaviour*. *One – shot behaviour* mengeksekusi metode *action* hanya sekali dengan otomatis memberikan nilai *true* pada metode done. *Cyclic behaviour* mengeksekusi metode *action* yang berisi operasi yang sama secara berulang tanpa akhir dengan otomatis memberikan nilai *false* pada metode done. *Generic behaviour* menyisipkan status dalam mengeksekusi metode *action* yang berisi operasi yang berbeda, *behaviour* ini akan menyelesaikan operasi tertentu pada metode *action* ketika syarat kondisi status yang diberikan sesuai. Berdasarkan penjadwalan operasi, *behaviour* dibagi menjadi 2 jenis diantaranya *Waker* *behaviour* dan *Ticker* *behaviour*. *Waker behaviour* mengeksekusi operasi dengan waktu tunggu yang ditentukan. *Ticker behaviour* mengeksekusi operasi secara periodik dengan waktu yang ditentukan. Pada tugas akhir ini, tipe *behaviour* yang digunakan adalah *generic behaviour* pada perangkat *client* dan *ticker behaviour* pada perangkat *server* disebabkan terdapat status tertentu yang harus terpenuhi untuk melakukan operasi dalam metode *action*, misalnya pada saat *agent* *client* melakukan operasi penerimaan pesan dari *agent server* harus dilakukan setelah *agent client* melakukan operasi pengiriman pesan ke *agent server*. Selain itu, *server* harus melakukan pengecekan secara periodik terhadap pesan yang telah dikirimkan oleh perangkat *client* untuk dieksekusi.

Pada JADE *agent client*, pengiriman dan penerimaan pesan dilakukan dengan menggunakan kelas *ACLMessage*. Paradigma komunikasi yang diadopsi adalah pengiriman pesan secara *asynchronous*. Setiap *agent* memiliki *queue* untuk pesan yang digunakan jika JADE *runtime* menampilkan pesan yang dikirimkan oleh *agent* lain. Setiap kali ada pesan yang dikirim didalam *queue*, *agent* penerima akan diberitahu dan mengambil pesan tersebut. Seperti dapat dilihat pada Gambar 3.8 di bawah ini



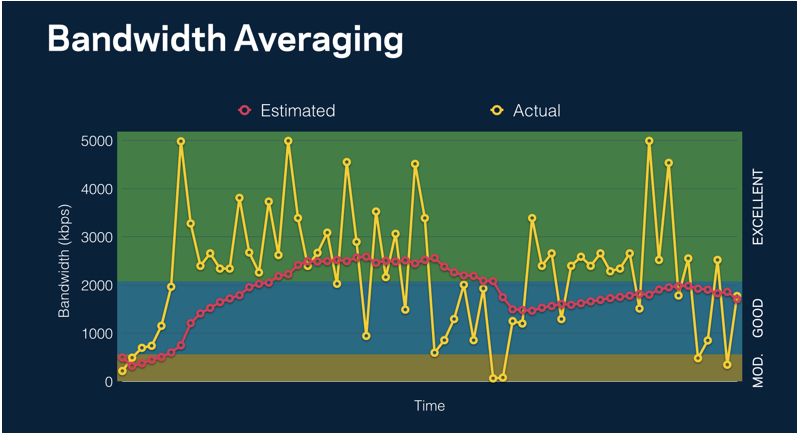
Gambar 3.8 Paradigma pengiriman pesan JADE secara *asynchronous* [12]

Pengiriman pesan ke *agent* lain dilakukan dengan mengisi *field* pada objek *ACLMessage* dan memanggil metode *send* dari kelas *agent*. Pengisian *field* pada *ACLMessage* untuk *container* lain yang berisi informasi *agent* untuk tujuan pengiriman pesan antara lain pengisian nama *agent*, nama *platform* dan alamat *host* menggunakan *AID*. Begitu pun sebaliknya pada JADE agent *server*, pesan yang diterima akan diproses menggunakan metode *receive* kelas *agent* dan akan dibalas dengan pesan yang berisi hasil pemrosesan menggunakan *ACLMessage* dengan pengisian *field* sama seperti sebelumnya. Pengaturan *AID* dilakukan dengan menggunakan informasi dari *container* yang melakukan pengiriman sebelumnya.

## Faktor Penentu Metode *Offloading*

### Kualitas Koneksi Internet

Data yang pertama adalah kecepatan *bandwidth* internet yang terbagi menjadi 4 kelas urut dari kualitas koneksi yang baik ke buruk yaitu *Excellent*, *Good*, *Moderate*, dan *Poor* serta jika koneksi internet tidak tersedia pada perangkat *client* maka dimasukkan dalam kelas *Unknown*. Data ini diperoleh dari fungsi eksekusi mengunduh data gambar kualitas rendah yang terpasang pada perangkat lunak sebagai penguji kualitas koneksi internet yang dimiliki oleh perangkat bergerak saat itu. Pengujian dilakukan dengan mengecek *buffer* data aktual yang terjadi saat mengunduh data gambar dengan estimasi *buffer* data berdasarkan besarnya ukuran data yang di unduh. Untuk lebih jelasnya dapat dilihat pada Gambar 3.9 Informasi estimasi *buffer* data diperoleh dari kode sumber *connection class facebook network*.



Gambar 3.9 Contoh pengelompokan kualitas koneksi berdasarkan estimasi *buffer* data dan *buffer* data actual [14]

### Kondisi Level Baterai Perangkat Bergerak

Data yang kedua adalah kondisi level baterai perangkat bergerak yang berupa nilai *range* mulai dari 0 hingga 100. Nilai ini diperoleh dengan mengambil nilai level baterai perangkat bergerak secara *real time* menggunakan kode sumber IDE Android Studio.

### Waktu Eksekusi Proses *Image Recognition*

Data yang ketiga adalah waktu eksekusi proses *image recognition* yang diperoleh dari menghitung waktu dimulainya proses *image recognition* menggunakan metode SURF hingga diperolehnya data jarak kedekatan citra daun datatest dengan masing – masing citra daun dataset. Perhitungan waktu eksekusi dilakukan baik pada eksekusi secara lokal maupun eksekusi secara *offloading*.

## *Offloading Framework*

*Offloading framework* merupakan salah satu metode untuk penghematan daya pada perangkat bergerak dengan cara melakukan *offload* beban kerja dari perangkat bergerak ke *server*. Metode ini sangat mendukung untuk diimplementasikan sebab konektivitas internet yang bertindak sebagai *virtual bus* dapat diperoleh atau didapatkan dimana saja dan kapan saja saat ini. Studi kasus pada tugas akhir ini, *server* tidak dibatasi oleh sumber daya baterai dan memiliki *processor* dengan performa jauh lebih cepat dibandingkan dengan perangkat bergerak *client* sehingga apabila melakukan *offloading* beban kerja ke server, diharapkan eksekusi operasi terhadap beban kerja lebih cepat dan meminimalisir waktu eksekusi.

Tidak semua beban kerja pada perangkat bergerak *client* dapat di *offloading*, misalnya pada layanan *background* data koordinat lokasi menggunakan GPS yang sensitif berubah secara berkelanjutan sehingga pengiriman data secara *offload* dilakukan secara terus menerus. Hal ini menyebabkan penggunaan sumber daya baterai lebih boros dan tidak sesuai dengan konsep dasar metode *offloading*. Pada tugas akhir ini, beban kerja yang akan di *offload* ke *server* sudah ditentukan yaitu *image recognition* antara datatest dan dataset citra daun menggunakan metode SURF.

Dalam memulai eksekusi *offload* beban kerja pada *server*, ada beberapa hal yang harus dilakukan sebelumnya agar hasil dari eksekusi *offload* beban kerja ke *server* maksimal dan sesuai yang diharapkan. Identifikasi beban kerja, kondisi perangkat bergerak *client* dan pemantauan kualitas jaringan internet sangat penting dalam *framework* ini. Identifikasi beban kerja seperti perhitungan waktu eksekusi baik dilakukan secara lokal maupun *offloading*, kondisi perangkat bergerak *client* seperti kondisi level baterai, serta pemantauan kualitas jaringan internet yang dilakukan secara dinamis dan berkala setiap adanya beban kerja yang akan di eksekusi diperlukan dalam *framework*. *Framework* yang melakukan eksekusi beban kerja secara dinamis dan berkala diharapkan dapat memberikan keputusan metode eksekusi yang tepat dan optimal baik secara lokal maupun *offloading* terhadap hal – hal yang dapat mempengaruhi pengambilan keputusan metode eksekusi yang contohnya telah disebutkan sebelumnya. Selanjutnya, hal – hal yang dapat mempengaruhi pengambilan keputusan metode eksekusi kita sebut sebagai faktor – faktor penentu metode *offloading* serta fungsi dalam *framework* yang digunakan untuk menentukan keputusan metode eksekusi kita sebut sebagai *decision maker*.

*Decision maker* adalah fungsi yang dipanggil setiap beban kerja yang telah ditentukan pada perangkat bergerak akan dieksekusi. Data faktor – faktor penentu metode *offloading* akan digunakan sebagai catatan histori performa oleh *decision maker* untuk memperkirakan dan membandingkan biaya dari kedua metode eksekusi tersebut sebelum memutuskan metode eksekusi yang akan dipilih dan memberikan keuntungan yang lebih besar dalam penghematan sumber daya dan waktu eksekusi. Pada perancangan tugas akhir ini, *decision maker* melakukan pengecekan data terhadap kualitas koneksi internet perangkat *client*, kondisi level baterai perangkat *client* dan rata – rata waktu eksekusi masing – masing metode pengeksekusian beban kerja *image recognition*. Untuk eksekusi pertama, akan dilakukan eksekusi secara lokal untuk mengetahui waktu eksekusi secara lokal yang digunakan sebagai pembanding dengan metode offloading selanjutnya. Jika dari data – data tersebut memenuhi untuk dilakukannya metode *offloading*, maka dilakukanlah eksekusi secara *offloading* serta hasil waktu eksekusinya akan dibandingkan dengan metode secara lokal. Dalam menentukan metode eksekusi selanjutnya setelah pengeksekusian beban kerja menggunakan waktu eksekusi digunakan Persamaan 3.1 dan Persamaan 3.2 [15].

(3.1)

(3.2)

Penjelasan pada persamaan di atas, *I* mewakili rangkaian beban kerja yang dilakukan dalam perangkat lunak. Untuk setiap beban kerja *i* *I* yang dapat di eksekusi secara *offloading*, adalah waktu eksekusi dari *i* pada *client*. adalah waktu eksekusi dari *i* pada *server*. adalah waktu pengiriman dari *i*. adalah variabel indikator: = 0 jika *i* dieksekusi secara lokal, = 1 jika *i* dieksekusi secara *remote* pada *server* atau *offloading*, *l* didasarkan pada kebutuhan waktu yang digunakan oleh pengembang.

Untuk studi kasus menentukan metode eksekusi, akan dilakukan eksekusi secara *offloading* jika selisih waktu eksekusi pada perangkat *server* terhadap waktu eksekusi pada perangkat *client* kurang dari sama dengan waktu yang telah ditentukan.

Penerapan menentukan keputusan sebagian besar didasarkan pada Persamaan 3.2 dengan beberapa modifikasi seperti penggunaan waktu rata – rata eksekusi metode *offloading* ( + ) dan lokal () serta *l* diinisialisasi dengan nilai 8 detik. Jika memenuhi lagi untuk metode *offloading* maka dilakukan metode *offloading* begitu seterusnya sampai semua beban kerja pada proses *image recognition* telah selesai dieksekusi.

## Metode SURF

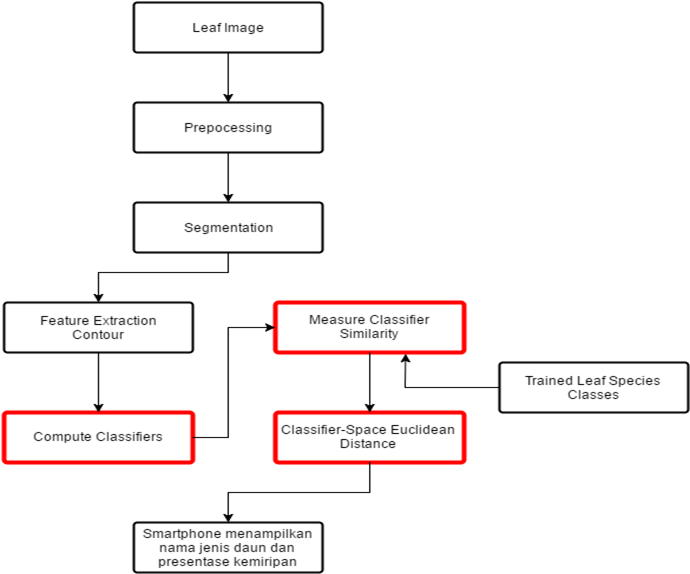
SURF singkatan dari *Speed Up Robust Features* merupakan salah satu metode yang digunakan untuk *feature extraction* pada proses *image recognition*. Pada *image recognition* sendiri terdapat empat proses dalam pengolahan citra diantaranya *preprocessing*, *segmentation*, *feature extraction* dan *classification*.

Tahap pertama, *preprocessing* adalah metode untuk memaksimalkan citra yang didapat untuk proses *image* *recognition*. Pada tugas akhir ini, tahap *preprocessing* dilakukan dengan mengkompresi citra (datatest) yang didapat oleh kamera perangkat bergerak (*client*) agar sesuai dengan ukuran dataset dan konversi citra ke pewarnaan *grayscale*.

Tahap kedua, *Segmentation* adalah metode untuk identifikasi *image* *recognition* dari gambar dengan cara mengenali ciri – ciri unik gambar sehingga dapat dibedakan antara yang satu dengan yang lainnya. Pada tugas akhir ini, tahap *segmentation* dilakukan dengan menemukan titik – titik penting fitur citra dengan menggunakan *feature* *detector* metode SURF yang disediakan *library* OpenCV.

Tahap ketiga, *feature extraction* adalah metode untuk menghitung ciri – ciri citra yang diperoleh dari tahap *segmentation* ke dalam bentuk tertentu. Ciri – ciri citra yang di dapat umumnya disebut *descriptor*. *Descriptor* memiliki beberapa bentuk yang digunakan sebagai ciri – ciri pada citra seperti garis luar/*outline* objek pada citra, garis kerangka pada citra, serta umumnya titik – titik ujung maupun sudut – sudut pada citra. Pada tugas akhir ini, tahap *feature extraction* dilakukan dengan menghitung *descriptor* berupa titik – titik yang digunakan sebagai fitur baik pada citra datatest maupun citra dataset. *Descriptor extractor* metode SURF yang disediakan *library* OpenCV digunakan sebagai penghitung nilai *descriptor* fitur kedua citra tersebut .

Tahap akhir, *classification* adalah metode untuk mengklasifikasi gambar dari hasil *feature extraction* berdasarkan kelas yang telah didefinisikan. Kelas ini didapatkan dari pengklasifikasian data *training* / dataset yang sudah dilakukan terlebih dahulu. Pada tugas akhir ini, tahap *classification* dilakukan dengan menghitung jarak kedekatan dan kemiripan nilai *descriptor extractor* antara datatest dan dataset dengan *descriptor matcher*. *Descriptor matcher* dilakukan denganmetode FLANNBASED yang disediakan metode OpenCV dengan ketentuan semakin dekat dan mirip nilai kedua *descriptor extractor* maka nilai *descriptor matcher* semakin mendekati nol. Pada tugas akhir ini, nilai jarak kedekatan *descriptor* yang memenuhi kriteria bernilai kurang dari 0.015. Urutan langkah pada proses *image recognition* dapat terlihat pada Gambar 3.10.



Gambar 3.10 Proses *image recognition* pada citra daun

## Java *Class Object Serialization* dari *Client* ke *Server*

Java *Class Object Serialization Client* ke *Server* adalah sebuah kelas java yang terdiri data – data yang akan dikirimkan dari *client* ke *server* dan di serialiasasi menjadi *object* java. Serialisasi dari data java menjadi *object* Java dapat dilakukan menggunakan *library* Commons – Lang yang berfungsi menambah fungsionalitas pada inti Java. Pada tugas akhir ini, serialisasi data menjadi *object* java digunakan sebagai modul pengiriman data dari *client* ke *server.* Data – data tersebut terdiri dari data *string* *descriptor* citra daun datatest yang dikonversi dari tipe data matriks OpenCV menggunakan GSON, data *string* nama file citra daun dataset yang di akan dilakukan *image recogniton*, data *string* waktu *agent* JADE memulai modul pengiriman data dari *client* ke *server* yang terdiri dari keterangan jam, menit, detik dan milidetik. Data – data *string* yang dikirimkan dalam modul data akan diterima oleh *server* dan dikonversikan kembali menjadi tipe data dalam java seperti semula. Misalnya, data *string* *descriptor* citra daun datatest akan dikonversikan kembali menjadi tipe data matriks OpenCV pada *server* untuk pemrosesan *image recognition* citra daun.

## Java *Class Object Serialization* dari *Server* ke *Client*

Java *Class Object Serialization Server* ke *Client* adalah sebuah kelas java yang terdiri dari data – data yang akan dikirimkan dari *server* ke *client* dan di serialisasi menjadi *object* java seperti yang telah dijelaskan sebelumnya. Perbedaan dengan kelas serialisasi objek sebelumnya yaitu data – data pada kelas serialisasi objek dari *server* ke *client* berisi data – data hasil *image recognition* citra daun menggunakan metode SURF yang akan ditampilkan pada perangkat bergerak (*client*). Data – data tersebut terdiri dari data *string* nama file citra daun dataset yang telah dilakukan *image recognition*, data *integer* jumlah titik – titik yang jarak kemiripannya diterima / valid sesuai dengan batas jarak minimal yang telah ditentukan, data *double* jarak kedekatan minimal yang dapat diperoleh antara citra daun datatest dan dataset, data *double* jarak kedekatan maksimal yang dapat diperoleh antara citra daun datatest dan dataset, data *double* waktu yang dibutuhkan *agent* dalam pengiriman modul data, data *string* waktu agent JADE memulai modul pengiriman data dari *server* ke *client* yang terdiri dari keterangan jam, menit, detik dan milidetik dan data *double* waktu yang dibutuhkan *agent* dalam penerimaan modul data dari server.

# BAB IV IMPLEMENTASI

Bab ini berisi penjelasan mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa *Pseudocode* untuk membangun program.

## Lingkungan Implementasi

Implementasi penerapan *offloading computation mobile framework* pada sistem perangkat lunak *image recognition* menggunakan metode SURF menggunakan spesifikasi perangkat keras dan perangkat lunak seperti yang ditunjukkan pada Tabel 4.1.

Tabel 4.1 Lingkungan Implementasi Perangkat Lunak

|  |  |  |  |
| --- | --- | --- | --- |
| *Privilege* | Perangkat | Jenis Perangkat | Spesifikasi |
| *Client* | Perangkat Keras | Prosesor | Intel(R) Atom(TM) CPU Z2580 2,00 GHz |
| Memori | 2 GB DDR3 |
| Perangkat Lunak | Sistem Operasi | Android 4.4.2 KitKat |
| *Server* | Perangkat Keras | Prosesor | Intel(R) Core(TM) i7-2.5 GHz |
| Memori | 8 GB 800 MHz DDR3 |
| Perangkat Lunak | Sistem Operasi | Windows 10 dan Ubuntu 16.04 |
| Perangkat Pengembang | Android Studio IDE dan Netbeans IDE 8.2 |

## Implementasi

Pada sub bab implementasi ini menjelaskan mengenai pembangunan perangkat lunak secara detail dan menampilkan *Pseudocode* yang digunakan mulai tahap *preprocessing* hingga *feature extraction* antara datatest dan dataset yang digunakan dalam penerapan *image recognition* dengan metode SURF, selain itu, terdapat mekanisme eksekusi menggunakan *offloading framework* untuk penghematan daya dan peningkatan kinerja perangakat *client*. Pada tugas akhir ini data yang digunakan, seperti yang telah dijelaskan pada bab sebelumnya, yaitu terdiri dari dua jenis data masukan. Data masukan jenis pertama diperoleh dengan menggunakan kamera yang terpasang di perangkat Android dan dilakukan pemfotoan saat menjalankan perangkat lunak Android. Data masukan jenis pertama yang belum diketahui jenisnya tersebut digunakan sebagai datatest untuk uji coba prediksi pada *image recognition* menggunakan metode SURF untuk ekstraksi fitur. Sedangkan data masukan jenis kedua adalah sebuah dataset terdiri dari 40 buah citra daun yang dapat dibagi 10 citra daun berdasarkan jenis daunnya sehingga setiap jenis daun memiliki 4 citra daun. Data masukan jenis kedua yang berjumlah 40 tersebut digunakan sebagai data masukan untuk uji coba pencocokan dengan datatest pada *image recognition* menggunakan metode SURF untuk ekstraksi fitur.

### JADE *Middleware*

#### Client

Hal pertama yang dilakukan pada JADE *middleware* adalah mengaktifkan JADE *runtime* dari Android *Activity* untuk menghubungkannya dengan *MicroRuntimeService* melalui *Pseudocode* 4.1. Hal ini bertujuan untuk memungkinkan perangkat bergerak *client* menjalankan operasi manajemen JADE. Terdapat variabel – variabel yang harus diinisialisasi untuk terhubung dengan *MicroRuntimeService*. Variabel – variabel tersebut digunakan sebagai parameter dalam menghubungkan perangkat lunak dengan *MicroRuntimeService*, variabel tersebut diantaranya *host* dan *port* yang diletakkan dalam tipe data properties yang digunakan untuk mereferensi alamat *platform* serta agentControlleryang digunakan untuk pembuatan *container* baru. Sebelum menghubungkan *MicroRuntimeService*, dilakukan pengecekan dahulu terhadap *MicroRuntimeService* apakah sudah terhubung atau belum. Jika belum terhubung, maka *MicroRuntimeService* dihubungkan dan dilanjutkan pada tahap selanjutnya.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **function** bindService(String host, String port,  RuntimeCallback<AgentController>  agentStartupCallback)  Properties pp := new Properties()  pp.setProperty(Profile.MAIN\_HOST, host)  pp.setProperty(Profile.MAIN\_PORT, port)  pp.setProperty(Profile.JVM,  Profile.ANDROID)  **if** (microRuntimeServiceBinder == NIL)  mServiceConnection := new  ServiceConnection()  @Override  **function**  onServiceConnected(ComponentName  componentName, IBinder service)  microRuntimeServiceBinder :=  (MicroRuntimeServiceBinder)  service  Log.i(TAG,"###Gateway  successfully bound to  RuntimeService")  startMainContainer(pp,  agentStartupCallback)  @Override  **function**  onServiceDisconnected(ComponentName  componentName)  Log.i(TAG,"###Gateway unbound  from RuntimeService")  Log.i(TAG,"###Binding Gateway to  RuntimeService...")  bindService(new  Intent(getApplicationContext(),  MicroRuntimeService.class),  mServiceConnection,  Context.BIND\_AUTO\_CREATE)  else  startMainContainer(pp,  agentStartupCallback) |

*Pseudocode* 4.1 Kode program menghubungkan JADE *runtime* dengan *MicroRuntimeService*

Tahap selanjutnya adalah membuat *container* dari *MicroRuntimeService* melalui *Pseudocode* 4.2. Nilai parameter yang dibutuhkan dalam membuat *container* adalah nilai – nilai yangdisisipkan dalam propertiesdan agentControllerseperti yang dijelaskan pada fungsi bindService(dapat dilihat pada *Pseudocode* 4.1). Nilai *host* pada properties berisi alamat *http* dari *server*. Sehingga *client* tidak membuat *main container* sendiri pada perangkat bergerak yang dimilikinya tetapi pada perangkat *server*. Penerapan metode container seperti ini disebut *Split container*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **function** startMainContainer(Properties profile,  RuntimeCallback<AgentController>  agentStartupCallback)  microRuntimeServiceBinder.  startAgentContainer(profile,new  RuntimeCallback<Void>()  @Override  function onSuccess(Void  thisIsNull)  Log.i(TAG, "###Agent-  Container  created...")  bindservice := true  @Override  function onFailure(Throwable  throwable)  Log.i(TAG, "###Failed to  create  Main Container") |

*Pseudocode* 4.2 Kode Program membuat *Container*

*Agent* dapat dibuat setelah *container* pada *platform* *server* telah berjalan. Pembuatan *agent* dapat dilakukan melalui *Pseudocode* 4.3. Terdapat nilai – nilai parameter yang dibutuhkan dalam pembuatan *agent* seperti nama *agent*, nama kelas java yang mengimplementasikan fungsional *agent*, dan *agentController*. Selain itu, terdapat data – data yang diinisialisasi ke dalam beberapa Object args yang akan disisipkan dalam pembuatan *agent* yaitu *context* dari kelas java tempat pembuatan *agent*, *descriptor* dari datatest citra daun yang telah di konversi menjadi *string*, nama file dataset citra daun yang akan dilakukan *image recognition*, dan waktu dimulainya eksekusi pembuatan *agent client* yang disamakan dengan waktu pengiriman pesan ke *agent server*. Selain itu, disediakan *error handling* seperti pengecekan keberadaan *MicroRuntimeService* pada baris 2, pengecekan keberhasilan eksekusi pembuatan *agent* baru pada baris 16.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | **function** createAgent(String name, String  className,RuntimeCallback<AgentController>  agentStartupCallback)  **if** (microRuntimeServiceBinder != NIL)  TimeZone local :=  TimeZone.getTimeZone("Asia/Jakarta")  Calendar now :=  Calendar.getInstance(local)  SimpleDateFormat formatter := new  SimpleDateFormat("HH:mm:ss.SSS")  formatter.setTimeZone(local)  String dateString :=  formatter.format(now.getTime())  Log.i(TAG, "Time Send Start : " +  dateString)  Object[] args := Object[0..3]  args[0] := SURFExampleActivity.this  args[1] := JSONdescCamera  args[2] := ImageName  args[3] := dateString  microRuntimeServiceBinder.startAgent  (name, className, args, new  RuntimeCallback<Void>()  @Override  **function** onSuccess(Void  thisIsNull)  Log.i(TAG,"###Success to  create agent")  **try**  agentStartupCallback.  onSuccess(MicroRuntime.  getAgent(name))  **catch**(ControllerException  e)  agentStartupCallback.  onFailure(e)  @Override  **function** onFailure(Throwable  throwable)  Log.i(TAG, "###Failed to  created an Agent")    agentStartupCallback.  onFailure(throwable)  )  **else**  Log.e(TAG, "###Can't get Main-  Container to create agent") |

*Pseudocode* 4.3 Kode Program membuat *Agent*

Pada kelas *agent*, terdapat variabel – variabel yang di inisialisasi yang dapat dilihat melalui *Pseudocode* 4.4. Variabel StatusTaskdan agentCreateddigunakan untuk proses pada *offloading framework* (Bab 4.2.2). Variabel desc dan ImageName digunakan untuk proses *image recognition* serta variabel SendAgentStartTime, ReceiveAgentStartTime dan receiveAgentTime digunakan untuk menghitung waktu eksekusi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **class** SendObjectAgent inherits Agent implements SimpleAgentInterface  String TAG := "SendObjectAgent"  StatusTask := false  agentCreated := false  Context context  String ipAddress  String agentName  String desc  String ImageName  String SendAgentStartTime  String ReceiveAgentStartTime  receiveAgentTime |

*Pseudocode* 4.4 Kode program inisialisasi variabel *agent* pada *client*

Pada fungsi setup() pada kelas *agent* digunakan untuk memulai operasi – operasi yang dilakukan oleh *agent* seperti yang telah dijelaskan pada Bab 3.3. Pengaturan setup() dapat dilihat pada *Pseudocode* 4.5. Pertama, kita ambil dahulu nilai parameter dari pembuatan *agent* pada *Pseudocode* 4.3sebelumnya. Nilai parameter kita tampung pada variabel context, desc, ImageName dan SendAgentStartTime. Selanjutnya, kita tambahkan *behaviour* untuk menjalankan operasi. Pada fungsi setup(), untuk mengetahui *host* dan *port* server digunakan SharedPreferences yang akan dijelaskan pada Bab 4.2.2.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **function** setup()  Object[] args := getArguments()  **if** (args != NIL AND args.length > 0)  **if** (args[0] instanceof Context)  context := (Context) args[0]  **if** (args[1] instanceof String)  desc := (String) args[1]  **if** (args[2] instanceof String)  ImageName := (String) args[2]  **if** (args[3] instanceof String)  SendAgentStartTime := (String)  args[3]  addBehaviour(new ReceiveMessage())    registerO2Ainterface  (SimpleAgentInterface.class, this)  Intent broadcast := new Intent()  broadcast.setAction  ("jade.demo.agent.SEND\_MESSAGE")  Log.i(TAG, "###Sending broadcast " +  broadcast.getAction())  context.sendBroadcast(broadcast)  SharedPreferences sharedPreferences :=  context.getSharedPreferences  (Constants.PREFS\_FILE\_NAME,  Context.MODE\_PRIVATE)  ipAddress := sharedPreferences.getString  (Constants.PREFS\_HOST\_ADDRESS,  ipAddress)  agentName := sharedPreferences.getString  (Constants.PREFS\_AGENT\_NAME,  agentName)  **function** onHostChanged(String host)  ipAddress := host  **function** onAgentNameChanged(String name)  agentName := name |

*Pseudocode* 4.5 Kode program metode *Setup* pada *client*

*Behaviour* yang digunakan *agent* pada client adalah *generic behaviour* melalui *Pseudocode* 4.6, untuk penjelasannya bisa dilihat pada Bab 3.3. Pada *behaviour*, terdapat operasi yang dapat di pisah menjadi dua sub operasi. Sub operasi yang pertama yaitu mengirim pesan menuju *server*. Pada sub operasi ini, langkah awal adalah membuat *AID* beserta pengaturan nama dan alamat *agent* pada *server*. Langkah selanjutnya menyisipkan variabel desc, ImageName dan SendAgentStartTime pada kelas serialisasi *ObjectDataMat* yang akan dijelaskan pada Bab 4.2.4. Langkah akhir yaitu mengirim pesan kepada server. Jika sub operasi satu selesai, sub operasi yang kedua langsung dijalankan. Sub operasi yang kedua adalah menerima pesan yang berisi hasil operasi beban kerja dari *server*. Langkah awal adalah menunggu adanya pesan diterima dari *server*. Jika pesan sudah diterima, maka langkah selanjutnya pesan akan ditampung menggunakan kelas serialisasi *data* yang lebih jelasnya akan dijelaskan pada Bab 4.2.5 beserta pencatat waktu penerimaan pesan. Langkah akhir adalah mengirim kembali hasil operasi dari pesan yang telah diterima ke kelas java pembuat *agent* melalui fungsi exportLog.Jika tugas *behaviour* selesai, maka dilanjutkan memanggil fungsi doDelete() untuk menghapus *behaviour* dari *agent*. Jika salah satu *behaviour* dari *agent* dihapus maka fungsi done() otomatis dipanggil.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | **class** ReceiveMessage inherits Behaviour  step := 0  @Override  **function** action()  **switch** (step) :  **case** 0:  **try**  agentCreated := true  ACLMessage message := new  ACLMessage(ACL  Message.INFORM)  AID dummyAid := new AID()  dummyAid.setName  (agentName + "@" +  ipAddress +  ":1099/JADE")  dummyAid.addAddresses  ("http://" + ipAddress  + ":7778/acc")    message.addReceiver(dummyAid)  String convId := "C-" +  myAgent.getLocalName()  ObjectDataMat p := new  ObjectDataMat(desc, ImageName,  SendAgentStartTime)    message.setContentObject(p)  message.setLanguage  ("JavaSerialization")  step := 1  myAgent.send(message)  Log.i(TAG, "###Send  message:" + message.  getContent())  **catch** (IOException e)  e.printStackTrace()  **break**  **case** 1:  agentCreated := true  ACLMessage messageserver :=  myAgent.receive()  **if** (messageserver != NIL)  **if** ("JavaSerialization".equals  (messageserver.getLanguage()))  **try**  data d := (data)  messageserver.  getContentObject()  TimeZone local :=  TimeZone.getTime  Zone("Asia/Jakarta")  Calendar now :=  Calendar.get  Instance(local)  SimpleDateFormat  formatter := new  SimpleDateFormat  ("HH:mm:ss.SSS")    formatter.set  TimeZone(local)    String ReceiveAgent  FinishTime := formatter.  format(now.getTime())  String[] time\_receive2 :=  ReceiveAgentFinishTime.  split(":")  hour\_receive2 :=  Integer.parseInt  (time\_receive2[0])  minute\_receive2 :=  Integer.parseInt  (time\_receive2[1])  seconds\_receive2 :=  Double.parseDouble  (time\_receive2[2])  ReceiveAgentStart  Time := d.getReceive  AgentStartTime()  String[] time\_receive1 :=  ReceiveAgentStartTime.  split(":")  hour\_receive1 :=  Integer.parseInt  (time\_receive1[0])  minute\_receive1 :=  Integer.parseInt  (time\_receive1[1])  seconds\_receive1 :=  Double.parseDouble  (time\_receive1[2])    Log.i("SURFExample  Activity", "Time  Receive Start : "  + ReceiveAgent  StartTime)    Log.i("SURFExample  Activity", "Time  Receive Finish :  " + ReceiveAgent  FinishTime)    hour\_receive :=  hour\_receive2 –  hour\_receive1  minute\_receive :=  minute\_receive2 –  minute\_receive1  seconds\_receive :=  seconds\_receive2 –  seconds\_receive1  receiveAgentTime:= 0  **if** (hour\_receive  != 0)  receiveAgentTime  := receiveAgent  Time + (hour\_  receive \* 3600)  **if** (minute\_receive  != 0)    receiveAgentTime  := receiveAgent  Time + (minute\_  receive \* 60)  **if** (seconds\_receive  != 0)    receiveAgentTime  := receiveAgent  Time + seconds\_  receive    d.setReceiveAgent  Time(receive  AgentTime)  exportLog(d)  **catch** (UnreadableException e)  e.printStackTrace()  **else**  Log.d("result", "Failed")  StatusTask := true  doDelete()  **else**  block()  @Override  **function** done()  **return** false  **function** exportLog(data log)  SURFExampleActivity surfExampleActivity :=  (SURFExampleActivity) context  surfExampleActivity.exportLogConsole(log) |

*Pseudocode* 4.6 Kode program generic *behaviour* pada *client*

#### Server

Pada *server*, kelas *agent* memiliki variabel – variabel yang diinisialisasi melalui *Pseudocode* 4.7. Variabel Mat descriptor, MatOfDMatch matches, Bitmap inputImage, FeatureDetector detector, DescriptorExtractor descriptorExtractor dan DescriptorMatcher descriptorMatcher digunakan untuk *image recognition*. Kelas *data* digunakan menampung hasil operasi dari *image recognition* serta sendAgentTime digunakan untuk perhitungan waktu eksekusi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class TestObject inherits Agent  Mat descriptor  MatOfDMatch matches  Bitmap inputImage  FeatureDetector detector  DescriptorExtractor descriptorExtractor  DescriptorMatcher descriptorMatcher  String path := "/home/putrosw/training/"  data d  sendAgentTime |

*Pseudocode* 4.7 Kode program inisialisasi variabel *agent* pada *server*

Fungsi setup() pada kelas *agent server* digunakan untuk menjalankan *behaviour* yag berhubungan dengan operasi *image recognition*. Pengaturan setup() pada *server* dapat dilihat pada *Pseudocode* 4.8. Pertama, kita panggil dahulu *native library* agar metode yang digunakan dalam *image recognition* menggunakan OpenCV dapat diterapkan. Selanjutnya, kita tambahkan *behaviour* untuk menjalankan operasi *image recognition*.

|  |  |
| --- | --- |
| 1  2  3  4 | **function** setup()  System.loadLibrary  (Core.NATIVE\_LIBRARY\_NAME)  writeln "Hallo! Receivermessage-agent " +  getAID().getName() + " is ready."  addBehaviour(new TickerBehaviour(this, 1000) |

*Pseudocode* 4.8 Kode program metode *Setup* pada *server*

*Behaviour* yang digunakan *agent* pada *server* adalah *ticker behaviour* bias dilihat melalui *Pseudocode* 4.9, untuk penjelasannya bias dilihat pada Bab 3.3. Pada *behaviour*, terdapat operasi yang digunakan untuk melakukan *image recognition*. Operasi pada *behaviour* diawali dengan pemilihan metode fitur *descriptor*, *descriptor extractor*, dan *descriptor matcher*. Pada tugas akhir ini, metode fitur *descriptor* sekaligus *descriptor extractor* adalah SURF, dan *descriptor matcher* yang digunakan adalah metode FLANNBASED. Selanjutnya, *behaviour* akan menunggu sampai adanya pesan masuk dari *agent client*. Jika pesan masuk, data dalam pesan tersebut akan ditampung dalam kelas *ObjectDataMat* disertai dengan pencatatan waktu sebagai penanda jika pengiriman *agent* telah selesai dilakukan. *Descriptor* datatest yang bertipe data *string* diubah menjadi tipe data *Mat* menggunakan fungsi matFromJsonagar dapat diproses dalam *image recognition*, untuk lebih jelasnya bisa dilihat pada Bab 4. Setelah *descriptor* datatest berubah menjadi tipe data *Mat*, maka fungsi surf() dipanggil untuk menjalankan *image recognition*, untuk lebih jelasnya bisa dilihat pada Bab 4.2.3. Setelah *image recognition* selesai dilakukan, data yang dihasilkan akan dikirimkan kembali ke *agent client*. *AID* baru dibuat dengan pengaturan nama dan alamat *agent* disesuaikan dengan *agent client* pengirim pesan. Langkah terakhir adalah mengirimkan kembali pesan yang berisi data hasil *image recognition* ke *agent client* pengirim pesan serta menyisipkan waktu yang dicatat saat *server* mengirimkan pesan ke *agent client*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | **function** onTick()  **try**  Mat rgba := new Mat()  detector := FeatureDetector.create  (FeatureDetector.SURF)  descriptorExtractor := DescriptorExtractor.  create(DescriptorExtractor.SURF)  descriptorMatcher := DescriptorMatcher.  create(DescriptorMatcher.FLANNBASED)  writeln "Trying receive object"  ACLMessage msg := blockingReceive()    **if**("JavaSerialization".equals  (msg.getLanguage()))  ObjectDataMat p := (ObjectDataMat)  msg.getContentObject()  writeln getLocalName() + " read Java  Object " + p.getClass().getName()  TimeZone local :=  TimeZone.getTimeZone("Asia/Jakarta")  Calendar now :=  Calendar.getInstance(local)  SimpleDateFormat formatter := new  SimpleDateFormat("HH:mm:ss.SSS")  formatter.setTimeZone(local)  String SendAgentFinishTime :=  formatter.format(now.getTime())  writeln "Send Agent Finish Time : " +  SendAgentFinishTime  String[] time\_send2 :=  SendAgentFinishTime.split(":")  hour\_send2 :=  Integer.parseInt(time\_send2[0])  minute\_send2 :=  Integer.parseInt(time\_send2[1])  seconds\_send2 :=  Double.parseDouble(time\_send2[2])  String SendAgentStartTime :=  p.getSendAgentStartTime()  String[] time\_send1 :=  SendAgentStartTime.split(":")  hour\_send1 :=  Integer.parseInt(time\_send1[0])  minute\_send1 :=  Integer.parseInt(time\_send1[1])  seconds\_send1 :=  Double.parseDouble(time\_send1[2])  hour\_send := hour\_send2 - hour\_send1  minute\_send := minute\_send2 –  minute\_send1  seconds\_send := seconds\_send2 –  seconds\_send1  sendAgentTime := 0  **if** (hour\_send != 0)  sendAgentTime := sendAgentTime +  (hour\_send \* 3600)  **if** (minute\_send != 0)  sendAgentTime := sendAgentTime +  (minute\_send \* 60)  **if** (seconds\_send != 0)  sendAgentTime := sendAgentTime +  seconds\_send  writeln "sendAgentTime : " +  sendAgentTime + " seconds"  descriptor :=  matFromJson(p.getDescriptor())  writeln descriptor  writeln p.getImageName()  surf(p.getImageName())  String SenderAgent :=  msg.getSender().getName()  String[] parts := SenderAgent.split("@")  String part1 := parts[0]  String part2 := parts[1]  String[] parts2 := part2.split(":")  String IpAddress := parts2[0]  writeln "IpAddress : " + IpAddress  ACLMessage reply := new  ACLMessage(ACLMessage.CONFIRM)    reply.setLanguage("JavaSerialization")  writeln msg.getSender()  writeln "Name : " +  msg.getSender().getName() + " Local  Name : " +  msg.getSender().getLocalName() + "  Address " +  msg.getSender().getAddressesArray()[0]    reply.addReceiver(msg.getSender())  **try**  reply.setContentObject(d)  **catch** (IOException ex)  Logger.getLogger(TestObject.  class.getName()).log(Level.SEVERE,  NIL, ex)  myAgent.send(reply)  **else**  writeln getLocalName() + " read Java  String " + msg.getContent()  String[] parts :=  msg.getContent().split("/")  String part1 := parts[0]  String part2 := parts[1]  writeln part1 + " " + part2  **catch** (UnreadableException e3)  System.err.println(getLocalName() + "  catched exception " + e3.getMessage())  ) |

*Pseudocode* 4.9 Kode program *Ticker behaviour* pada *server*

### *Offloading Framework*

Hal pertama yang dilakukan pada *framework* yang melakukan penerepan metode *offloading* adalah menginisialisasi tampilan layar perangkat lunak menjadi tampilan mode kamera untuk mengambil datatest melalui *Pseudocode* 4.10. Pemanggilan fungsi untuk pengecekan kondisi koneksi internet juga dilakukan pada fungsi onCreate. Gambar citra datatest akan diperoleh dengan menekan tombol yang telah diatur pada fungsi onKeyDown. Gambar citra yang diperoleh dilakukan kompresi agar sesuai dengan citra dataset saat dilakukan *image recognition*. Selanjutnya akan dimulai eksekusi *image recognition* secara *asynchronous*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | @Override  **function** onCreate(Bundle savedInstanceState)  super.onCreate(savedInstanceState)  getWindow().setFlags(WindowManager.  LayoutParams.FLAG\_FULLSCREEN,  WindowManager.LayoutParams.  FLAG\_FULLSCREEN)    requestWindowFeature(Window.  FEATURE\_NO\_TITLE)  mResultView := new ResultView(this)  mPreview := new Preview(this)  setContentView(mPreview)  addContentView(mResultView,new LayoutParams  (LayoutParams.WRAP\_CONTENT,  LayoutParams.WRAP\_CONTENT))  checkConnection()  new DownloadImage().execute(mURL)  @Override  **function** onKeyDown(keycode, KeyEvent event)  **if** (keycode == KeyEvent.KEYCODE\_VOLUME\_UP)  **if** (mResultView.IsShowingResult)  mResultView.IsShowingResult :=  false  **else if** (mCameraReadyFlag == true)  mCameraReadyFlag := false  mPreview.camera.takePicture  (shutterCallback, rawCallback,  jpegCallback)  return true  return super.onKeyDown(keycode, event)  ShutterCallback shutterCallback := new  ShutterCallback()  **function** onShutter()  PictureCallback rawCallback := new  PictureCallback()  @Override  **function** onPictureTaken([] arg0,  android.hardware.Camera arg1)    PictureCallback jpegCallback := new  PictureCallback()  @Override  **function** onPictureTaken([] imageData,  android.hardware.Camera camera)  **if** (imageData != NIL)  Intent mIntent := new Intent()  compressByteImage(mContext,  imageData, 75)  setResult(0, mIntent)  ServerTask task := new ServerTask()  task.execute(Environment.  getExternalStorage  Directory().toString() +  INPUT\_IMG\_FILENAME)  camera.startPreview() |

*Pseudocode* 4.10 Kode program inisialisasi perangkat *client* dalam memulai *framework*

Tahap selanjutnya adalah mengeksekusi *image recognition* yang dilakukan secara *asynchronous* melalui *Pseudocode* 4.11yang berarti proses komunikasi data terikat dengan waktu tetap serta kecepatan eksekusi cukup relatif bergantung kondisi perangkat pengirim dan penerima. Variabel dialog adalah sebuah *pop – up* pada tampilan untuk *client* yang digunakan untuk memantau proses yang sedang berjalan pada perangkat lunak. Pada fungsi doInBackground, dilakukan inisialisasi untuk memulai *image recognition* seperti inisialisasi *path* letak dataset dan datatest, jumlah dataset serta pengambilan *descriptor* dari datatest yang digunakan untuk eksekusi proses *image recognition* melaui surfDatatest() yang akan dijelaskan pada Bab 4.2.3. Selain itu, pemanggilan fungsi startRepeatingTask() dilakukan untuk memulai eksekusi secara *asynchronous* baik secara lokal maupun *offload* ke *server*. Pada fungsi onProgressUpdate, dilakukan pembaruan status dari *pop – up* tampilan. Pada fungsi onPostExecute, pemanggilan fungsi startRepeatingBar(), digunakan sebagai pemantau proses *image recognition*. Fungsi ini digunakan untuk melihat progress berjalannya eksekusi *image recognition* yang ditampilkan pada *pop – up*  dialog.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | **class** ServerTask inherits AsyncTask<String,  Integer, Void>  ServerTask()  dialog := new ProgressDialog(mContext)  **function** onPreExecute()  dialog.setMessage("Photo captured")  dialog.show()  @Override  **function** doInBackground(String... params)  publishProgress(UPLOADING\_  PHOTO\_STATE)  surfDatatest()  publishProgress(SERVER\_PROC\_STATE)    startRepeatingTask()    mCameraReadyFlag := true  return NIL  @Override  **function** onProgressUpdate(Integer...  progress)  **if** (progress[0] == UPLOADING\_PHOTO\_STATE)  dialog.setMessage("Uploading")  dialog.setCanceledOnTouch  Outside(false)  dialog.show()  **else if** (progress[0] ==  SERVER\_PROC\_STATE)  **if** (dialog.isShowing())  dialog.dismiss()  dialog.setMessage("Processing")  dialog.show()  @Override  **function** onPostExecute(Void param)  startRepeatingBar() |

*Pseudocode* 4.11 Kode program inisialisasi penerapan *Asynchronous Task* pada *framework*

Selanjutnya dilakukan pengecekan status sebelum dilakukannya eksekusi beban kerja *image recognition* melalui *Pseudocode* 4.12. Pada fungsi run() secara *asynchronous* dari pemanggilan fungsi startRepeatingTask() diawali dengan pengecekan status dari eksekusi beban kerja yang sedang berjalan. Syarat dieksekusinya beban kerja baru adalah jika beban kerja sebelumnya selesai dikerjakan.

Jika eksekusi baru dimulai, maka beban kerja sebelumnya dianggap sudah selesai. Jika eksekusi sebelumnya dilakukan secara *offloading*, dilakukan pengecekan status beban kerja pada kelas java yang mengimplementasikan *agent* dan jika eksekusi beban kerja selesai dilakukan maka di cetak waktu lama eksekusi beban kerja pada *console* Android Studio dan disimpan untuk data histori waktu eksekusi *offloading* yang dibutuhkan oleh *decision maker* pada *framework*. Selain itu, juga dilakukan pengecekan jika eksekusi sebelumnya dilakukan secara *offloading* akan tetapi dibatalkan menjadi eksekusi lokal karena koneksi internet yang buruk. Pengecekan yang terakhir adalah jika eksekusi sebelumnya dilakukan secara lokal dan selesai dilakukan maka di cetak waktu lama eksekusi beban kerja pada *console* Android Studio dan disimpan untuk data histori waktu eksekusi lokal yang dibutuhkan oleh *decision maker* pada *framework*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **function** startRepeatingTask()  AgentTask.run()  **function** stopRepeatingTask()  handler.removeCallbacks(AgentTask)  Runnable AgentTask := new Runnable()  i := 0  @Override  **function** run()  agentCreated := SendObjectAgent.  AgentCreated()  StatusTask := false  **if** (i == 0)  StatusTask := true  **else if** (Offloading == true)  **if** (StatusTaskLocal)  StatusTask := StatusTaskLocal  StatusTaskLocal := false  **else**  StatusTask := SendObjectAgent.  FinishTask()  **if** (StatusTask)  endTimeTaskOffload()  **else if** (StatusTaskLocal & Offloading ==  false)  StatusTask := StatusTaskLocal  StatusTaskLocal := false  endTimetaskLocal() |

*Pseudocode* 4.12 Kode program pengecekan status eksekusi beban kerja *image recognition*

Dimulainya eksekusi beban kerja *image recognition* pada *framework* dapat dilihat melalui *Pseudocode* 4.13. Ada dua studi kasus dalam memulai eksekusi beban kerja. Studi kasus pertama adalah beban kerja tersebut adalah beban kerja baru disebabkan beban kerja sebelumnya sudah selesai di eksekusi baik secara lokal maupun *offloading* dan hasil eksekusi telah didapatkan. Studi kasus kedua adalah beban kerja tersebut adalah beban kerja yang belum selesai dieksekusi secara sempurna oleh metode *offloading* disebabkan perubahan koneksi internet menjadi buruk sehingga dialihkan menjadi eksekusi secara lokal. Untuk menangani studi kasus tersebut, jika beban kerja tersebut adalah beban kerja baru maka secara langsung dilakukan pengecekan koneksi internet untuk keputusan metode eksekusi selanjutnya. Jika beban kerja tersebut adalah beban kerja yang belum selesai dieksekusi secara sempurna maka status beban kerja dianggap selesai dikerjakan akan tetapi urutan file nama gambar dataset diundur satu kali agar dialihkan pada metode eksekusi secara lokal.

Pada beban kerja yang diputuskan untuk dieksekusi menggunakan metode *offloading*, akan dilakukan pengambilan beberapa data yang dibutuhkan untuk dilakukan *image recognition* dengan datatest pada *server* yang telah dijelaskan pada Bab 4.2.1.1 seperti nama file dataset, data *string* dari *descriptor* datatest yang didapat menggunakan fungsi matToJson yang akan dijelaskan pada Bab 4.2.4. Dalam pembuatan *agent*, dibutuhkan beberapa data seperti nama *agent* yang terdiri dari kombinasi model perangkat client, tanggal, bulan, tahun, jam, menit dan detik yang bertujuan untuk membedakan *agent* yang satu dengan yang lainnya. Selain itu, agentStartupCallback dibutuhkan untuk menemukan *container* tempat dimana *agent* dibuat serta SendObjectAgent.class.getName() digunakan untuk mereferensi kelas java yang mengimplementasi *agent* yang akan dibuat. Selain itu, dilakukan pengambilan data waktu dimulainya pembuatan *agent* untuk kebutuhan faktor penentu metode *offloading* yang akan dijelaskan pada Bab 4.2.6. Pada beban kerja yang diputuskan untuk dieksekusi secara lokal, akan dilakukan pengambilan data hanya nama file dataset untuk dilakukan proses *image recognition* secara lokal dengan fungsi surf() serta dilakukan pengambilan data waktu dimulainya eksekusi secara lokal sama seperti pada metode *offloading*. Jika semua dataset telah dilakukan *image recognition* dengan datatest, maka dipanggil fungsi stopRepeatingTask() untuk menghentikan eksekusi beban kerja. Berjalannya eksekusi beban kerja pada *framework* dapat dilihat pada **finally** dimana eksekusi beban kerja dilakukan secara teratur selama 1 detik dengan syarat beban kerja sebelumnya telah selesai dilakukan.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | **Try**  **if** (corruptTask & i != 0)  StatusTask := true  **if** (StatusTask)  checkConnection()  **if** (i < files.length & StatusTask == true &  Offloading == true)  **if** (corruptTask & i != 0)  i := i – 1  corruptTask := false  Log.d(TAG, "CurrentFileName:" +  currentFileTrain)  Log.d(TAG, "FileName:" +  files[i].getName())  file\_train := files[i].getName()  **if** (currentFileTrain.equals(file\_train))  i++  file\_train := files[i].getName()  Log.d(TAG, "NewFileName:" +  files[i].getName())  JSONdescCamera := matToJson(descCamera)  ImageName := file\_train  String deviceModel := Build.MANUFACTURER  + "/" + Build.MODEL + "/" +  Build.VERSION.RELEASE + "/" +  Build.VERSION\_CODES.class.getFields()  [android.os.Build.VERSION.SDK\_INT].  getName()  Calendar c := Calendar.getInstance()  date := c.get(Calendar.DATE)  month := c.get(Calendar.MONTH)  year := c.get(Calendar.YEAR)  hour := c.get(Calendar.HOUR)  min := c.get(Calendar.MINUTE)  seconds := c.get(Calendar.SECOND)  startTimeTask()  createAgent("agentOf" + deviceModel + "-  in-" + date + "/" + month + "/" + year +  "at" + hour + ":" + min + ":" + seconds,  SendObjectAgent.class.getName(),  agentStartupCallback)  i++  **if** (i < files.length & StatusTask == true &  Offloading == false)  **if** (corruptTask & i != 0)  i := i – 1  corruptTask := false  Log.d(TAG, "CurrentFileName:" +  currentFileTrain)  Log.d(TAG, "FileName:" +  files[i].getName())  file\_train := files[i].getName()  **if** (currentFileTrain.equals(file\_train))  i++  file\_train := files[i].getName()  Log.d(TAG, "NewFileName:" +  files[i].getName())  startTimeTask()  surf(file\_train)  i++  **if** (i >= files.length)  stopRepeatingTask()  **finally**  StatusTask := false  handler.postDelayed(AgentTask, 1000) |

*Pseudocode* 4.13 Kode program metode eksekusi beban kerja *image recognition* pada *framework*

Selain eksekusi yang dilakukan secara bertahap dan teratur pada beban kerja, dibutuhkan juga suatu eksekusi secara bertahap dan teratur terhadap pemantauan berjalannya eksekusi beban kerja yang diperlukan oleh client sebagai *user interface* dalam perangkat lunak melalui *Pseudocode* 4.14. Pada fungsi run() secara *asynchronous* dari pemanggilan fungsi startRepeatingBar() diawali dengan pengecekan status dari eksekusi beban kerja yang telah selesai dilakukan dan pembaruan pada tampilan *pop – up* dialog. Selain itu, terdapat pengecekan jika semua beban kerja telah selesai dilakukan pada *image recognition* maka dihilangkannya *pop – up* dialoguntuk menghentikan pemantauan eksekusi beban kerja. Berjalannya pemantauan eksekusi beban kerja pada *framework* dapat dilihat pada **finally** dimana pemantauan dilakukan secara teratur selama 0.5 detik.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | **function** startRepeatingBar()  AgentBar.run()  **function** stopRepeatingBar()  handler.removeCallbacks(AgentBar)  Collections.sort(listdata, new  Comparator<data>()  @Override  function compare(data t1, data t2)  return t2.getAcceptpoint() –  t1.getAcceptpoint()  )  Intent result := new Intent(getBaseContext(),  PageResult.class)  result.putExtra("resultListData",  (Serializable) listdata)  startActivity(result)  Runnable AgentBar := new Runnable()  @Override  **function** run()  **try**  **if** (GetFinishAgentTask !=  FinishAgentTask)  dialog.setMessage("Processing " +  FinishAgentTask)  GetFinishAgentTask :=  FinishAgentTask  **if** (dialog.isShowing() &  FinishAgentTask >= files.length)  dialog.dismiss()  mResultView.invalidate()  stopRepeatingBar()  **finally**  handler.postDelayed(AgentBar, 500) |

*Pseudocode* 4.14 Kode program kode pemantauan eksekusi beban kerja *image recognition* pada *framework*

Untuk menentukan keputusan metode eksekusi yang optimal antara *offloading* dan lokal. Dibutuhkan *decision maker* yang menentukan keputusan berdasarkan faktor – faktor penentu *offloading* yang dapat dilihat secara bertahap melalui *Pseudocode* 4.15, *Pseudocode* 4.16, *Pseudocode* 4.17, *Pseudocode* 4.18dan *Pseudocode* 4.19. Pada *Pseudocode* 4.15, dilakukan pengecekan adanya koneksi internet pada perangkat. Pada fungsi showSnack() dilakukan inisialisasi variabel batteryLevel yang digunakan pada salah satu faktor penentu metode *offloading*. Nilai batteryLevel didapatkan dari fungsi getBatteryPercent() yang akan dijelaskan pada Bab 4.2.6. Variabel weightOffload digunakan untuk pembobotan nilai yang digunakan dalam dasar pengambilan keputusan oleh *decision maker*.

Jika dalam pengecekan koneksi internet dinyatakan adanya koneksi pada perangkat, maka selanjutnya akan dilakukan pengecekan terhadap tersedianya microRuntimeServiceBinder pada perangkat *client*. Jika belum tersedia, maka dibuat microRuntimeServiceBinder baru menggunakan fungsi bindService() dengan parameter *host* dan *port* yang ditampung oleh SharedPreferences. Pada studi kasus lain, microRuntimeServiceBinder sudah terbuat dan menjalankan eksekusi beban kerja secara *offloading* akan tetapi koneksi internet terputus sehingga eksekusi dialihkan secara lokal. Apabila koneksi tersambung kembali dan diputuskan untuk melakukan metode *offloading*, maka harus dilakukan pembuatan microRuntimeServiceBinder yang baru dan menghapus yang lama.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **function** checkConnection()  isConnected :=  ConnectivityReceiver.isConnected()  showSnack(isConnected)  **function** showSnack(isConnected)  batteryLevel := getBatteryPercent()  weightOffload := 0  **if** (isConnected)  **if** (NOT bindservice)  **if** (rateExecutionTimeOffload != 0)  unbindService(mServiceConnection)  microRuntimeServiceBinder := NIL  SharedPreferences sharedPreferences :=  SURFExampleActivity.this.  getSharedPreferences(Constants.  PREFS\_FILE\_NAME, Context.  MODE\_PRIVATE)  String host :=  sharedPreferences.getString  (Constants.PREFS\_HOST\_ADDRESS,  ipAddress)  String port := settings.getString  ("defaultPort", "")  Log.e(TAG, "Connecting to --> host :  " + host + " - " + "port : " + port)  bindService(host, port,  agentStartupCallback) |

*Pseudocode* 4.15 Kode program *Decision Maker* melakukan pengecekan terhadap ketersediaan koneksi internet dan *microRuntimeServiceBinder*

Pada *Pseudocode* 4.16, dilakukan pengecekan kualitas koneksi internet dengan 4 pembagian, yaitu *Excellent*, *Good*, *Moderate* dan *Poor*. Jika dinyatakan kualitas internet *Excellent*, maka pembobotan ditambahkan nilai tiga. Jika dinyatakan kualitas internet *Good*, maka pembobotan ditambahkan nilai dua. Jika dinyatakan kualitas internet *Moderate*, maka pembobotan ditambahkan nilai satu. Jika dinyatakan kualitas internet *Poor*, maka tidak ada penambahan bobot nilai.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | Connectivity := new ConnectivityReceiver()  NetworkInfo info := Connectivity.  getNetworkInfo(getBaseContext())  **if** (mConnectionClassManager.getCurrentBandwidth  Quality() == ConnectionQuality.EXCELLENT |  mConnectionClassManager.getCurrentBandwidth  Quality() == ConnectionQuality.GOOD |  mConnectionClassManager.getCurrentBandwidth  Quality() == ConnectionQuality.MODERATE)  **if** (mConnectionClassManager.getCurrent  BandwidthQuality() ==  ConnectionQuality.EXCELLENT)  weightOffload := weightOffload + 3  Log.e(TAG, "+++weightOffload  Connection: " + mConnectionClass  Manager.getCurrent  BandwidthQuality())  **if** (mConnectionClassManager.getCurrent  BandwidthQuality() ==  ConnectionQuality.GOOD)  weightOffload := weightOffload + 2  Log.e(TAG, "++weightOffload  Connection: " + mConnectionClass  Manager.getCurrent  BandwidthQuality())  **if** (mConnectionClassManager.getCurrent  BandwidthQuality() ==  ConnectionQuality.MODERATE)  weightOffload++  Log.e(TAG, "+weightOffload  Connection: " + mConnectionClass  Manager.getCurrent  BandwidthQuality())  **if** (mConnectionClassManager.getCurrentBandwidth  Quality() == ConnectionQuality.POOR)  Log.d(TAG, "Internet Poor")  StatusTaskLocal := true |

*Pseudocode* 4.16 Kode program *Decision Maker* melakukan pengecekan terhadap kualitas koneksi internet

Pada *Pseudocode* 4.17, dilakukan pengecekan terhadap level baterai sebagai salah satu faktor penentu metode *offloading*, Jika level baterai berada pada nilai dibawah dua puluh yang berarti harus dilakukan penghematan lebih agar perangkat *client* tidak mati akibat kehabisan baterai, maka ditambahkan satu nilai pembobotan untuk lebih cenderung dilakukan metode *offloading*.

|  |  |
| --- | --- |
| 1  2  3  4 | Log.e(TAG, "Battery level : " + batteryLevel)  **if** (batteryLevel < 20)  weightOffload++  Log.e(TAG, "+weightOffload Battery: " +  batteryLevel) |

*Pseudocode* 4.17 Kode program *Decision Maker* melakukan pengecekan terhadap kondisi level baterai

Pada *Pseudocode* 4.18*,* dilakukan pengecekan tehadap berjalannya eksekusi beban kerja serta pengambilan keputusan oleh *Decision Maker*. Hal yang dilakukan pertama kali adalah eksekusi beban kerja yang pertama harus dilakukan secara lokal sesuai dengan penjelasan pada Bab 3.5 untuk memperoleh data waktu eksekusi secara lokal yang selanjutnya akan dibandingkan waktu eksekusi secara *offloading*. Eksekusi beban kerja selanjutnya akan dilakukan secara *offloading* jika tersedia koneksi internet dan kualitas koneksi tidak buruk. Setelah dilakukan kedua metode eksekusi beban kerja pada *framework*, *Decision Maker* akan membandingkan waktu eksekusi tiap metode eksekusi. Jika selisih rata – rata waktu eksekusi secara *offloading* dengan rata – rata waktu eksekusi secara lokal kurang dari sama dengan delapan detik, maka ditambahkan satu nilai pembobotan.

Untuk melakukan eksekusi beban kerja secara *offloading*, nilai pembobotan harus lebih dari dua. Akan tetapi, jika dalam eksekusi beban kerja secara *offloading* koneksi internet terputus secara otomatis eksekusi beban kerja dialihkan secara lokal oleh *framework*. Ketika koneksi internet kembali tersedia pada perangkat, maka dilakukan pengecekan terhadap kualitas koneksi internet. Jika kualitas koneksi internet buruk maupun normal, maka eksekusi beban kerja tetap dilakukan secara lokal sebab koneksi internet saat awal tersedia pada perangkat *client* tidak stabil sehingga menyebabkan gagalnya pembuatan *agent* pada perangkat *client*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **if** (rateExecutionTimeOffload == 0 &  mConnectionClassManager.getCurrent  BandwidthQuality() != ConnectionQuality.POOR &  mConnectionClassManager.getCurrent  BandwidthQuality() != ConnectionQuality.UNKNOWN)  Offloading := true  **if** (rateExecutionTimeLocal == 0)  Offloading := false  **else if** (rateExecutionTimeLocal != 0)  **if** (rateExecutionTimeOffload != 0)  Log.e(TAG, "Delta Execution Time: " +  (rateExecutionTimeOffload –  rateExecutionTimeLocal))  **if** (rateExecutionTimeOffload –  rateExecutionTimeLocal <= 8)  Log.e(TAG, "+weightOffload Delta  Execution Time: " +  (rateExecutionTimeOffload –  rateExecutionTimeLocal))  weightOffload++  **if** (weightOffload >= 2)  Offloading := true  **else**  Offloading := false  **if** (corruptTaskOffload)  **if** (mConnectionClassManager.getCurrent  BandwidthQuality() ==  ConnectionQuality.MODERATE &  mConnectionClassManager.getCurrent  BandwidthQuality() ==  ConnectionQuality.POOR)  Offloading := false  corruptTaskOffload := false |

*Pseudocode* 4.18 Kode program *Decision Maker* melakukan pembobotan nilai untuk keputusan metode eksekusi

Pada *Pseudocode* 4.19, dilakukan eksekusi secara lokal jika tidak tersedia koneksi pada perangkat *client*. Inisialisasi pada variabel *Decision Maker* dilakukan agar *framework* melakukan eksekusi beban kerja secara lokal. Selain itu, terdapat beberapa variabel yang perlu di inputkan nilainya jika dalam eksekusi beban kerja secara *offloading* koneksi internet terputus dan dialihkan pada eksekusi beban kerja secara lokal

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **else**  Log.e(TAG, "Not Connected")  mConnectionClassManager.reset()  Log.e(TAG, "Connection class : " +  mConnectionClassManager.getCurrent  BandwidthQuality().toString())  **if** (Offloading)  corruptTask := true  corruptTaskOffload := true  corruptTaskHandler := true  file\_train\_corrupt := file\_train  Offloading := false  bindservice := false |

*Pseudocode* 4.19 Kode program *Decision Maker* melakukan metode eksekusi lokal jika tidak tersedianya koneksi internet pada perangkat *client*

Tahap akhir dari *framework* adalah menyimpan data hasil eksekusi beban kerja pada suatu *array* bertipe kelas *data* yaitu listdata melalui *Pseudocode* 4.20 yang selanjutnya akan diurutkan secara *descending* (besar ke kecil) menurut jumlah titik fitur yang sesuai kriteria pada fungsi stopRepeatingBar() pada *Pseudocode* 4.14 serta ditampilkan pada *console* Android Studio IDE untuk pengecekan. Selain itu, dilakukan pengecekan koneksi internet kembali melalui DownloadImage().execute() yang akan dijelaskan pada Bab 4.2.6.

|  |  |
| --- | --- |
| 1  2  3  4  6  7  8  9  10  11  12  13  14  15  16  17 | **function** exportLogConsole(data log)  Message logMessage := new Message()  logMessage.obj := log  handlerMessage.sendMessage(logMessage)  **function** onCreate(Bundle savedInstanceState)  handlerMessage := new Handler()  @Override  **function** handleMessage(Message  dataServer)  **if** (NOT  file\_train\_corrupt.equals(((data)  dataServer.obj).getNama()) |  corruptTaskHandler)  corruptTaskHandler := false  listdata.add((data)  dataServer.obj)  Log.i(TAG,  dataServer.obj.toString())  Log.i(TAG, "name : " + ((data)  dataServer.obj).getNama() + "  min distance : " + ((data)  dataServer.obj).getMin\_dist() +"  max distance : " + ((data)  dataServer.obj).getMax\_dist() +"  accepted : " + ((data)  dataServer.obj).  getAcceptpoint()+"\n" + "  Send Agent Time : " + ((data)  dataServer.obj).  getSendAgentTime() + "\n" + "  Receive Agent Time : " + ((data)  dataServer.obj).  getReceiveAgentTime() + "\n" + "  AgentTransportTime : " + ((data)  dataServer.obj).  getTransportAgentTime())  currentFileTrain := ((data)  dataServer.obj).getNama()  FinishAgentTask++  Log.i(TAG, " Task : " +  FinishAgentTask)  new DownloadImage().execute(mURL) |

*Pseudocode* 4.20 Kode program menyimpan hasil eksekusi beban kerja pada *framework*

### Metode SURF

Hal yang dibutuhkan untuk melakukan metode SURF adalah memperoleh *descriptor* dari matriks citra daun datatest dan dataset. Untuk agar memperoleh *descriptor* yang optimal dari matriks citra datatest maka dilakukan kompresi melalui *Pseudocode* 4.21.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | **function** compressByteImage(Context mContext, []  imageData, quality)  File sdCard := Environment.getExternal  StorageDirectory()  FileOutputStream fileOutputStream := NIL  **try**  BitmapFactory.Options options := new  BitmapFactory.Options()  options.inSampleSize := 1  Bitmap myImage :=  BitmapFactory.decodeByteArray  (imageData, 0, imageData.length,options)  fileOutputStream := new FileOutputStream  (sdCard.toString() + INPUT\_IMG\_FILENAME)  BufferedOutputStream bos := new Buffered  OutputStream(fileOutputStream)  myImage.compress(CompressFormat.JPEG,  quality, bos)  bos.flush()  bos.close()  fileOutputStream.close()  **catch** (FileNotFoundException e)  Log.e(TAG, "FileNotFoundException")  e.printStackTrace()  **catch** (IOException e)  Log.e(TAG, "IOException")  e.printStackTrace()  return true |

*Pseudocode* 4.21 Program kompresi citra datatest

Setelah dikompresi, dilakukan pengambilan *descriptor* pada citra datatest melalui *Pseudocode* 4.22 dengan menggunakan FeatureDetector SURF serta DescriptorExtractor SURF. Dilakukan juga pewarnaan *grayscale* pada tahap *preprocessing*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | **function** surfDatatest()  System.loadLibrary("opencv\_java")  System.loadLibrary("nonfree")  Bitmap inputImage, cameraImage  FeatureDetector detector :=  FeatureDetector.create(FeatureDetector.SURF)  DescriptorExtractor descriptorExtractor :=  DescriptorExtractor.create  (DescriptorExtractor.SURF)  DescriptorMatcher descriptorMatcher :=  DescriptorMatcher.create  (DescriptorMatcher.FLANNBASED)  Mat rgbaCamera, descCamera, result  MatOfDMatch matches  MatOfKeyPoint keyPointsCamera  rgbaCamera := new Mat()  descCamera := new Mat()  matches := new MatOfDMatch()  result := new Mat()  BitmapFactory.Options bmOptions := new  BitmapFactory.Options()  File testimage := new  File(Environment.getExternalStorage  Directory().toString(), "  INPUT\_IMG\_FILENAME")  cameraImage := BitmapFactory.decodeFile  (testimage.getAbsolutePath(), bmOptions)  cameraImage := Bitmap.createScaledBitmap  (cameraImage, cameraImage.getWidth(),  cameraImage.getHeight(), true)  Utils.bitmapToMat(cameraImage, rgbaCamera)  keyPointsCamera := new MatOfKeyPoint()  Imgproc.cvtColor(rgbaCamera, rgbaCamera,  Imgproc.COLOR\_RGBA2GRAY)  detector.detect(rgbaCamera, keyPointsCamera)  descriptorExtractor.compute(rgbaCamera,  keyPointsCamera, descCamera)  Features2d.drawKeypoints(rgbaCamera,  keyPointsCamera, rgbaCamera)  Utils.matToBitmap(rgbaCamera, cameraImage) |

*Pseudocode* 4.22 Kode Program mendapatkan *descriptor* pada datatest

Pada eksekusi beban kerja secara lokal, dilakukan juga pengambilan *descriptor* pada citra dataset hampir sama dengan datatest melalui *Pseudocode* 4.23 dengan menggunakan FeatureDetector SURF serta DescriptorExtractor SURF. Dilakukan juga pewarnaan *grayscale* pada tahap *preprocessing*. Selain itu, untuk menghitung jarak kemiripan digunakan DescriptorMatcher FLANNBASED. Selanjutnya jarak fitur antara datatest dan dataset yang didapatkan berupa titik – titik tersebut dilakukan pengecekan, jika nilai jarak kemiripan diterima yaitu 0.15 maka akan acceptPoint pada kelas *Java Object Serialization* *data* yang berarti semakin besar acceptPoint maka datatest dan dataset semakin mirip.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | **function** surf(String imageName)  FeatureDetector detector2 :=  FeatureDetector.create(FeatureDetector.SURF)  DescriptorExtractor descriptorExtractor2 :=  DescriptorExtractor.create  (DescriptorExtractor.SURF)  DescriptorMatcher descriptorMatcher2 :=  DescriptorMatcher.create  (DescriptorMatcher.FLANNBASED)  Mat rgba := new Mat()  Mat desc := new Mat()  BitmapFactory.Options bmOptions := new  BitmapFactory.Options()  File image := new File(path, imageName)  inputImage := BitmapFactory.decodeFile  (image.getAbsolutePath(), bmOptions)  inputImage := Bitmap.createScaledBitmap  (inputImage, inputImage.getWidth(),  inputImage.getHeight(), true)  Utils.bitmapToMat(inputImage, rgba)  Imgproc.cvtColor(rgba, rgba,  Imgproc.COLOR\_RGBA2GRAY)  detector2.detect(rgba, keyPoints)  descriptorExtractor2.compute  (rgba, keyPoints, desc)  descriptorMatcher2.match  (descCamera, desc, matches)  count := 0  max\_dist := 0  min\_dist := 0.15  List<DMatch> matchesList := matches.toList()  Features2d.drawKeypoints  (rgba, keyPoints, rgba)  **for** i := 0 loop till i < descCamera.rows() by  i++ each step  dist := matchesList.get(i).distance  **if** (dist < 0.15)  if (dist < min\_dist)  min\_dist := dist  count++  **if** (dist > max\_dist)  max\_dist := dist  data newdata := new data(imageName, count,  min\_dist, max\_dist, 0, "0", 0)  exportLogConsole(newdata)  StatusTaskLocal := true  Log.e(TAG, "Execution Local") |

*Pseudocode* 4.23 Kode program eksekusi *image recognition* secara lokal

Pada eksekusi beban kerja secara *offloading*, dilakukan juga pengambilan *descriptor* pada citra dataset yang disediakan oleh *server* melalui *Pseudocode* 4.24dengan menggunakan FeatureDetector SURF serta DescriptorExtractor SURF sama seperti pada eksekusi secara lokal. Dilakukan juga pewarnaan *grayscale* pada tahap *preprocessing*. Hampir seluruh urutan tahapnya sama dengan metode eksekusi beban kerja secara lokal yang telah dijelaskan sebelumnya, hanya perbedaan dalam menulis kode sumber.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | String path :=  "C://xampp//htdocs//ServerCode//training//  test//training//"  System.loadLibrary(Core.NATIVE\_LIBRARY\_NAME)  **function** surf(String imageName)  detector := FeatureDetector.  create(FeatureDetector.SURF)  descriptorExtractor :=  DescriptorExtractor.  create(DescriptorExtractor.SURF)  descriptorMatcher :=  DescriptorMatcher.create  (DescriptorMatcher.FLANNBASED)  Mat rgba := new Mat()  Mat desc := new Mat()  BitmapFactory.Options bmOptions := new  BitmapFactory.Options()  File image := new File(path, imageName)  rgba := Highgui.imread(path + imageName,  Highgui.CV\_LOAD\_IMAGE\_COLOR)  MatOfKeyPoint keyPoints := new  MatOfKeyPoint()  Imgproc.cvtColor(rgba, rgba,  Imgproc.COLOR\_RGBA2GRAY)  detector.detect(rgba, keyPoints)  descriptorExtractor.compute(rgba, keyPoints,  desc)  matches := new MatOfDMatch()  descriptorMatcher.match(descriptor, desc,  matches)  count := 0  max\_dist := 0  min\_dist := 0.15  List<DMatch> matchesList := matches.toList()  Features2d.drawKeypoints(rgba, keyPoints,  rgba)  **for** i := 0 loop till i < descriptor.rows() by  i++ each step  dist := matchesList.get(i).distance  **if** (dist < 0.15)  if (dist < min\_dist)  min\_dist := dist  count++  **if** (dist > max\_dist)  max\_dist := dist  System.err.println("result " + " min distance  : " + min\_dist + " max distance : " + max\_dist  + " accepted : " + count)  TimeZone local :=  TimeZone.getTimeZone("Asia/Jakarta")  Calendar now := Calendar.getInstance(local)  SimpleDateFormat formatter := new  SimpleDateFormat("HH:mm:ss.SSS")  formatter.setTimeZone(local)  String ReceiveAgentStartTime :=  formatter.format(now.getTime())  d := new data(imageName, count, min\_dist,  max\_dist, sendAgentTime,  ReceiveAgentStartTime, 0) |

*Pseudocode* 4.24 Kode Program eksekusi *image recognition* secara *offloading*

### Java *Class Object Serialization* dari *Client* ke *Server*

Proses awal yang dilakukan dalam membuat kelas *Java Object Serialization* dari *client* ke *server* adalah mengubah data matriks citra daun datatest menjadi data string agar dapat disisipkan pada variabel kelas *Java Object Serialization*. Pengubahan data matriks citra daun menjadi string dapat dilihat pada *Pseudocode* 4.25. Pada fungsi matToJson() dilakukan inisialisasi baris dan kolom serta tipe data dari nilai di dalam matriks. Pada studi kasus matriks citra daun pada tugas akhir ini, tipe data pada nilai matriks adalah *float* sehingga pada fungsi akan dikenal dengan tipe CvType.CV\_32F. Fungsi matToJson() akan mengembalikan nilai string dari konversi matriks citra daun datatest.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | **function** matToJson(Mat mat)  JsonObject obj := new JsonObject()  cols := mat.cols()  rows := mat.rows()  elemSize := mat.elemSize()  type := mat.type()  obj.addProperty("rows", mat.rows())  obj.addProperty("cols", mat.cols())  obj.addProperty("type", mat.type())  String dataString  **if** (type == CvType.CV\_32S OR type ==  CvType.CV\_32SC2 OR type == CvType.CV\_32SC3  OR type == CvType.CV\_16S)  [] data := [0..cols \* rows \* elemSize-1]  mat.get(0, 0, data)  dataString := new  String(Base64.encode(Serialization  Utils.serialize(data), Base64.DEFAULT))  **else if** (type == CvType.CV\_32F OR type ==  CvType.CV\_32FC2)  [] data := [0..cols \* rows \* elemSize-1]  mat.get(0, 0, data)  dataString := new  String(Base64.encode(Serialization  Utils.serialize(data), Base64.DEFAULT))  **else if** (type == CvType.CV\_64F OR type ==  CvType.CV\_64FC2)  [] data := [0..cols \* rows \* elemSize-1]  mat.get(0, 0, data)  dataString := new  String(Base64.encode(Serialization  Utils.serialize(data), Base64.DEFAULT))  **else if** (type == CvType.CV\_8U)  [] data := [0..cols \* rows \* elemSize-1]  mat.get(0, 0, data)  dataString := new String(Base64.encode  (data, Base64.DEFAULT))  **else**  throw new UnsupportedOperation  Exception("unknown type")  obj.addProperty("data", dataString)  Gson gson := new Gson()  String json := gson.toJson(obj)  return json |

*Pseudocode* 4.25 Kode program pengubahan data matriks menjadi string pada perangkat *client*

Untuk dapat menyisipkan berbagai data pada modul pengiriman data yang dikirim oleh *agent*, dibuatlah kelas *Java Object Serialization* ObjectDataMat seperti pada

*Pseudocode* 4.26. Pada kelas tersebut akan diinisialisasi serialVersionUID yang harus bernilai sama dengan serialVersionUID pada kelas *Java Object Serialization* ObjectDataMat yang berada pada *server* dengan tujuan agar modul pengiriman data dapat terkirim ke *server*. Inisialisasi data lain yang dilakukan adalah data yang dibutuhkan untuk *image recognition* seperti *descriptor* datatest, nama file dataset, dan waktu dimulainya eksekusi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **class** ObjectDataMat implements Serializable  serialVersionUID := 8551377534047755760L  System.loadLibrary("opencv\_java")  System.loadLibrary("nonfree")  String descCamera, ImageName,  SendAgentStartTime  ObjectDataMat(String desc, String ImgName,  String SendAgentStartTime)  this.descCamera := desc  this.ImageName := ImgName  this.SendAgentStartTime :=  SendAgentStartTime |

*Pseudocode* 4.26 Kode Program kelas *Java* *Object Serialization* ObjectDataMat pada perangkat *client*

Pada kelas *Java Object Serialization* ObjectDataMat pada *server* yang diperlihatkan pada *Pseudocode* 4.27, kontennya hampir sama dengan kelas *Java Object Serialization* ObjectDataMat pada perangkat *client*. Perbedaannya hanya terletak pada penambahan fungsi untuk mengambil nilai dari masing – masing variabel.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **class** ObjectDataMat implements Serializable  serialVersionUID := 8551377534047755760L  String descCamera, ImageName,  SendAgentStartTime  ObjectDataMat(String desc, String ImgName,  String SendAgentStartTime)  this.descCamera := desc  this.ImageName := ImgName  this.SendAgentStartTime :=  SendAgentStartTime  **function** getDescriptor()  return this.descCamera  **function** getImageName()  return this.ImageName  **function** getSendAgentStartTime()  return this.SendAgentStartTime  **function** toString()  return ("desc : " + descCamera.toString()  + "\n" + "image name : " +  ImageName.toString() + "\n" + "Send Agent  Start Time : " +  SendAgentStartTime.toString()) |

*Pseudocode* 4.27 Kode Program kelas *Java* *Object Serialization* ObjectDataMat pada perangkat *server*

Agar data *descriptor* yang berupa *string* harus dirubah terlebih dahulu menjadi tipe data *Mat* agar dapat diproses dalam *image recognition*. Pengubahan tipe data *string* menjadi tipe data *Mat* dilakukan oleh fungsi matFromJson() dapat dilihat pada *Pseudocode* 4.28. Proses pengubahannya hampir sama dengan fungsi matToJson() pada perangkat *client*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | **function** matFromJson(String json)  JsonParser parser := new JsonParser()  JsonObject JsonObject :=  parser.parse(json).getAsJsonObject()  rows := JsonObject.get("rows").getAsInt()  cols := JsonObject.get("cols").getAsInt()  type := JsonObject.get("type").getAsInt()  Mat mat := new Mat(rows, cols, type)  String dataString :=  JsonObject.get("data").getAsString()  **if** (type == CvType.CV\_32S OR type ==  CvType.CV\_32SC2 OR type == CvType.CV\_32SC3  OR type == CvType.CV\_16S)  [] data := SerializationUtils.deserialize  (Base64.decodeBase64  (dataString.getBytes()))  mat.put(0, 0, data)  **else if** (type == CvType.CV\_32F OR type ==  CvType.CV\_32FC2)  [] data := SerializationUtils.deserialize  (Base64.decodeBase64  (dataString.getBytes()))  mat.put(0, 0, data)  **else if** (type == CvType.CV\_64F OR type ==  CvType.CV\_64FC2)  [] data := SerializationUtils.deserialize  (Base64.decodeBase64  (dataString.getBytes()))  mat.put(0, 0, data)  **else if** (type == CvType.CV\_8U)  [] data := Base64.decodeBase64  (dataString.getBytes())  mat.put(0, 0, data)  **else**  throw new UnsupportedOperation  Exception("unknown type")  return mat |

*Pseudocode* 4.28 Kode program pengubahan string menjadi Data Matriks pada perangkat *server*

### Java *Class Object Serialization* dari *Server* ke *Client*

Proses awal yang dilakukan dalam membuat kelas *Java Object Serialization* dari *server* ke *client* adalah memastikan data hasil proses *image recognition* telah didapatkan dan siap untuk di inisialisasi pada kelas *Java Object Serialization* data yang dapat dilihat melalui *Pseudocode* 4.29. Implementasi kelas *Java Object Serialization* data untuk modul pengiriman data dari *server* ke *client* hampir sama dengan penerapan kelas *Java Object Serialization* ObjectDataMat yang telah dijelaskan sebelumnya. Terdapat data – data hasil *image recognition* yang diinisialisasi pada kelas *Java Object Serialization* data seperti acceptpoint, min\_dist, max\_dist, SendAgentTime, ReceiveAgentStartTime, dan ReceiveAgentTime. Terdapat juga fungsi – fungsi yang digunakan untuk mengambil dan mengubah data tiap – tiap variabel.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | **class** data implements Serializable  serialVersionUID := 8551377534047755760L  String name  acceptpoint  min\_dist  max\_dist  String ReceiveAgentStartTime  SendAgentTime  ReceiveAgentTime  data(String name, acceptpoint, min\_dist,  max\_dist, SendAgentTime, String  ReceiveAgentStartTime, ReceiveAgentTime)  this.name := name  this.acceptpoint := acceptpoint  this.min\_dist := min\_dist  this.max\_dist := max\_dist  this.SendAgentTime := SendAgentTime  this.ReceiveAgentStartTime :=  ReceiveAgentStartTime  this.ReceiveAgentTime := ReceiveAgentTime  **function** getNama()  return name  **function** setNama(String name)  this.name := name  **function** getAcceptpoint()  return acceptpoint  **function** setAcceptpoint(acceptpoint)  this.acceptpoint := acceptpoint  **function** getMin\_dist()  return min\_dist  **function** setMin\_dist(min\_dist)  this.min\_dist := min\_dist  **function** getMax\_dist()  return max\_dist  **function** setMax\_dist(max\_dist)  this.max\_dist := max\_dist  **function** getSendAgentTime()  return SendAgentTime  **function** setSendAgentTime(SendAgentTime)  this.SendAgentTime := SendAgentTime  **function** getReceiveAgentStartTime()  return ReceiveAgentStartTime  **function** setReceiveAgentStartTime(String  ReceiveAgentStartTime)  this.ReceiveAgentStartTime :=  ReceiveAgentStartTime  **function** getReceiveAgentTime()  return ReceiveAgentTime  **function** setReceiveAgentTime  (ReceiveAgentTime)  this.ReceiveAgentTime := ReceiveAgentTime |

*Pseudocode* 4.29 Kode Program kelas *Java* *Object Serialization* data pada perangkat *server*

Pada kelas *Java Object Serialization* data pada perangkat *client* yang dapat dilihat melalui *Pseudocode* 4.30, inisialisasi variabelnya sama dengan kelas *Java Object Serialization* data pada *server*. Begitu juga dengan penerapan fungsi yang digunakan untuk mengambil dan mengubah data tiap – tiap variabel.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | **class** data implements Serializable  serialVersionUID := 8551377534047755760L  String name  acceptpoint  min\_dist  max\_dist  String ReceiveAgentStartTime  SendAgentTime  ReceiveAgentTime  data(String name, acceptpoint, min\_dist,  max\_dist, SendAgentTime, String  ReceiveAgentStartTime, ReceiveAgentTime)  this.name := name  this.acceptpoint := acceptpoint  this.min\_dist := min\_dist  this.max\_dist := max\_dist  this.SendAgentTime := SendAgentTime  this.ReceiveAgentStartTime :=  ReceiveAgentStartTime  this.ReceiveAgentTime := ReceiveAgentTime  **function** getNama()  return name  **function** setNama(String name)  this.name := name  **function** getAcceptpoint()  return acceptpoint  **function** setAcceptpoint(acceptpoint)  this.acceptpoint := acceptpoint  **function** getMin\_dist()  return min\_dist  **function** setMin\_dist(min\_dist)  this.min\_dist := min\_dist  **function** getMax\_dist()  return max\_dist  **function** setMax\_dist(max\_dist)  this.max\_dist := max\_dist  **function** getSendAgentTime()  return SendAgentTime  **function** setSendAgentTime(SendAgentTime)  this.SendAgentTime := SendAgentTime  **function** getReceiveAgentStartTime()  return ReceiveAgentStartTime  **function** setReceiveAgentStartTime(String  ReceiveAgentStartTime)  this.ReceiveAgentStartTime :=  ReceiveAgentStartTime  **function** getReceiveAgentTime()  return ReceiveAgentTime  **function** setReceiveAgentTime  (ReceiveAgentTime)  this.ReceiveAgentTime := ReceiveAgentTime    **function** getTransportAgentTime()  return (SendAgentTime + ReceiveAgentTime) |

*Pseudocode* 4.30 Kode Program kelas *Java* *Object Serialization* data pada perangkat *client*

### Faktor Penentu Metode *Offloading*

#### Kualitas Koneksi Internet

Pengecekan kualitas koneksi internet melalui *Pseudocode* 4.31 diawali dengan inisialisasi *library* konektivitas yang diimplementasikan oleh kelas utama SURFExampleActivity. Pada fungsi onCreate() dilakukan pemanggilan fungsi checkConnection() dan new DownloadImage().execute() dengan tujuan pengecekan koneksi internet dilakukan secara berkala dan dimunculkannya pemberitahuan jika terdapat perubahan terhadap kualitas koneksi internet selama proses eksekusi beban kerja image recognition berlangsung.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **class** SURFExampleActivity inherits Activity  implements ConnectivityReceiver.  ConnectivityReceiverListener    ConnectivityReceiver Connectivity  ConnectionClassManager  mConnectionClassManager  DeviceBandwidthSampler  mDeviceBandwidthSampler  ConnectionChangedListener mListener  String mURL := "https://c1.staticflickr.com  /6/5646/30422515475\_5482a5e51b\_b.jpg"  mTries := 0    ConnectionQuality mConnectionClass :=  ConnectionQuality.UNKNOWN  @Override  **function** onCreate(Bundle savedInstanceState)  super.onCreate(savedInstanceState)  mConnectionClassManager :=  ConnectionClassManager.getInstance()  mDeviceBandwidthSampler :=  DeviceBandwidthSampler.getInstance()  mListener := new  ConnectionChangedListener()  checkConnection()  new DownloadImage().execute(mURL) |

*Pseudocode* 4.31 Kode Program inisialisasi pengecekan koneksi internet pada perangkat *client*

*Pseudocode* 4.32 memperlihatkan fungsi dalam kode sumber yang digunakan untuk pengecekan secara berkala ketersediaan koneksi internet pada perangkat *client* melalui onNetworkConnectionChanged() maupun kualitas koneksi internet melalui ConnectionChangedListener.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @Override  **function** onNetworkConnectionChanged(isConnected)  showSnack(isConnected)  class ConnectionChangedListener implements  ConnectionClassManager.Connection  ClassStateChangeListener  @Override  function onBandwidthState  Change(ConnectionQuality bandwidthState)  mConnectionClass := bandwidthState  runOnUiThread(new Runnable()  @Override  function run()  Log.e(TAG, "Connection Class: " +  mConnectionClass.toString())  ) |

*Pseudocode* 4.32 Kode Program kondisi saat koneksi internet mengalami perubahan kualitas dan pengecekan ketersediaan

Untuk mengetahui kualitas koneksi internet pada perangkat *client*, dijalankan kelas DownloadImage yang berjalan secara *asynchronous* dalam pengecekan kualitas koneksi internet menggunakan *buffer* data saat melakukan *transmit* data melalui *Pseudocode* 4.33. Transmit dilakukan dengan mengunduh file random yang memiliki ukuran yang disesuaikan. Pada tugas akhir ini, data yang digunakan adalah file gambar dengan ukuran data kurang lebih 500 KB. *Buffer* data saat mengunduh file akan dibandingkan dengan referensi *buffer* data sesuai ukuran file yang telah dijelaskan pada Bab 3.4.1. Transmit data akan dihentikan jika kelas tersebut menemukan kualitas koneksi internet perangkat *client*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | **class** DownloadImage inherits AsyncTask<String, Void, Void>  @Override  **function** onPreExecute()  mDeviceBandwidthSampler.startSampling()  @Override  **function** doInBackground(String... url)  String imageURL := url[0]  **try**  URLConnection connection := new  URL(imageURL).openConnection()  connection.setUseCaches(false)  connection.connect()  InputStream input :=  connection.getInputStream()  **try**  [] buffer := [0..1023]  while (input.read(buffer) != -1)  **finally**  input.close()  **catch** (IOException e)  Log.e(TAG, "Error while downloading  image.")  return NIL  @Override  **function** onPostExecute(Void v)  mDeviceBandwidthSampler.stopSampling()  **if** (mConnectionClass ==  ConnectionQuality.UNKNOWN AND mTries <  10)  mTries++  new DownloadImage().execute(mURL)  **if** (NOT  mDeviceBandwidthSampler.isSampling())  Log.e(TAG, "Finish Check Connection")  mTries := 0  Log.e(TAG, "+" + mConnectionClass  Manager.getCurrentBandwidth  Quality().toString())  **if** (mConnectionClassManager.  getCurrentBandwidthQuality() !=  ConnectionQuality.UNKNOWN)  Log.e(TAG, "OK") |

Pseudocode 4.33 Kode program inisialisasi penerapan *Asynchronous Task* pada pengecekan buffer data

#### Kondisi Level Baterai Perangkat Bergerak

*Pseudocode* 4.34 memperlihatkan fungsi untuk mendapatkan level baterai dari perangkat *client*. Nilai level akan berubah sesuai kondisi level baterai perangkat *client* jika fungsi ini dipanggil saat adanya perubahan pada level baterai perangkat *client*.

|  |  |
| --- | --- |
| 1  2  3  4  5 | **function** getBatteryPercent()  IntentFilter ifilter := new IntentFilter  (Intent.ACTION\_BATTERY\_CHANGED)  Intent batteryStatus := getBaseContext().  registerReceiver(NIL, ifilter)  level := batteryStatus.getIntExtra  (BatteryManager.EXTRA\_LEVEL, -1)  return level |

*Pseudocode* 4.34 Kode program mendapatkan nilai level baterai pada perangkat *client*

#### Waktu Eksekusi Proses *Image Recognition*

*Pseudocode* 4.35 memperlihatkan fungsi untuk memulai perhitungan waktu saat dimulainya eksekusi beban kerja *image recognition* baik secara lokal maupun *offloading* yang dapat dilihat pada pemanggilan fungsi startTimeTask() pada *Pseudocode* 4.13.

|  |  |
| --- | --- |
| 1  2 | **function** startTimeTask()  startTime := SystemClock.  elapsedRealtime() |

*Pseudocode* 4.35 Kode program perhitungan waktu dimulainya beban kerja dieksekusi

*Pseudocode* 4.36 memperlihatkan fungsi untuk mengakhiri perhitungan saat selesainya eksekusi beban kerja *image recognition* secara *offloading* yang dapat dilihat pada pemanggilan fungsi endTimeTaskOffload() pada *Pseudocode* 4.12. Hasil waktu eksekusi secara *offloading* selanjutnya akan di rata – rata setiap dilakukannya eksekusi beban kerja secara *offloading* dengan tujuan untuk dipergunakan oleh *Decision Maker*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **function** endTimeTaskOffload()  endTime :=  SystemClock.elapsedRealtime()  elapsedMilliSeconds := endTime -  startTime  elapsedSeconds := elapsedMilliSeconds  / 1000.0  Log.i(TAG, "Time Elapsed Offload " +  currentFileTrain + " : " +  elapsedSeconds + " seconds")  TaskCountOffload++  rateExecutionTimeOffload :=  ((rateExecutionTimeOffload \*  (TaskCountOffload - 1)) +  elapsedSeconds) / TaskCountOffload  Log.i(TAG, "Time Elapsed Rate  Offload: " + rateExecutionTimeOffload + "  seconds") |

*Pseudocode* 4.36 Kode program perhitungan waktu selesainya beban kerja dieksekusi secara *offloading*

*Pseudocode* 4.37 memperlihatkan fungsi untuk mengakhiri perhitungan saat selesainya eksekusi beban kerja *image recognition* secara lokalyang dapat dilihat pada pemanggilan fungsi endTimeTaskOffload() pada *Pseudocode* 4.37. Hasil waktu eksekusi secara lokalselanjutnya akan di rata – rata setiap dilakukannya eksekusi beban kerja secara lokaldengan tujuan untuk dipergunakan oleh *Decision Maker*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **function** endTimeTaskLocal()  endTime :=  SystemClock.elapsedRealtime()  elapsedMilliSeconds := endTime –  startTime  elapsedSeconds := elapsedMilliSeconds /  1000.0  Log.i(TAG, "Time Elapsed Local " +  currentFileTrain + " : " +  elapsedSeconds + " seconds")  TaskCountLocal++  rateExecutionTimeLocal :=  ((rateExecutionTimeLocal \*  (TaskCountLocal - 1)) + elapsedSeconds)  / TaskCountLocal  Log.i(TAG, "Time Elapsed Rate Local: " +  rateExecutionTimeLocal + " seconds") |

*Pseudocode* 4.37 Kode program perhitungan waktu selesainya beban kerja dieksekusi secara lokal

# BAB V HASIL UJI COBA DAN EVALUASI

Bab ini berisi penjelasan mengenai skenario uji coba dan evaluasi penentuan keputusan eksekusi beban kerja pada *offloading computation* *framework* pada perangkat lunak Android. Hasil uji coba didapatkan dari implementasi pada Bab 4 dengan skenario yang berbeda. Bab ini berisikan pembahasan mengenai lingkungan pengujian, data pengujian, dan uji kinerja.

## Lingkungan Pengujian

Lingkungan pengujian pada uji coba permasalahan penentuan keputusan eksekusi beban kerja oleh *framework* melalui faktor – faktor penentu metode *offloading* menggunakan spesifikasi keras dan perangkat lunak seperti yang ditunjukkan pada Tabel 5.1.

Tabel 5.1 Spesifikasi Lingkungan Pengujian

|  |  |  |  |
| --- | --- | --- | --- |
| *Privilege* | Perangkat | Jenis Perangkat | Spesifikasi |
| *Client* | Perangkat Keras | Prosesor | Intel(R) Atom(TM) CPU Z2580 2,00 GHz |
| Memori | 2 GB DDR3 |
| Perangkat Lunak | Sistem Operasi | Android 4.4.2 KitKat |
| *Server* | Perangkat Keras | Prosesor | Intel(R) Core(TM) i7-2.5 GHz |
| Memori | 8 GB 800 MHz DDR3 |
| Perangkat Lunak | Sistem Operasi | Windows 10 dan Ubuntu 16.04 |
| Perangkat Pengembang | Android Studio IDE dan Netbeans IDE 8.2 |

## Data Pengujian

Subbab ini menjelaskan mengenai data yang digunakan pada uji coba. Seperti yang telah dijelaskan sebelumnya, terdiri dari dua jenis data masukan. Data masukan jenis pertama diperoleh dengan menggunakan kamera yang terpasang di perangkat Android dan dilakukan pengambilan gambar saat menjalankan perangkat lunak Android. Data masukan jenis kedua adalah sebuah dataset terdiri dari 40 buah citra daun yang dapat dibagi 10 citra daun berdasarkan jenis daunnya sehingga setiap jenis daun memiliki 4 citra daun.

Kedua data tersebut akan diolah menggunakan kompresi pada untuk citra datatest dan pewarnaan *grayscale* untuk kedua citra datatest dan dataset sehingga menghasilkan citra dengan matriks yang lebih baik jika akan diambil masing – masing *descriptor* matriksnya. Data hasil *preprossesing* tersebut akan dihitung kemiripannya menggunakan metode SURF untuk proses *image recognition*. Selanjutnya dilakukan pengecekan terhadap perhitungan kemiripan dengan melakukan seleksi jarak kedekatan titik fitur yang didapatkan sesuai kriteria yang ditentukan dengan dasar nilai terkecil (mendekati nol) memiliki kemiripan yang lebih besar.

## *Preprocessing* citra

*Preprocessing* citra yang digunakan dalam skenario uji coba adalah pada tahap kompresi citra dan pewarnaan *grayscale* yaitu tahap untuk menghasilkan citra dengan matriks yang lebih baik jika akan diambil masing – masing *descriptor* matriksnya untuk tahap *image recognition*.

## Skenario Uji Coba

Sebelum melakukan uji coba, perlu ditentukan skenario yang akan digunakan dalam uji coba. Melalui skenario ini, perangkat akan diuji apakah sudah berjalan dengan benar dan bagaimana performa pada masing-masing skenario. Dan membandingkan skenario manakah yang memiliki hasil lebih baik.

Di dalam skenario uji coba, terdapat kondisi koneksi internet dan level baterai pada perangkat *client* yang digunakan dalam menentukan metode pengeksekusian beban kerja. Kondisi koneksi internet dan level baterai akan dikategorikan ke dalam syarat tertentu untuk keperluan uji coba.

Kondisi koneksi internet akan dikategorikan menjadi 3 kategori yang berbeda yaitu Koneksi A, Koneksi B, dan Koneksi C. Pembagian kategori ini didasarkan pada nilai kualitas koneksi internet perangkat *client* seperti yang telah dijelaskan pada Bab 3.4.1. Pada Koneksi C, untuk mendapatkan nilai kualitas *unkown* pada uji coba dilakukan dua kali penonaktifan koneksi internet pada saat perangkat *client* melakukan proses *image recognition*. Untuk lebih jelasnya mengenai pembagian kategori kondisi koneksi internet akan dijelaskan pada Tabel 5.2

Tabel 5.2 Pembagian kategori koneksi internet

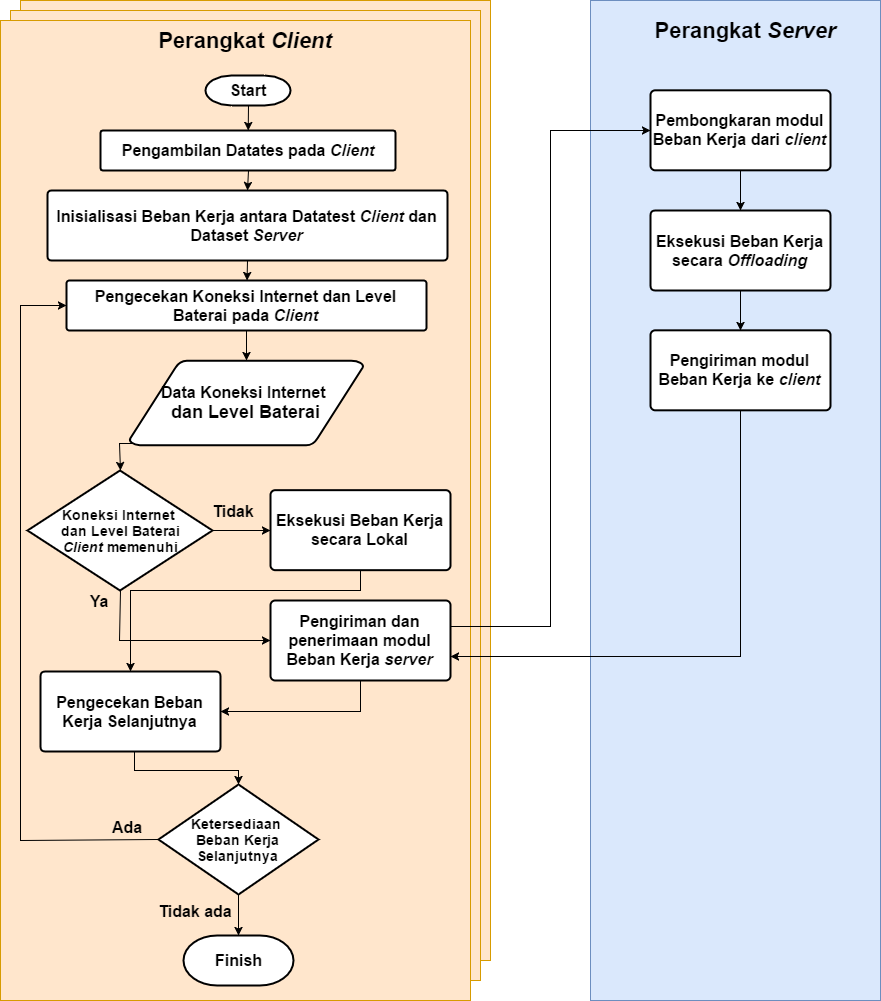
|  |  |
| --- | --- |
| Kategori | Nilai Kualitas |
| Koneksi A | *Excellent*, *Good* dan *Moderate* |
| Koneksi B | *Poor* |
| Koneksi C | *Unknown* |

Kondisi level baterai akan dikategorikan menjadi 2 kategori yang berbeda yaitu Baterai A dan Baterai B. Pembagian kategori ini didasarkan pada nilai level baterai perangkat *client* seperti yang telah dijelaskan pada Bab 3.4.2. Untuk lebih jelasnya mengenai pembagian kategori kondisi level baterai akan dijelaskan pada Tabel 5.3.

Tabel 5.3 Pembagian kategori level baterai

|  |  |
| --- | --- |
| Kategori | Nilai level |
| Baterai A | 21% - 100% |
| Baterai B | 0% - 20% |

Dalam menjalankan eksekusi beban kerja proses *image recognition*, *framework* akan melakukan urutan langkah – langkah kerja yang telah dijelaskan sebelumnya pada Bab 3.5. Pada skenario uji coba kali ini akan digabungkan antara langkah – langkah kerja dari *framework* dengan kondisi koneksi internet dan level baterai seperti yang telah dijelaskan sebelumnya. Pada Gambar 5.1 akan dijelaskan bagaimana langkah – langkah kerja *framework* dan kondisi koneksi internet dan level baterai dilakukan.



Gambar 5.1 Bagan alur kerja skenario uji coba

Terdapat 6 macam skenario uji coba, yaitu:

1. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan level Baterai A.
2. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A.
3. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B.
4. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai B.
5. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A.
6. Perhitungan penghematan daya, performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B.

### Skenario Uji Coba 1

Skenario uji coba 1 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai A. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 1 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.6. Hasil waktu eksekusi pada uji coba 1 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.7.

Tabel 5.4 Performa tanpa penerapan *framework* pada perangkat Android

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. Cek | Penggunaan Memori | | Penggunaan CPU | Bandwidth Internet | Level Baterai | Suhu perangkat |
| (5s) | (MB) | (%) | (%) | (KB) | (%) | (C) |
| 1 | 37.47 | 1.89 | 0 | 0 | 60 | 33.7 |
| 2 | 35.5 | 1.79 | 7.84 | 0 | 60 | 33.7 |
| 3 | 78.97 | 3.98 | 34.25 | 0 | 60 | 33.7 |
| 4 | 71.71 | 3.61 | 60.61 | 0 | 60 | 33.7 |
| 5 | 47.65 | 2.4 | 74.58 | 0 | 60 | 33.7 |
| 6 | 53.33 | 2.69 | 67.71 | 0 | 60 | 33.7 |
| 7 | 52.65 | 2.65 | 57.85 | 0 | 60 | 33.7 |
| 8 | 54.04 | 2.72 | 66.8 | 0 | 60 | 33.7 |
| 9 | 54.41 | 2.74 | 58.37 | 0 | 60 | 33.7 |
| 10 | 53.33 | 2.69 | 85.36 | 0 | 60 | 33.7 |
| 11 | 54.05 | 2.72 | 67.94 | 0 | 60 | 33.7 |
| 12 | 77.6 | 3.91 | 64.06 | 0 | 60 | 33.7 |
| 13 | 53.2 | 2.68 | 82.94 | 0 | 59 | 34.1 |
| 14 | 53.86 | 2.71 | 66.14 | 0 | 59 | 34.1 |
| 15 | 53.22 | 2.68 | 61.4 | 0 | 59 | 34.1 |
| 16 | 53.08 | 2.67 | 59.84 | 0 | 59 | 34.1 |
| 17 | 71.44 | 3.6 | 61.77 | 0 | 59 | 34.1 |
| 18 | 59.66 | 3.01 | 63.41 | 0 | 59 | 34.1 |
| 19 | 64.45 | 3.25 | 61.83 | 0 | 59 | 34.1 |
| 20 | 54.52 | 2.75 | 80.63 | 0 | 59 | 34.1 |
| 21 | 53.31 | 2.69 | 62.09 | 0 | 59 | 34.1 |
| 22 | 54.2 | 2.73 | 61.88 | 0 | 59 | 34.1 |
| 23 | 53.59 | 2.7 | 60.3 | 0 | 59 | 34.1 |
| 24 | 55.05 | 2.77 | 56.87 | 0 | 59 | 34.9 |
| 25 | 53.6 | 2.7 | 59.44 | 0 | 59 | 34.9 |
| 26 | 52.98 | 2.67 | 65.62 | 0 | 59 | 34.9 |
| 27 | 53.18 | 2.68 | 74.61 | 0 | 59 | 34.9 |
| 28 | 53.09 | 2.68 | 60.91 | 0 | 59 | 34.9 |
| 29 | 52.45 | 2.64 | 59.58 | 0 | 59 | 34.9 |
| 30 | 46.83 | 2.36 | 66.31 | 0 | 59 | 34.9 |
| 31 | 50.66 | 2.55 | 86.47 | 0 | 59 | 34.9 |
| 32 | 53.97 | 2.72 | 61.49 | 0 | 59 | 34.9 |
| 33 | 82.91 | 4.18 | 64.88 | 0 | 59 | 34.9 |
| 34 | 51.71 | 2.61 | 73.9 | 0 | 59 | 34.9 |
| 35 | 54.38 | 2.74 | 71.35 | 0 | 59 | 34.9 |
| 36 | 53.35 | 2.69 | 61.14 | 0 | 58 | 36.2 |
| 37 | 52.44 | 2.64 | 60.86 | 0 | 58 | 36.2 |
| 38 | 47.31 | 2.38 | 73.74 | 0 | 58 | 36.2 |
| 39 | 53.62 | 2.7 | 78.8 | 0 | 58 | 36.2 |
| 40 | 53.93 | 2.72 | 56.12 | 0 | 58 | 36.2 |
| 41 | 52.13 | 2.63 | 54 | 0 | 58 | 36.2 |
| 42 | 51.38 | 2.59 | 52.15 | 0 | 58 | 36.2 |
| 43 | 52.38 | 2.64 | 50.9 | 0 | 58 | 36.2 |
| 44 | 66.46 | 3.35 | 56.17 | 0 | 58 | 36.2 |
| 45 | 52.08 | 2.62 | 61.87 | 0 | 58 | 36.2 |
| 46 | 52.98 | 2.67 | 60.19 | 0 | 58 | 36.2 |
| 47 | 60.49 | 3.05 | 61.79 | 0 | 57 | 37.5 |
| 48 | 52.58 | 2.65 | 83.17 | 0 | 57 | 37.5 |
| 49 | 53.58 | 2.7 | 67.52 | 0 | 57 | 37.5 |
| 50 | 71.75 | 3.62 | 59.81 | 0 | 57 | 37.5 |
| 51 | 47.26 | 2.38 | 75.3 | 0 | 57 | 37.5 |
| 52 | 51.56 | 2.6 | 71.96 | 0 | 57 | 37.5 |
| 53 | 53.93 | 2.72 | 61.11 | 0 | 57 | 37.5 |
| 54 | 47.75 | 2.41 | 12.86 | 0 | 57 | 37.5 |
| Rata2 | 55.24 | 2.784 | 61.639 | 0 | - | - |

Gambar 5.2 Grafik performa tanpa penerapan *framework* pada perangkat Android

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* mempunyai penggunaan memori rata – rata 55.24 MB atau 2.784% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 61.639% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 0 KB, penurunan level baterai mencapai 3% dan kenaikan suhu pada perangkat mencapai 3.8 C.

Tabel 5.5 Waktu eksekusi tanpa penerapan *framework* pada perangkat lunak *image recognition*

|  |  |  |  |
| --- | --- | --- | --- |
| No. Eksekusi | Metode | Waktu (s) | Kualitas Koneksi |
| 1 | Lokal | 7.695 | - |
| 2 | Lokal | 5.834 | - |
| 3 | Lokal | 3.739 | - |
| 4 | Lokal | 5.822 | - |
| 5 | Lokal | 8.575 | - |
| 6 | Lokal | 8.098 | - |
| 7 | Lokal | 9.957 | - |
| 8 | Lokal | 6.762 | - |
| 9 | Lokal | 4.974 | - |
| 10 | Lokal | 5.435 | - |
| 11 | Lokal | 5.094 | - |
| 12 | Lokal | 4.684 | - |
| 13 | Lokal | 6.398 | - |
| 14 | Lokal | 4.424 | - |
| 15 | Lokal | 5.338 | - |
| 16 | Lokal | 3.809 | - |
| 17 | Lokal | 6.05 | - |
| 18 | Lokal | 6.242 | - |
| 19 | Lokal | 5.383 | - |
| 20 | Lokal | 4.923 | - |
| 21 | Lokal | 7.603 | - |
| 22 | Lokal | 6.998 | - |
| 23 | Lokal | 7.092 | - |
| 24 | Lokal | 6.279 | - |
| 25 | Lokal | 5.238 | - |
| 26 | Lokal | 7.872 | - |
| 27 | Lokal | 6.686 | - |
| 28 | Lokal | 7.818 | - |
| 29 | Lokal | 3.106 | - |
| 30 | Lokal | 3.517 | - |
| 31 | Lokal | 3.93 | - |
| 32 | Lokal | 3.666 | - |
| 33 | Lokal | 3.563 | - |
| 34 | Lokal | 4.756 | - |
| 35 | Lokal | 4.575 | - |
| 36 | Lokal | 4.429 | - |
| 37 | Lokal | 7.137 | - |
| 38 | Lokal | 7.838 | - |
| 39 | Lokal | 5.776 | - |
| 40 | Lokal | 8.5 | - |
| Rata - rata | = | 5.89 |  |

Gambar 5.3 Grafik waktu eksekusi tanpa penerapan *framework* pada perangkat lunak *image recognition*

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dilakukan secara lokal mencapai 100% dan mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 5.89 detik.

Tabel 5.6 Performa dengan penerapan *framework* pada Koneksi A dan Baterai A

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. Cek | Penggunaan Memori | | Penggunaan CPU | Bandwidth Internet | Level Baterai | Suhu perangkat |
| (5s) | (MB) | (%) | (%) | (KB) | (%) | (C) |
| 1 | 39.45 | 1.99 | 0 | 0 | 68 | 40.1 |
| 2 | 42.46 | 2.14 | 11.73 | 412 | 68 | 40.1 |
| 3 | 47.62 | 2.4 | 46.46 | 947 | 68 | 40.1 |
| 4 | 53.1 | 2.68 | 67.22 | 947 | 68 | 40.1 |
| 5 | 51.78 | 2.61 | 58.76 | 1309 | 68 | 40.1 |
| 6 | 50.56 | 2.55 | 22 | 2332 | 68 | 40.1 |
| 7 | 52.82 | 2.66 | 22.39 | 3340 | 68 | 40.1 |
| 8 | 53.24 | 2.68 | 20.77 | 4158 | 68 | 40.1 |
| 9 | 50.77 | 2.56 | 20.64 | 4923 | 68 | 40.1 |
| 10 | 52.11 | 2.63 | 19.87 | 5506 | 68 | 40.1 |
| 11 | 51.65 | 2.6 | 21.31 | 6223 | 68 | 40.1 |
| 12 | 52.07 | 2.62 | 22.2 | 6864 | 68 | 40.1 |
| 13 | 50.87 | 2.56 | 20.4 | 7700 | 68 | 40.6 |
| 14 | 52.48 | 2.64 | 20.5 | 8413 | 68 | 40.6 |
| 15 | 52.79 | 2.66 | 20.25 | 8995 | 68 | 40.6 |
| 16 | 48 | 2.42 | 20.85 | 9577 | 68 | 40.6 |
| 17 | 48.36 | 2.44 | 21.85 | 10552 | 68 | 40.6 |
| 18 | 51.1 | 2.58 | 19.96 | 11318 | 68 | 40.6 |
| 19 | 49.45 | 2.49 | 19.53 | 11901 | 68 | 40.6 |
| 20 | 49.04 | 2.47 | 21.74 | 12616 | 68 | 40.6 |
| 21 | 51.43 | 2.59 | 21.14 | 13287 | 68 | 40.6 |
| 22 | 49.55 | 2.5 | 19.31 | 14146 | 68 | 40.6 |
| 23 | 47.73 | 2.41 | 20.58 | 14806 | 68 | 40.6 |
| 24 | 50.39 | 2.54 | 21.41 | 15388 | 67 | 41.9 |
| 25 | 50.67 | 2.55 | 22.04 | 16197 | 67 | 41.9 |
| 26 | 51.12 | 2.58 | 20.31 | 17024 | 67 | 41.9 |
| 27 | 49.51 | 2.5 | 20.87 | 17710 | 67 | 41.9 |
| 28 | 49.23 | 2.48 | 20.73 | 18426 | 67 | 41.9 |
| 29 | 51.79 | 2.61 | 21.58 | 19007 | 67 | 41.9 |
| 30 | 51.2 | 2.58 | 21.18 | 19875 | 67 | 41.9 |
| 31 | 49.73 | 2.51 | 20.71 | 20614 | 67 | 41.9 |
| 32 | 51.31 | 2.59 | 19.98 | 21335 | 67 | 41.9 |
| 33 | 51.63 | 2.6 | 21.73 | 21917 | 67 | 41.9 |
| 34 | 51.56 | 2.6 | 21.19 | 22785 | 67 | 41.9 |
| 35 | 51.59 | 2.6 | 20.56 | 23525 | 67 | 41.9 |
| 36 | 48.91 | 2.46 | 20.06 | 24107 | 66 | 42.6 |
| 37 | 49.87 | 2.51 | 21.58 | 24822 | 66 | 42.6 |
| 38 | 46.84 | 2.36 | 13.41 | 24822 | 66 | 42.6 |
| Rata2 | 50.09 | 2.525 | 43.39 | 12311.2 | - | - |

Gambar 5.4 Grafik performa dengan penerapan *framework* pada Koneksi A dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai A mempunyai penggunaan memori rata – rata 50.09 MB atau 2.525% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 43.39% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 12311.2 KB, penurunan level baterai mencapai 2% dan kenaikan suhu pada perangkat mencapai 2.5 C.

Tabel 5.7 Waktu eksekusi dengan penerapan *framework* pada Koneksi A dan Baterai A

|  |  |  |  |
| --- | --- | --- | --- |
| No. Eksekusi | Metode | Waktu (s) | Kualitas Koneksi |
| 1 | Lokal | 7.666 | *GOOD* |
| 2 | Offloading | 3.319 | *GOOD* |
| 3 | Offloading | 3.395 | *GOOD* |
| 4 | Offloading | 3.334 | *EXCELLENT* |
| 5 | Offloading | 3.355 | *EXCELLENT* |
| 6 | Offloading | 3.418 | *EXCELLENT* |
| 7 | Offloading | 3.314 | *GOOD* |
| 8 | Offloading | 2.742 | *GOOD* |
| 9 | Offloading | 3.442 | *GOOD* |
| 10 | Offloading | 3.347 | *GOOD* |
| 11 | Offloading | 3.348 | *EXCELLENT* |
| 12 | Offloading | 3.522 | *EXCELLENT* |
| 13 | Offloading | 3.36 | *EXCELLENT* |
| 14 | Offloading | 3.329 | *EXCELLENT* |
| 15 | Offloading | 3.337 | *EXCELLENT* |
| 16 | Offloading | 3.287 | *EXCELLENT* |
| 17 | Offloading | 3.346 | *EXCELLENT* |
| 18 | Offloading | 3.357 | *EXCELLENT* |
| 19 | Offloading | 3.349 | *EXCELLENT* |
| 20 | Offloading | 3.308 | *EXCELLENT* |
| 21 | Offloading | 3.396 | *EXCELLENT* |
| 22 | Offloading | 3.625 | *EXCELLENT* |
| 23 | Offloading | 3.351 | *GOOD* |
| 24 | Offloading | 3.39 | *GOOD* |
| 25 | Offloading | 3.476 | *GOOD* |
| 26 | Offloading | 3.321 | *EXCELLENT* |
| 27 | Offloading | 3.373 | *GOOD* |
| 28 | Offloading | 3.32 | *GOOD* |
| 29 | Offloading | 3.328 | *GOOD* |
| 30 | Offloading | 3.349 | *GOOD* |
| 31 | Offloading | 3.312 | *GOOD* |
| 32 | Offloading | 3.34 | *GOOD* |
| 33 | Offloading | 3.348 | *GOOD* |
| 34 | Offloading | 3.246 | *GOOD* |
| 35 | Offloading | 3.323 | *GOOD* |
| 36 | Offloading | 3.434 | *GOOD* |
| 37 | Offloading | 3.331 | *GOOD* |
| 38 | Offloading | 3.315 | *GOOD* |
| 39 | Offloading | 3.601 | *GOOD* |
| 40 | Offloading | 3.41 | *GOOD* |
| Rata - rata | = | 3.4616 |  |

Gambar 5.5 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi A dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai A dilakukan secara lokal mencapai 2.5% dan secara *offloading* mencapai 97.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 3.4616 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai A akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 50.09 MB atau 2.525%. Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 43.39%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 2%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar 5.16 MB atau 0.259%, penggunaan CPU rata – rata sebesar 18.249% dan penggunaan level baterai mencapai 1%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 3.4616 detik sehingga dihasilkan penghematan waktu eksekusi mencapai 2.4284 detik setiap eksekusi beban kerjanya.

### Skenario Uji Coba 2

Skenario uji coba 2 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 2 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.8. Hasil waktu eksekusi pada uji coba 2 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.9.

Tabel 5.8 Performa dengan penerapan *framework* pada Koneksi B dan Baterai A

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. Cek | Penggunaan Memori | | Penggunaan CPU | Bandwidth Internet | Level Baterai | Suhu perangkat |
| (5s) | (MB) | (%) | (%) | (KB) | (%) | (C) |
| 1 | 41.47 | 2.09 | 0 | 0 | 56 | 38.4 |
| 2 | 40.38 | 2.04 | 11.51 | 43 | 56 | 38.4 |
| 3 | 39.55 | 1.99 | 17.62 | 147 | 56 | 38.4 |
| 4 | 39.12 | 1.97 | 18.15 | 249 | 56 | 38.4 |
| 5 | 47.4 | 2.39 | 52.27 | 295 | 56 | 38.4 |
| 6 | 53.02 | 2.67 | 70.86 | 295 | 56 | 38.4 |
| 7 | 55.72 | 2.81 | 56.8 | 341 | 56 | 38.4 |
| 8 | 64.38 | 3.24 | 61.01 | 477 | 56 | 39.2 |
| 9 | 76.91 | 3.88 | 57.47 | 592 | 56 | 39.2 |
| 10 | 53.8 | 2.71 | 71.68 | 712 | 56 | 39.2 |
| 11 | 56.37 | 2.84 | 74.81 | 841 | 56 | 39.2 |
| 12 | 56.22 | 2.83 | 56.73 | 911 | 56 | 39.2 |
| 13 | 55.77 | 2.81 | 85.12 | 1016 | 56 | 39.2 |
| 14 | 55.28 | 2.79 | 60.58 | 1128 | 56 | 39.2 |
| 15 | 49.39 | 2.49 | 68.54 | 1242 | 56 | 39.2 |
| 16 | 55.73 | 2.81 | 77.42 | 1344 | 56 | 39.2 |
| 17 | 55.87 | 2.82 | 58.36 | 1470 | 56 | 39.2 |
| 18 | 53.07 | 2.67 | 34.34 | 1572 | 55 | 39.2 |
| 19 | 54.83 | 2.76 | 18.05 | 1690 | 55 | 41.9 |
| 20 | 38.27 | 1.93 | 13.27 | 1817 | 55 | 41.9 |
| 21 | 51.97 | 2.62 | 16.14 | 1931 | 55 | 41.9 |
| 22 | 51.97 | 2.62 | 17.69 | 2016 | 55 | 41.9 |
| 23 | 51.35 | 2.59 | 18.56 | 2135 | 55 | 41.9 |
| 24 | 51.54 | 2.6 | 18.2 | 2253 | 55 | 41.9 |
| 25 | 51.41 | 2.59 | 17.35 | 2305 | 55 | 41.9 |
| 26 | 51.44 | 2.59 | 17.7 | 2386 | 55 | 41.9 |
| 27 | 51.41 | 2.59 | 18.02 | 2506 | 55 | 41.9 |
| 28 | 51.34 | 2.59 | 17.8 | 2637 | 55 | 41.9 |
| 29 | 67.53 | 3.4 | 22.74 | 2727 | 55 | 41.9 |
| 30 | 55.13 | 2.78 | 63.58 | 2736 | 55 | 41.9 |
| 31 | 55.14 | 2.78 | 61.31 | 2857 | 55 | 43.1 |
| 32 | 55.14 | 2.78 | 72.67 | 2967 | 55 | 43.1 |
| 33 | 55.01 | 2.77 | 53.4 | 3065 | 55 | 43.1 |
| 34 | 53.73 | 2.71 | 58.58 | 3115 | 55 | 43.1 |
| 35 | 76.1 | 3.83 | 62.21 | 3156 | 55 | 43.1 |
| 36 | 53.77 | 2.71 | 58.26 | 3229 | 55 | 43.1 |
| 37 | 65.46 | 3.3 | 62.49 | 3317 | 54 | 43.1 |
| 38 | 50.41 | 2.54 | 75.23 | 3414 | 54 | 43.1 |
| 39 | 52.05 | 2.62 | 47.01 | 3484 | 54 | 43.1 |
| 40 | 68.72 | 3.46 | 52.29 | 3579 | 54 | 43.1 |
| 41 | 46.44 | 2.34 | 59.44 | 3701 | 54 | 43.1 |
| 42 | 52.48 | 2.64 | 80.06 | 3828 | 54 | 44.5 |
| 43 | 52.24 | 2.63 | 56.38 | 3950 | 54 | 44.5 |
| 44 | 46.3 | 2.33 | 70.34 | 4023 | 54 | 44.5 |
| 45 | 52.43 | 2.64 | 77.8 | 4143 | 54 | 44.5 |
| 46 | 52.45 | 2.64 | 54.26 | 4255 | 54 | 44.5 |
| 47 | 51.13 | 2.58 | 61.79 | 4354 | 54 | 44.5 |
| 48 | 51.18 | 2.58 | 71.22 | 4474 | 54 | 44.5 |
| 49 | 52.74 | 2.66 | 66.54 | 4591 | 54 | 44.5 |
| 50 | 52.51 | 2.65 | 56.34 | 4713 | 54 | 44.5 |
| 51 | 51.45 | 2.59 | 69.91 | 4812 | 54 | 44.5 |
| 52 | 52.72 | 2.66 | 74.28 | 4913 | 54 | 44.5 |
| 53 | 52.55 | 2.65 | 55.54 | 4974 | 54 | 44.5 |
| 54 | 51.23 | 2.58 | 55 | 4983 | 54 | 45.7 |
| 55 | 51.25 | 2.58 | 50.2 | 5072 | 54 | 45.7 |
| 56 | 46.62 | 2.35 | 49.68 | 5148 | 54 | 45.7 |
| 57 | 49.27 | 2.48 | 51.55 | 5242 | 53 | 45.7 |
| 58 | 51.25 | 2.58 | 65.75 | 5314 | 53 | 45.7 |
| 59 | 51.1 | 2.58 | 59.6 | 5394 | 53 | 45.7 |
| 60 | 51.1 | 2.58 | 59.36 | 5476 | 53 | 45.7 |
| 61 | 52.77 | 2.66 | 71.34 | 5584 | 53 | 45.7 |
| 62 | 52.6 | 2.65 | 57.03 | 5680 | 53 | 45.7 |
| 63 | 51.56 | 2.6 | 78.23 | 5811 | 53 | 45.7 |
| 64 | 52.93 | 2.67 | 71.67 | 5941 | 53 | 45.7 |
| 65 | 52.62 | 2.65 | 56.92 | 6069 | 53 | 45.7 |
| 66 | 51.65 | 2.6 | 83.3 | 6172 | 53 | 46.5 |
| 67 | 43.62 | 2.2 | 15.89 | 6231 | 53 | 46.5 |
| Rata2 | 52.826 | 2.662 | 51.869 | 3087.24 | - | - |

Gambar 5.6 Grafik performa dengan penerapan *framework* pada Koneksi B dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A. mempunyai penggunaan memori rata – rata 52.826 MB atau 2.662% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 51.869% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 3087.24 KB, penurunan level baterai mencapai 3% dan kenaikan suhu pada perangkat mencapai 8.1 C.

Tabel 5.9 Waktu eksekusi dengan penerapan *framework* pada Koneksi B dan Baterai A

|  |  |  |  |
| --- | --- | --- | --- |
| No. Eksekusi | Metode | Waktu (s) | Kualitas Koneksi |
| 1 | Lokal | 6.438 | *POOR* |
| 2 | Lokal | 5.211 | *POOR* |
| 3 | Lokal | 3.806 | *POOR* |
| 4 | Lokal | 5.894 | *POOR* |
| 5 | Lokal | 8.506 | *POOR* |
| 6 | Lokal | 7.749 | *POOR* |
| 7 | Lokal | 6.777 | *POOR* |
| 8 | Lokal | 6.606 | *POOR* |
| 9 | Offloading | 57.738 | *MODERATE* |
| 10 | Lokal | 4.428 | *POOR* |
| 11 | Lokal | 5.143 | *POOR* |
| 12 | Lokal | 4.626 | *POOR* |
| 13 | Lokal | 6.677 | *POOR* |
| 14 | Lokal | 4.729 | *POOR* |
| 15 | Lokal | 5.504 | *POOR* |
| 16 | Lokal | 3.953 | *POOR* |
| 17 | Lokal | 5.173 | *POOR* |
| 18 | Lokal | 6.777 | *POOR* |
| 19 | Lokal | 7.057 | *POOR* |
| 20 | Lokal | 5.052 | *POOR* |
| 21 | Lokal | 7.404 | *POOR* |
| 22 | Lokal | 6.57 | *POOR* |
| 23 | Lokal | 6.821 | *POOR* |
| 24 | Lokal | 6.889 | *POOR* |
| 25 | Lokal | 5.407 | *POOR* |
| 26 | Lokal | 6.009 | *MODERATE* |
| 27 | Lokal | 7.263 | *MODERATE* |
| 28 | Lokal | 6.958 | *MODERATE* |
| 29 | Lokal | 3.214 | *POOR* |
| 30 | Lokal | 3.188 | *POOR* |
| 31 | Lokal | 3.696 | *POOR* |
| 32 | Lokal | 3.37 | *POOR* |
| 33 | Lokal | 4.158 | *POOR* |
| 34 | Lokal | 4.89 | *POOR* |
| 35 | Lokal | 4.167 | *POOR* |
| 36 | Lokal | 4.216 | *POOR* |
| 37 | Lokal | 6.547 | *POOR* |
| 38 | Lokal | 7.593 | *POOR* |
| 39 | Lokal | 6.048 | *POOR* |
| 40 | Lokal | 8.483 | *POOR* |
| Rata - rata | = | 7.018 |  |

Gambar 5.7 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi B dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A dilakukan secara lokal mencapai 97.5% dan secara *offloading* mencapai 2.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 7.018 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai A akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 52.826 MB atau 2.662%. Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 51.869%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 3%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar 2.414 MB atau 0.122%, penggunaan CPU rata – rata sebesar 9.77% dan penggunaan level baterai mencapai 0%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 7.018 detik sehingga dihasilkan penghematan waktu eksekusi mencapai -1.128 detik setiap eksekusi beban kerjanya.

### Skenario Uji Coba 3

Skenario uji coba 3 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 3 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.10. Hasil waktu eksekusi pada uji coba 3 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.11.

Tabel 5.10 Performa dengan penerapan *framework* pada Koneksi A dan Baterai B

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. Cek | Penggunaan Memori | | Penggunaan CPU | Bandwidth Internet | Level Baterai | Suhu perangkat |
| (5s) | (MB) | (%) | (%) | (KB) | (%) | (C) |
| 1 | 40.28 | 2.03 | 0 | 140 | 19 | 37.3 |
| 2 | 39.97 | 2.01 | 15.12 | 2280 | 19 | 37.3 |
| 3 | 48.65 | 2.45 | 36.13 | 2957 | 19 | 37.3 |
| 4 | 54.41 | 2.74 | 69.55 | 2957 | 19 | 37.3 |
| 5 | 52.94 | 2.67 | 61.53 | 3229 | 19 | 37.3 |
| 6 | 57.52 | 2.9 | 20.68 | 4517 | 19 | 38.3 |
| 7 | 57.88 | 2.92 | 17.78 | 5255 | 19 | 38.3 |
| 8 | 57.94 | 2.92 | 18.97 | 5977 | 19 | 38.3 |
| 9 | 57.85 | 2.92 | 18.72 | 6680 | 19 | 38.3 |
| 10 | 57.96 | 2.92 | 18.53 | 7342 | 19 | 38.3 |
| 11 | 58.23 | 2.93 | 18.13 | 7970 | 19 | 38.3 |
| 12 | 56.86 | 2.87 | 19.13 | 8739 | 18 | 38.3 |
| 13 | 58.92 | 2.97 | 17.86 | 9361 | 18 | 38.3 |
| 14 | 57.29 | 2.89 | 19.13 | 10036 | 18 | 38.3 |
| 15 | 58.4 | 2.94 | 19.15 | 11010 | 18 | 38.3 |
| 16 | 58.42 | 2.94 | 18.47 | 11536 | 18 | 38.3 |
| 17 | 58.53 | 2.95 | 19.45 | 12325 | 18 | 38.3 |
| 18 | 58.52 | 2.95 | 19.51 | 13105 | 18 | 38.9 |
| 19 | 58.56 | 2.95 | 19.14 | 13757 | 18 | 38.9 |
| 20 | 58.56 | 2.95 | 18.94 | 14465 | 18 | 38.9 |
| 21 | 58.41 | 2.94 | 18.45 | 15261 | 18 | 38.9 |
| 22 | 58.48 | 2.95 | 18.42 | 16143 | 18 | 38.9 |
| 23 | 58.51 | 2.95 | 19.89 | 16721 | 18 | 38.9 |
| 24 | 58.5 | 2.95 | 18.18 | 17414 | 18 | 38.9 |
| 25 | 53.83 | 2.71 | 19.68 | 18209 | 18 | 38.9 |
| 26 | 58.25 | 2.94 | 19.77 | 18706 | 18 | 38.9 |
| 27 | 58.27 | 2.94 | 19.05 | 19346 | 18 | 38.9 |
| 28 | 58.26 | 2.94 | 19.75 | 19991 | 18 | 38.9 |
| 29 | 58.26 | 2.94 | 19.86 | 20740 | 18 | 38.9 |
| 30 | 58.25 | 2.94 | 20.18 | 21427 | 18 | 40.8 |
| 31 | 58.82 | 2.96 | 19.93 | 22123 | 18 | 40.8 |
| 32 | 58.42 | 2.94 | 19.67 | 22793 | 18 | 40.8 |
| 33 | 58.02 | 2.92 | 19.79 | 23404 | 18 | 40.8 |
| 34 | 58.35 | 2.94 | 18.92 | 24153 | 18 | 40.8 |
| 35 | 58.48 | 2.95 | 19.69 | 24874 | 18 | 40.8 |
| 36 | 58.5 | 2.95 | 19.95 | 25519 | 17 | 40.8 |
| 37 | 57.95 | 2.92 | 18.97 | 26262 | 17 | 40.8 |
| 38 | 58.12 | 2.93 | 19.07 | 26864 | 17 | 40.8 |
| 39 | 55.44 | 2.79 | 19.95 | 27539 | 17 | 40.8 |
| 40 | 58.35 | 2.94 | 20.13 | 28161 | 17 | 40.8 |
| 41 | 53.74 | 2.71 | 16.45 | 28297 | 17 | 42.4 |
| Rata2 | 61.25 | 2.85 | 21.26 | 15063.049 | - | - |

Gambar 5.8 Grafik performa dengan penerapan *framework* pada Koneksi A dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B. mempunyai penggunaan memori rata – rata 61.25 MB atau 2.85% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 21.869% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 15063.049 KB, penurunan level baterai mencapai 2% dan kenaikan suhu pada perangkat mencapai 5.1 C.

Tabel 5.11 Waktu eksekusi dengan penerapan *framework* pada Koneksi A dan Baterai B

|  |  |  |  |
| --- | --- | --- | --- |
| No. Eksekusi | Metode | Waktu (s) | Kualitas Koneksi |
| 1 | Lokal | 6.568 | *EXCELLENT* |
| 2 | Offloading | 4.002 | *EXCELLENT* |
| 3 | Offloading | 4.005 | *EXCELLENT* |
| 4 | Offloading | 4.003 | *EXCELLENT* |
| 5 | Offloading | 4.008 | *EXCELLENT* |
| 6 | Offloading | 4.006 | *EXCELLENT* |
| 7 | Offloading | 4.025 | *EXCELLENT* |
| 8 | Offloading | 4.03 | *EXCELLENT* |
| 9 | Offloading | 3.005 | *EXCELLENT* |
| 10 | Offloading | 4.37 | *EXCELLENT* |
| 11 | Offloading | 4.033 | *EXCELLENT* |
| 12 | Offloading | 3.001 | *EXCELLENT* |
| 13 | Offloading | 5.003 | *EXCELLENT* |
| 14 | Offloading | 3.004 | *EXCELLENT* |
| 15 | Offloading | 4.006 | *EXCELLENT* |
| 16 | Offloading | 3.003 | *EXCELLENT* |
| 17 | Offloading | 4.002 | *EXCELLENT* |
| 18 | Offloading | 4.008 | *EXCELLENT* |
| 19 | Offloading | 4.003 | *EXCELLENT* |
| 20 | Offloading | 3.016 | *EXCELLENT* |
| 21 | Offloading | 5.01 | *EXCELLENT* |
| 22 | Offloading | 4.014 | *EXCELLENT* |
| 23 | Offloading | 3.009 | *GOOD* |
| 24 | Offloading | 4.137 | *GOOD* |
| 25 | Offloading | 4.314 | *GOOD* |
| 26 | Offloading | 4.188 | *GOOD* |
| 27 | Offloading | 4.296 | *GOOD* |
| 28 | Offloading | 4.024 | *GOOD* |
| 29 | Offloading | 4.017 | *GOOD* |
| 30 | Offloading | 4.016 | *GOOD* |
| 31 | Offloading | 4.013 | *GOOD* |
| 32 | Offloading | 4.576 | *GOOD* |
| 33 | Offloading | 4.013 | *GOOD* |
| 34 | Offloading | 4.037 | *EXCELLENT* |
| 35 | Offloading | 4.429 | *EXCELLENT* |
| 36 | Offloading | 4.024 | *EXCELLENT* |
| 37 | Offloading | 4.026 | *EXCELLENT* |
| 38 | Offloading | 4.027 | *GOOD* |
| 39 | Offloading | 4.238 | *GOOD* |
| 40 | Offloading | 4.402 | *GOOD* |
| Rata - rata | = | 4.048 |  |

Gambar 5.9 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi A dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B dilakukan secara lokal mencapai 2.5% dan secara *offloading* mencapai 97.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 4.048 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 61.25 MB atau 2.85%. Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 21.26%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 2%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar -6.01 MB atau -0.066%, penggunaan CPU rata – rata sebesar 40.379% dan penggunaan level baterai mencapai 1%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 4.048 detik sehingga dihasilkan penghematan waktu eksekusi mencapai 1.842 detik setiap eksekusi beban kerjanya.

### Skenario Uji Coba 4

Skenario uji coba 4 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai B. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 4 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.12. Hasil waktu eksekusi pada uji coba 4 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.13.

Tabel 5.12 Performa dengan penerapan *framework* pada Koneksi B dan Baterai B

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. Cek | Penggunaan Memori | | Penggunaan CPU | Bandwidth Internet | Level Baterai | Suhu perangkat |
| (5s) | (MB) | (%) | (%) | (KB) | (%) | (C) |
| 1 | 40.62 | 2.05 | 0 | 3 | 17 | 45.3 |
| 2 | 39.37 | 1.98 | 15.26 | 7 | 17 | 45.3 |
| 3 | 40.7 | 2.05 | 17.66 | 54 | 17 | 45.3 |
| 4 | 40.73 | 2.05 | 18.17 | 80 | 17 | 45.3 |
| 5 | 40.88 | 2.06 | 17.83 | 85 | 17 | 45.3 |
| 6 | 40.88 | 2.06 | 17.36 | 119 | 17 | 45.3 |
| 7 | 40.9 | 2.06 | 18.71 | 155 | 17 | 45.3 |
| 8 | 40.93 | 2.06 | 16.8 | 191 | 17 | 45.3 |
| 9 | 40.78 | 2.06 | 18.36 | 234 | 17 | 45.3 |
| 10 | 41.11 | 2.07 | 17.53 | 271 | 17 | 45.3 |
| 11 | 75.11 | 3.79 | 30.22 | 300 | 17 | 45.4 |
| 12 | 82.47 | 4.16 | 66.99 | 300 | 17 | 45.4 |
| 13 | 53.36 | 2.69 | 72.36 | 301 | 17 | 45.4 |
| 14 | 56.87 | 2.87 | 58.41 | 353 | 17 | 45.4 |
| 15 | 56.95 | 2.87 | 56.65 | 392 | 17 | 45.4 |
| 16 | 57.18 | 2.88 | 59.78 | 423 | 17 | 45.4 |
| 17 | 57.38 | 2.89 | 56.7 | 443 | 16 | 45.4 |
| 18 | 57.33 | 2.89 | 81.5 | 462 | 16 | 45.4 |
| 19 | 57.46 | 2.9 | 56.1 | 492 | 16 | 45.4 |
| 20 | 55.11 | 2.78 | 79.98 | 510 | 16 | 45.4 |
| 21 | 59.41 | 2.99 | 58.81 | 522 | 16 | 45.4 |
| 22 | 64.99 | 3.27 | 58.92 | 544 | 16 | 45.4 |
| 23 | 57.29 | 2.89 | 78.05 | 563 | 16 | 46.1 |
| 24 | 57.42 | 2.89 | 52.94 | 584 | 16 | 46.1 |
| 25 | 58.62 | 2.95 | 55.59 | 596 | 16 | 46.1 |
| 26 | 60.59 | 3.05 | 55.97 | 608 | 16 | 46.1 |
| 27 | 52.5 | 2.65 | 64.79 | 618 | 16 | 46.1 |
| 28 | 56.5 | 2.85 | 60.13 | 624 | 16 | 47.2 |
| 29 | 57.59 | 2.9 | 66.14 | 632 | 16 | 47.2 |
| 30 | 57.49 | 2.9 | 55.76 | 641 | 16 | 47.2 |
| 31 | 57.49 | 2.9 | 59.01 | 656 | 16 | 47.2 |
| 32 | 52.3 | 2.64 | 63.3 | 686 | 16 | 47.2 |
| 33 | 68.72 | 3.46 | 57.61 | 701 | 16 | 47.2 |
| 34 | 51.72 | 2.61 | 67.33 | 715 | 16 | 47.7 |
| 35 | 57.57 | 2.9 | 63.09 | 740 | 16 | 47.7 |
| 36 | 57.56 | 2.9 | 58.05 | 763 | 16 | 47.7 |
| 37 | 51.49 | 2.59 | 62.05 | 785 | 16 | 47.7 |
| 38 | 54.08 | 2.73 | 66.54 | 797 | 15 | 47.7 |
| 39 | 57.8 | 2.91 | 71.97 | 806 | 15 | 47.7 |
| 40 | 82.26 | 4.15 | 59.83 | 818 | 15 | 48.2 |
| 41 | 57.75 | 2.91 | 78.74 | 827 | 15 | 48.2 |
| 42 | 57.69 | 2.91 | 55.26 | 836 | 15 | 48.2 |
| 43 | 52.42 | 2.64 | 66.97 | 853 | 15 | 48.2 |
| 44 | 57.82 | 2.91 | 69.71 | 874 | 15 | 48.2 |
| 45 | 57.66 | 2.91 | 56.32 | 891 | 15 | 48.2 |
| 46 | 58.07 | 2.93 | 58.49 | 907 | 15 | 48.6 |
| 47 | 57.91 | 2.92 | 64.95 | 923 | 15 | 48.6 |
| 48 | 57.49 | 2.9 | 55.88 | 956 | 15 | 48.6 |
| 49 | 55.28 | 2.79 | 78.93 | 983 | 15 | 48.6 |
| 50 | 58.16 | 2.93 | 56.5 | 1017 | 15 | 48.6 |
| 51 | 56.72 | 2.86 | 65.57 | 1048 | 15 | 48.9 |
| 52 | 58.57 | 2.95 | 46.36 | 1078 | 15 | 48.9 |
| 53 | 93.48 | 4.71 | 56.5 | 1090 | 15 | 48.9 |
| 54 | 57.93 | 2.92 | 56.8 | 1115 | 15 | 48.9 |
| 55 | 59.79 | 3.01 | 58.18 | 1149 | 15 | 48.9 |
| 56 | 57.92 | 2.92 | 58.73 | 1203 | 15 | 48.9 |
| 57 | 59.68 | 3.01 | 51.76 | 1266 | 14 | 49.2 |
| 58 | 75.11 | 3.78 | 51.96 | 1316 | 14 | 49.2 |
| 59 | 54.2 | 2.73 | 57.12 | 1360 | 14 | 49.2 |
| 60 | 58.01 | 2.92 | 72.4 | 1435 | 14 | 49.2 |
| 61 | 57.54 | 2.9 | 54.22 | 1494 | 14 | 49.2 |
| 62 | 57.83 | 2.91 | 77.9 | 1530 | 14 | 49.2 |
| 63 | 57.53 | 2.9 | 54.19 | 1561 | 14 | 49.2 |
| 64 | 52.78 | 2.66 | 66.44 | 1592 | 14 | 49.2 |
| 65 | 58.6 | 2.95 | 67.65 | 1625 | 14 | 49.2 |
| 66 | 57.48 | 2.9 | 75.58 | 1663 | 14 | 49.2 |
| 67 | 51.94 | 2.62 | 17.04 | 1693 | 14 | 49.2 |
| Rata2 | 56.49 | 2.847 | 54.364 | 759.089 | - | - |

Gambar 5.10 Grafik performa dengan penerapan *framework* pada Koneksi B dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai B. mempunyai penggunaan memori rata – rata 56.49 MB atau 2.847% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 54.364% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 759.089 KB, penurunan level baterai mencapai 3% dan kenaikan suhu pada perangkat mencapai 3.9 C.

Tabel 5.13 Waktu eksekusi dengan penerapan *framework* pada Koneksi B Baterai B

|  |  |  |  |
| --- | --- | --- | --- |
| No. Eksekusi | Metode | Waktu (s) | Kualitas Koneksi |
| 1 | Lokal | 7.471 | *POOR* |
| 2 | Lokal | 5.582 | *POOR* |
| 3 | Lokal | 3.968 | *POOR* |
| 4 | Lokal | 6.289 | *POOR* |
| 5 | Lokal | 9.186 | *POOR* |
| 6 | Lokal | 8.447 | *POOR* |
| 7 | Lokal | 7.875 | *POOR* |
| 8 | Lokal | 7.637 | *POOR* |
| 9 | Lokal | 4.868 | *POOR* |
| 10 | Lokal | 5.823 | *POOR* |
| 11 | Lokal | 5.303 | *POOR* |
| 12 | Lokal | 4.753 | *POOR* |
| 13 | Lokal | 7.248 | *POOR* |
| 14 | Lokal | 4.702 | *POOR* |
| 15 | Lokal | 5.887 | *POOR* |
| 16 | Lokal | 4.116 | *POOR* |
| 17 | Lokal | 6.111 | *POOR* |
| 18 | Lokal | 7.066 | *POOR* |
| 19 | Lokal | 6.266 | *POOR* |
| 20 | Lokal | 5.093 | *POOR* |
| 21 | Lokal | 8.284 | *POOR* |
| 22 | Lokal | 7.905 | *POOR* |
| 23 | Lokal | 8.05 | *POOR* |
| 24 | Lokal | 7.44 | *POOR* |
| 25 | Lokal | 6.293 | *POOR* |
| 26 | Lokal | 7.84 | *POOR* |
| 27 | Lokal | 8.29 | *POOR* |
| 28 | Lokal | 8.818 | *POOR* |
| 29 | Lokal | 3.459 | *POOR* |
| 30 | Lokal | 3.48 | *POOR* |
| 31 | Lokal | 4.02 | *POOR* |
| 32 | Lokal | 4.024 | *POOR* |
| 33 | Lokal | 4.293 | *POOR* |
| 34 | Lokal | 5.7 | *POOR* |
| 35 | Lokal | 5.717 | *POOR* |
| 36 | Lokal | 5.145 | *POOR* |
| 37 | Lokal | 7.785 | *POOR* |
| 38 | Lokal | 9.441 | *POOR* |
| 39 | Lokal | 7.223 | *POOR* |
| 40 | Lokal | 10.741 | *POOR* |
| Rata - rata | = | 6.441 |  |

Gambar 5.11 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi B dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi B dan kondisi Baterai B dilakukan secara lokal mencapai 100% dan secara *offloading* mencapai 0% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 6.441 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi A dan kondisi Baterai B akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 56.49 MB atau 2.847%. Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 54.364%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 3%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar -1.25 MB atau -0.063%, penggunaan CPU rata – rata sebesar 7.275% dan penggunaan level baterai mencapai 0%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 6.411 detik sehingga dihasilkan penghematan waktu eksekusi mencapai -0.521 detik setiap eksekusi beban kerjanya.

### Skenario Uji Coba 5

Skenario uji coba 5 adalah perhitungan penghematan daya dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 5 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.14. Hasil waktu eksekusi pada uji coba 5 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.15.

Tabel 5.14 Performa dengan penerapan *framework* pada Koneksi C dan Baterai A

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. Cek | Penggunaan Memori | | Penggunaan CPU | Bandwidth Internet | Level Baterai | Suhu perangkat |
| (5s) | (MB) | (%) | (%) | (KB) | (%) | (C) |
| 1 | 39.7 | 2 | 0 | 140 | 55 | 37.3 |
| 2 | 41.78 | 2.11 | 17.78 | 2823 | 55 | 37.3 |
| 3 | 49.69 | 2.5 | 67.7 | 2957 | 55 | 37.3 |
| 4 | 56.33 | 2.84 | 78.12 | 2957 | 55 | 38.5 |
| 5 | 55.6 | 2.8 | 29.34 | 3803 | 55 | 38.5 |
| 6 | 59.63 | 3 | 20.31 | 4518 | 55 | 38.5 |
| 7 | 59.66 | 3.01 | 21.66 | 5100 | 55 | 38.5 |
| 8 | 60.25 | 3.04 | 21.86 | 5696 | 55 | 38.5 |
| 9 | 60.19 | 3.03 | 21.57 | 6563 | 55 | 38.5 |
| 10 | 60.4 | 3.04 | 21.53 | 7206 | 55 | 38.5 |
| 11 | 52.87 | 2.66 | 20.93 | 7803 | 55 | 38.5 |
| 12 | 58.83 | 2.96 | 33.11 | 7958 | 55 | 38.5 |
| 13 | 57.38 | 2.89 | 54.89 | 7958 | 55 | 38.5 |
| 14 | 58.83 | 2.96 | 40.16 | 7958 | 55 | 38.5 |
| 15 | 69.65 | 3.51 | 46.43 | 7958 | 55 | 41.8 |
| 16 | 68.31 | 3.44 | 60.33 | 7958 | 55 | 41.8 |
| 17 | 57.49 | 2.9 | 84.19 | 7958 | 54 | 41.8 |
| 18 | 58.94 | 2.97 | 55.53 | 7958 | 54 | 41.8 |
| 19 | 58.14 | 2.93 | 40.44 | 7958 | 54 | 41.8 |
| 20 | 56.76 | 2.86 | 61.05 | 7958 | 54 | 41.8 |
| 21 | 55.65 | 2.8 | 36.7 | 8095 | 54 | 41.8 |
| 22 | 52.97 | 2.67 | 30.34 | 8232 | 54 | 41.8 |
| 23 | 53.18 | 2.68 | 21.28 | 8680 | 54 | 41.8 |
| 24 | 53.77 | 2.71 | 21.67 | 9526 | 54 | 41.8 |
| 25 | 53.46 | 2.69 | 22.65 | 10422 | 54 | 41.8 |
| 26 | 53.56 | 2.7 | 19.87 | 11006 | 54 | 41.8 |
| 27 | 53.52 | 2.7 | 21.5 | 11721 | 54 | 43.1 |
| 28 | 53.76 | 2.71 | 21.65 | 12302 | 54 | 43.1 |
| 29 | 53.78 | 2.71 | 21.23 | 12886 | 54 | 43.1 |
| 30 | 53.97 | 2.72 | 18.14 | 13308 | 54 | 43.1 |
| 31 | 53.39 | 2.69 | 19.24 | 13308 | 54 | 43.1 |
| 32 | 57.61 | 2.9 | 68.18 | 13308 | 54 | 43.1 |
| 33 | 56.49 | 2.85 | 57.74 | 13308 | 54 | 43.1 |
| 34 | 81.98 | 4.13 | 53.11 | 13308 | 54 | 43.1 |
| 35 | 56.71 | 2.86 | 76.89 | 13308 | 54 | 43.1 |
| 36 | 56.29 | 2.84 | 37.6 | 13891 | 54 | 43.1 |
| 37 | 56.17 | 2.83 | 55.61 | 13891 | 54 | 43.1 |
| 38 | 55.46 | 2.79 | 39.92 | 14027 | 54 | 43.6 |
| 39 | 54.09 | 2.73 | 22.4 | 14182 | 54 | 43.6 |
| 40 | 53.29 | 2.69 | 20.77 | 14881 | 54 | 43.6 |
| 41 | 52.61 | 2.65 | 21.32 | 15638 | 53 | 43.6 |
| 42 | 53.43 | 2.69 | 21.77 | 16242 | 53 | 43.6 |
| 43 | 54.58 | 2.75 | 22.07 | 17147 | 53 | 43.6 |
| 44 | 55.46 | 2.79 | 21.64 | 17962 | 53 | 43.6 |
| 45 | 55.85 | 2.81 | 21.6 | 18545 | 53 | 43.6 |
| 46 | 54.5 | 2.75 | 20.93 | 19261 | 53 | 43.6 |
| 47 | 54.94 | 2.77 | 21.36 | 19842 | 53 | 43.6 |
| 48 | 54.27 | 2.73 | 22.69 | 20711 | 53 | 43.6 |
| 49 | 54.25 | 2.73 | 20.61 | 21004 | 53 | 43.6 |
| 50 | 51.02 | 2.57 | 11.77 | 21004 | 52 | 44.9 |
| 51 | 51.18 | 2.58 | 14.63 | 21004 | 52 | 44.9 |
| Rata2 | 55.823 | 2.812 | 33.052 | 1142.8 | - | - |

Gambar 5.12 Grafik performa dengan penerapan *framework* pada Koneksi C dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A. mempunyai penggunaan memori rata – rata 55.823 MB atau 2.812% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 33.052% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 1142.8 KB, penurunan level baterai mencapai 3% dan kenaikan suhu pada perangkat mencapai 6.4 C.

Tabel 5.15 Waktu eksekusi dengan penerapan *framework* pada Koneksi C dan Baterai A

|  |  |  |  |
| --- | --- | --- | --- |
| No. Eksekusi | Metode | Waktu (s) | Kualitas Koneksi |
| 1 | Lokal | 6.891 | *EXCELLENT* |
| 2 | Offloading | 3.353 | *EXCELLENT* |
| 3 | Offloading | 3.455 | *EXCELLENT* |
| 4 | Offloading | 3.369 | *EXCELLENT* |
| 5 | Offloading | 3.817 | *EXCELLENT* |
| 6 | Offloading | 3.52 | *EXCELLENT* |
| 7 | Offloading | 3.823 | *EXCELLENT* |
| 8 | Offloading | 3.91 | *EXCELLENT* |
| 9 | Offloading | 3.836 | *EXCELLENT* |
| 10 | Lokal | 7.552 | *UNKNOWN* |
| 11 | Lokal | 7.007 | *UNKNOWN* |
| 12 | Lokal | 4.482 | *UNKNOWN* |
| 13 | Lokal | 7.156 | *UNKNOWN* |
| 14 | Lokal | 4.344 | *UNKNOWN* |
| 15 | Lokal | 7.154 | *UNKNOWN* |
| 16 | Lokal | 7.063 | *UNKNOWN* |
| 17 | Offloading | 3.683 | *EXCELLENT* |
| 18 | Offloading | 3.128 | *EXCELLENT* |
| 19 | Offloading | 3.549 | *GOOD* |
| 20 | Offloading | 2.78 | *GOOD* |
| 21 | Offloading | 3.449 | *GOOD* |
| 22 | Offloading | 3.541 | *GOOD* |
| 23 | Offloading | 4.091 | *EXCELLENT* |
| 24 | Offloading | 3.599 | *GOOD* |
| 25 | Lokal | 6.816 | *UNKNOWN* |
| 26 | Lokal | 6.224 | *UNKNOWN* |
| 27 | Lokal | 8.045 | *UNKNOWN* |
| 28 | Offloading | 11.292 | *MODERATE* |
| 29 | Offloading | 4.087 | *GOOD* |
| 30 | Offloading | 2.408 | *GOOD* |
| 31 | Offloading | 2.49 | *MODERATE* |
| 32 | Offloading | 3.613 | *MODERATE* |
| 33 | Offloading | 3.742 | *MODERATE* |
| 34 | Offloading | 2.816 | *GOOD* |
| 35 | Offloading | 3.543 | *GOOD* |
| 36 | Offloading | 3.001 | *GOOD* |
| 37 | Offloading | 3.725 | *GOOD* |
| 38 | Offloading | 4.309 | *GOOD* |
| 39 | Offloading | 3.148 | *GOOD* |
| 40 | Offloading | 4.087 | *EXCELLENT* |
| Rata - rata | = | 4.547 |  |

Gambar 5.13 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi C dan Baterai A

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A dilakukan secara lokal mencapai 27.5% dan secara *offloading* mencapai 72.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 4.547 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai A akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 55.823 MB atau 2.812%. Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 33.052%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 3%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar -0.583 MB atau -0.028%, penggunaan CPU rata – rata sebesar 28.587% dan penggunaan level baterai mencapai 0%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 4.547 detik sehingga dihasilkan penghematan waktu eksekusi mencapai 1.343 detik setiap eksekusi beban kerjanya.

### Skenario Uji Coba 6

Skenario uji coba 6 adalah perhitungan performa dan waktu eksekusi yang dihasilkan oleh eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B. Skenario dilakukan dengan menerapkan penggunaan *offloading computation framework* dan tanpa penggunaan *offloading computation framework* pada perangkat lunak *image recognition*. Nilai performa dan waktu eksekusi diperoleh dari pemrosesan *image recognition* pada perangkat lunak yang terpasang pada perangkat bergerak Android hingga selesai. Pengecekan nilai performa dilakukan setiap 5 detik sekali. Hasil performa pada uji coba 6 tanpa penerapan *framework* dapat dilihat pada Tabel 5.4 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.16. Hasil waktu eksekusi pada uji coba 6 tanpa penerapan *framework* dapat dilihat pada Tabel 5.5 dan dengan penerapan *framework* dapat dilihat pada Tabel 5.17.

Tabel 5.16 Performa dengan penerapan *framework* pada Koneksi C dan Baterai B

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. Cek | Penggunaan Memori | | Penggunaan CPU | Bandwidth Internet | Level Baterai | Suhu perangkat |
| (5s) | (MB) | (%) | (%) | (KB) | (%) | (C) |
| 1 | 34.45 | 1.74 | 0 | 140 | 14 | 44.5 |
| 2 | 78.36 | 3.95 | 33.94 | 1752 | 14 | 44.5 |
| 3 | 79.99 | 4.03 | 65.93 | 1752 | 14 | 44.5 |
| 4 | 51.51 | 2.6 | 73.61 | 1753 | 14 | 44.5 |
| 5 | 55.17 | 2.78 | 22.2 | 3676 | 14 | 44.5 |
| 6 | 56.9 | 2.87 | 19.56 | 4219 | 14 | 44.5 |
| 7 | 56.11 | 2.83 | 21.1 | 4850 | 14 | 44.5 |
| 8 | 57.65 | 2.91 | 20.43 | 5632 | 14 | 44.5 |
| 9 | 58.06 | 2.93 | 21.43 | 6286 | 14 | 44.5 |
| 10 | 58.46 | 2.95 | 19.13 | 6929 | 14 | 44.5 |
| 11 | 62.78 | 3.16 | 51.78 | 6929 | 14 | 44.5 |
| 12 | 79.06 | 3.98 | 60.21 | 6929 | 14 | 45 |
| 13 | 76.51 | 3.86 | 56.62 | 6929 | 14 | 45 |
| 14 | 63.05 | 3.18 | 65.71 | 6930 | 14 | 45 |
| 15 | 59.46 | 3 | 21.99 | 7070 | 14 | 45 |
| 16 | 59.5 | 3 | 21.81 | 7772 | 13 | 45 |
| 17 | 59.67 | 3.01 | 19.84 | 8719 | 13 | 45 |
| 18 | 59.54 | 3 | 19.32 | 9442 | 13 | 45 |
| 19 | 59.87 | 3.02 | 20.35 | 10089 | 13 | 45 |
| 20 | 59.91 | 3.02 | 19.88 | 10714 | 13 | 45 |
| 21 | 59.88 | 3.02 | 20.42 | 11407 | 13 | 45 |
| 22 | 59.91 | 3.02 | 21.47 | 12188 | 13 | 45 |
| 23 | 59.91 | 3.02 | 19.86 | 12775 | 13 | 46.1 |
| 24 | 59.88 | 3.02 | 20.07 | 13442 | 13 | 46.1 |
| 25 | 59.95 | 3.02 | 20.19 | 14172 | 13 | 46.1 |
| 26 | 59.79 | 3.01 | 17.47 | 14444 | 13 | 46.1 |
| 27 | 64.19 | 3.23 | 71.3 | 14444 | 13 | 46.1 |
| 28 | 64.19 | 3.23 | 57.36 | 14444 | 13 | 46.1 |
| 29 | 63.01 | 3.18 | 65.96 | 14444 | 13 | 46.1 |
| 30 | 64.28 | 3.24 | 52.83 | 14444 | 13 | 46.1 |
| 31 | 60.69 | 3.06 | 45.54 | 14582 | 13 | 46.1 |
| 32 | 60.59 | 3.05 | 21 | 15264 | 13 | 46.1 |
| 33 | 60.31 | 3.04 | 19.62 | 16056 | 13 | 46.1 |
| 34 | 60.31 | 3.04 | 20.08 | 16725 | 13 | 46.1 |
| 35 | 60.34 | 3.04 | 19.25 | 17435 | 13 | 46.6 |
| 36 | 60.5 | 3.05 | 19.17 | 18088 | 13 | 46.6 |
| 37 | 60.48 | 3.05 | 19.79 | 18950 | 13 | 46.6 |
| 38 | 60.47 | 3.05 | 20.18 | 19705 | 12 | 46.6 |
| 39 | 61.62 | 3.11 | 21.36 | 20459 | 12 | 46.6 |
| 40 | 61.64 | 3.11 | 19.49 | 21379 | 12 | 46.6 |
| 41 | 61.61 | 3.1 | 19.96 | 22094 | 12 | 46.6 |
| 42 | 61.61 | 3.1 | 21.37 | 22840 | 12 | 46.6 |
| 43 | 61.62 | 3.11 | 19.97 | 23449 | 12 | 46.6 |
| 44 | 61.65 | 3.11 | 21.91 | 24152 | 12 | 46.6 |
| 45 | 61.64 | 3.11 | 19.71 | 24715 | 12 | 46.6 |
| Rata2 | 61.246 | 3.088 | 30.01 | 12235.8 | - | - |

Gambar 5.14 Grafik performa dengan penerapan *framework* pada Koneksi C dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan performa di atas, eksekusi beban kerja proses *image recognition* menggunakan metode SURF oleh *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B. mempunyai penggunaan memori rata – rata 61.246 MB atau 3.088% dari keseluruhan memori yang dimiliki perangkat *client*, penggunaan CPU rata – rata 30.01% dari keseluruhan yang dimiliki perangkat *client*, pemakaian *bandwidth* rata – rata adalah 12235.8 KB, penurunan level baterai mencapai 2% dan kenaikan suhu pada perangkat mencapai 2.1 C.

Tabel 5.17 Waktu eksekusi dengan penerapan *framework* pada Koneksi C dan Baterai B

|  |  |  |  |
| --- | --- | --- | --- |
| No. Eksekusi | Metode | Waktu (s) | Kualitas Koneksi |
| 1 | Lokal | 6.996 | *EXCELLENT* |
| 2 | Offloading | 4.201 | *EXCELLENT* |
| 3 | Offloading | 4.571 | *EXCELLENT* |
| 4 | Offloading | 3.476 | *GOOD* |
| 5 | Offloading | 4.011 | *GOOD* |
| 6 | Offloading | 4.065 | *GOOD* |
| 7 | Offloading | 4.065 | *GOOD* |
| 8 | Lokal | 7.537 | *UNKNOWN* |
| 9 | Lokal | 4.742 | *UNKNOWN* |
| 10 | Lokal | 6.86 | *UNKNOWN* |
| 11 | Offloading | 3.619 | *EXCELLENT* |
| 12 | Offloading | 3.007 | *GOOD* |
| 13 | Offloading | 4.132 | *GOOD* |
| 14 | Offloading | 4.093 | *GOOD* |
| 15 | Offloading | 4.635 | *GOOD* |
| 16 | Offloading | 3.073 | *GOOD* |
| 17 | Offloading | 3.145 | *GOOD* |
| 18 | Offloading | 3.472 | *GOOD* |
| 19 | Offloading | 4.374 | *GOOD* |
| 20 | Offloading | 4.38 | *GOOD* |
| 21 | Offloading | 4.008 | *EXCELLENT* |
| 22 | Lokal | 7.011 | *UNKNOWN* |
| 23 | Lokal | 8.32 | *UNKNOWN* |
| 24 | Lokal | 8.149 | *UNKNOWN* |
| 25 | Offloading | 3.284 | *MODERATE* |
| 26 | Offloading | 4.436 | *GOOD* |
| 27 | Offloading | 4.089 | *GOOD* |
| 28 | Offloading | 4.014 | *GOOD* |
| 29 | Offloading | 4.246 | *EXCELLENT* |
| 30 | Offloading | 3.105 | *GOOD* |
| 31 | Offloading | 3.095 | *GOOD* |
| 32 | Offloading | 4.137 | *GOOD* |
| 33 | Offloading | 3.038 | *GOOD* |
| 34 | Offloading | 3.052 | *GOOD* |
| 35 | Offloading | 3.276 | *GOOD* |
| 36 | Offloading | 3.143 | *GOOD* |
| 37 | Offloading | 3.318 | *GOOD* |
| 38 | Offloading | 4.139 | *GOOD* |
| 39 | Offloading | 3.569 | *GOOD* |
| 40 | Offloading | 4.071 | *GOOD* |
| Rata - rata | = | 4.349 |  |

Gambar 5.15 Grafik waktu eksekusi dengan penerapan *framework* pada Koneksi C dan Baterai B

Berdasarkan hasil yang ditunjukkan pada perhitungan waktu eksekusi di atas, eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B dilakukan secara lokal mencapai 17.5% dan secara *offloading* mencapai 82.5% serta mempunyai waktu eksekusi rata – rata untuk setiap eksekusi beban kerja mencapai 4.349 detik.

Hasil keluaran antara eksekusi beban kerja *image recognition* menggunakan metode SURF tanpa penerapan *offloading computation framework* dan eksekusi beban kerja proses *image recognition* dengan penerapan *offloading computation framework* pada kondisi Koneksi C dan kondisi Baterai B akan dibandingkan untuk perhitungan penghematan performa dan waktu eksekusi.

Perbandingan pada data performa dimulai dengan penggunaan memori rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 55.24 MB atau 2.784% sedangkan pada metode eksekusi kedua dihasilkan 61.246 MB atau 3.088%. Penggunaan CPU rata – rata pada perangkat *client* pada metode eksekusi pertama dihasilkan 61.639% sedangkan pada metode kedua dihasilkan 30.01%. Penggunaan level baterai pada perangkat *client* pada metode eksekusi pertama mencapai 3% sedangkan pada metode kedua mencapai 2%

Sehingga dapat dikatakan metode kedua dibandingkan dengan metode satu melakukan penghematan performa pada penggunaan memori rata – rata sebesar -6.006 MB atau -0.304%, penggunaan CPU rata – rata sebesar 31.629% dan penggunaan level baterai mencapai 1%.

Perbandingan pada data waktu eksekusi dapat dilihat pada waktu eksekusi rata – rata setiap beban kerja metode eksekusi pertama dihasilkan 5.89 detik sedangkan pada metode kedua dihasilkan 4.349 detik sehingga dihasilkan penghematan waktu eksekusi mencapai 1.541 detik setiap eksekusi beban kerjanya.

## Evaluasi Umum Skenario Uji Coba

Hasil performa dan waktu eksekusi dari ke enam skenario dapat dilihat pada Tabel 5.18.

Tabel 5.18 Hasil performa dan waktu eksekusi dari 6 skenario uji coba

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Uji Coba | | | 1 | 2 | 3 | 4 | 5 | 6 |
| Performa | Memori | (MB) | 50.09 | 52.85 | 61.25 | 56.49 | 55.82 | 61.25 |
| (%) | 2.525 | 2.662 | 2.85 | 2.847 | 2.812 | 3.088 |
| CPU | (%) | 43.39 | 51.87 | 21.26 | 54.36 | 33.05 | 30.01 |
| Baterai | (%) | 2 | 3 | 2 | 3 | 3 | 2 |
| Waktu | (s) | 3.462 | 7.018 | 4.048 | 6.411 | 4.547 | 4.349 |
| Penghematan | Memori | (MB) | 5.16 | 2.41 | -6.01 | -1.25 | -0.58 | -6.01 |
| (%) | 0.259 | 0.122 | -0.07 | -0.06 | -0.03 | -0.35 |
| CPU | (%) | 18.249 | 9.77 | 40.379 | 7.275 | 28.59 | 31.629 |
| Baterai | (%) | 1 | 0 | 1 | 0 | 0 | 1 |
| Waktu | (s) | 2.428 | -1.128 | 1.842 | -0.521 | 1.343 | 1.541 |

Berdasarkan skenario uji coba ke satu yang telah dilakukan, dapat diketahui bahwa adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan daya pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu sebesar **1%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya tingkat peningkatan kinerja pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu pada memori terdapat peningkatan sebesar **5.16 MB** atau **0.259%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **9.249%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui adanya tingkat peningkatan waktu eksekusi rata -rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation* *framework* sebesar **2.428 detik.**

Sedangkan berdasarkan skenario uji coba ke dua yang telah dilakukan, dapat diketahui bahwa tidak adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation* *framework* dengan daya pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu sebesar **0%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya peningkatan kinerja pada perangkat *client* dengan penerapan *offloading computation* *framework* dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu pada memori terdapat peningkatan sebesar **2.41 MB** atau **0.122%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **9.77%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui bahwa adanya tingkat penurunan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation* *framework* dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu sebesar -**1.128 detik.**

Sedangkan berdasarkan skenario uji coba ke tiga yang telah dilakukan, dapat diketahui bahwa adanya adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation* *framework* dengan daya pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu sebesar **1%**. Dari hasil uji coba juga diketahui bahwa terdapat peningkatan sekaligus penurunan kinerja pada perangkat *client* dengan penerapan *offloading computation* *framework* dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu pada memori terdapat penurunan sebesar -**6.01 MB** atau -**0.07%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **40.379%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui bahwa adanya tingkat peningkatan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation* *framework* dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu sebesar **1.842 detik**.

Sedangkan berdasarkan skenario uji coba ke empat yang telah dilakukan, dapat diketahui bahwa tidak adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan daya pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu sebesar **0%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya tingkat peningkatan sekaligus penurunan kinerja pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu pada memori terdapat penurunan sebesar -**1.25 MB** atau -**0.06%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **7.275%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui adanya tingkat penurunan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation* *framework* sebesar -**0.521 detik.**

Sedangkan berdasarkan skenario uji coba ke lima yang telah dilakukan, dapat diketahui bahwa tidak adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan daya pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu sebesar **0%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya tingkat peningkatan sekaligus penurunan kinerja pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu pada memori terdapat penurunan sebesar -**0.58 MB** atau -**0.03%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **28.59%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui adanya tingkat peningkatan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation* *framework* sebesar **1.343 detik.**

Sedangkan berdasarkan skenario uji coba ke enam yang telah dilakukan, dapat diketahui bahwa adanya tingkat penghematan daya pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan daya pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu sebesar **1%**. Dari hasil uji coba juga diketahui bahwa terdapat adanya tingkat peningkatan sekaligus penurunan kinerja pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan kinerja pada perangkat *client* tanpa penerapan *offloading computation* *framework* yaitu pada memori terdapat penurunan sebesar -**6.01 MB** atau -**0.35%** dari keseluruhan memori pada perangkat *client*, pada CPU terdapat peningkatan sebesar **31.629%** dari keseluruhan CPU yang dimiliki perangkat *client*. Selain itu, dari hasil uji coba juga diketahui adanya tingkat peningkatan waktu eksekusi beban rata – rata kerja proses *image recognition* pada perangkat *client* dengan penerapan *offloading computation* *framework* dibandingkan dengan waktu eksekusi rata – rata beban kerja proses *image recognition* pada perangkat *client* tanpa penerapan *offloading computation* *framework* sebesar **1.541 detik.**

***(Halaman ini sengaja dikosongkan)***

# BAB VI KESIMPULAN DAN SARAN

Bab ini berisikan kesimpulan yang dapat dapat diambil dari hasil uji coba yang telah dilakukan. Selain kesimpulan, terdapat juga saran yang ditujukan untuk pengembangan perangkat lunak selanjutnya.

## Kesimpulan

Kesimpulan yang didapatkan berdasarkan hasil uji coba penghematan daya dan peningkatan kinerja pada perangkat bergerak Android menggunakan *offloading computation framework* yang diterapkan pada perangkat lunak *image recognition* adalah sebagai berikut:

1. Implementasi metode *offloading* dapat digunakan sebagai salah satu metode penghematan daya dan peningkatan kinerja pada perangkat bergerak Android. Namun juga dapat menurunkan waktu eksekusi beban kerja dan penghematan daya pada perangkat bergerak Android.
2. Implementasi *mobile framework* dapat dijadikan sebagai metode yang cocok untuk meminimalisir kerugian menggunakan metode *offloading computation* dengan cara menentukan secara dinamis metode pengeksekusian beban kerja pada perangkat Android.
3. Implementasi JADE *middleware* dapat digunakan dalam menerapkan metode *offloading* dengan cara melakukan pengiriman dan penerimaan modul pengiriman data yang bertipe *string* antara *client* dan *server*.
4. Pada kasus Tugas Akhir ini, *mobile computation offloading framework* melakukan penghematan daya dan peningkatan kinerja yang paling optimal pada saat perangkat bergerak Android memiliki kualitas koneksi internet yang stabil dalam melakukan *transmit* dan *receive* data (kategori Koneksi A).
5. Perhitungan penghematan daya, performa, dan waktu eksekusi terbaik yang didapatkan masing – masing penghematan daya 1% (uji coba 1, 2, 3), penggunaan memori 2.525% sehingga melakukan penghematan penggunaan memori sebesar 0.259% dibandingkan dengan eksekusi beban kerja tanpa *framework* (uji coba 1), penggunaan CPU 21.26% sehingga melakukan penghematan penggunaan CPU sebesar 40.379% (uji coba 3) dan waktu eksekusi rata - rata beban kerja 3.462 detik sehingga melakukan penghematan waktu eksekusi rata – rata sebesar 2.428 detik (uji coba 1).
6. *Decision Maker* pada *mobile computation offloading framework* dapatmenentukan eksekusi beban kerja secara dinamis untuk meminimalisir waktu eksekusi, meningkatkan penghematan daya, serta peningkatan performa pada perangkat bergerak Android yang hasil penerapannya dapat dilihat pada Tabel 5.18.

## Saran

Saran yang diberikan terkait pengembangan pada Tugas Akhir ini adalah:

1. Menambah jumlah data karena dari total 40 data citra yang digunakan sebagai dataset dianggap masih terlalu sedikit.

Ditambahkannya kondisi jika perubahan kualitas koneksi internet menjadi buruk terjadi sesaat setelah pengiriman beban kerja oleh *offloading* *framework* menuju *server* dilakukan. Sehingga perangkat bergerak *client* akan menunda pemrosesan beban kerja *image recognition* selanjutnya akibat menunggu hasil eksekusi dari *server*. Pada studi kasus ini, *server* tidak dapat mengirim kembali hasil eksekusi akibat koneksi buruk yang dimiliki perangkat bergerak *client*.

1. Pengecekan sumber daya baterai lebih detail pada perangkat bergerak *client* yang bisa dilakukan dengan alat – alat tertentu.

# DAFTAR PUSTAKA

[1] K. Liu, J. Peng, H. Li, X. Zhang, dan W. Liu, “Multi-device task offloading with time-constraints for energy efficiency in mobile cloud computing.” [Daring]. Tersedia pada: http://www.sciencedirect.com/science/article/pii/S0167739X16300905. [Diakses: 01-Jun-2017].

[2] “Smartphone definition (Phone Scoop).” [Daring]. Tersedia pada: http://www.phonescoop.com/glossary/term.php?gid=131. [Diakses: 31-Mei-2017].

[3] “UCI Machine Learning Repository: Leaf Data Set.” [Daring]. Tersedia pada: https://archive.ics.uci.edu/ml/datasets/leaf. [Diakses: 01-Jun-2017].

[4] “Google’s Android OS: Past, Present, and Future,” *Phone Arena*. [Daring]. Tersedia pada: http://www.phonearena.com/news/Googles-Android-OS-Past-Present-and-Future\_id21273. [Diakses: 31-Mei-2017].

[5] M. Wang, “Novel Mobile Computation Offloading Framework for Android Devices,” *Eng. Appl. Sci. Theses Diss.*, Des 2014.

[6] “A survey on clustering algorithms for wireless sensor networks.” [Daring]. Tersedia pada: http://www.sciencedirect.com/science/article/pii/S0140366407002162. [Diakses: 31-Mei-2017].

[7] “Jade Site | Java Agent DEvelopment Framework.” .

[8] *gson: A Java serialization/deserialization library to convert Java Objects into JSON and back*. Google, 2017.

[9] “Lang – Home.” [Daring]. Tersedia pada: https://commons.apache.org/proper/commons-lang/. [Diakses: 31-Mei-2017].

[10] “OpenCV library.” [Daring]. Tersedia pada: http://opencv.org/. [Diakses: 31-Mei-2017].

[11] P. Sykora, P. Kamencay, dan R. Hudec, “Comparison of SIFT and SURF Methods for Use on Hand Gesture Recognition based on Depth Map,” *AASRI Procedia*, vol. 9, hal. 19–24, Jan 2014.

[12] G. Caire, “JADEProgramming-Tutorial-for-beginners.pdf.” 30-Jun-2009.

[13] G. Caire dan F. Pieri, “LEAPUserGuide.pdf.” 15-Nov-2011.

[14] *network-connection-class: Listen to current network traffic in the app and categorize the quality of the network*. Facebook, 2017.

[15] H. Qian dan D. Andersen, “Jade: Reducing Energy Consumption of Android App.” .

# LAMPIRAN

***(Halaman ini sengaja dikosongkan)***

# BIODATA PENULIS

Putro Satrio Wibowo merupakan anak dari pasangan Bapak Wasisto dan Ibu Nanik Wahyuni. Lahir di Lumajang pada tanggal 4 November 1994. Penulis menempuh pendidikan formal dimulai dari TK PGRI Pasirian (1999-2001), SDN 4 Pasirian (2001-2007), SMPN 1 Pasirian (2007-20010), SMAN 2 Lumajang (2010-2013) dan S1 Teknik Informatika ITS (2013-2017). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Komputasi Berbasis Jaringan (KBJ). Penulis aktif dalam organisasi seperti Himpunan Mahasiswa Teknik Computer-Informatika (2014-2015) dan KMI (2014-2016). Penulis juga aktif dalam berbagai kegiatan kepanitiaan yaitu SCHEMATICS 2014 divisi National Programming Competition dan SCHEMATICS 2015 divisi Perlengkapan dan Transportasi. Penulis juga menyukai kegiatan sosial dan pecinta alam. Penulis memiliki hobi futsal dan bermain game. Penulis dapat dihubungi melalui email: [putrosatrio27@gmail.com](mailto:putrosatrio27@gmail.com).