TUGAS AKHIR – IF184802

IMPLEMENTASI METODE REINFORCEMENT LEARNING PADA SIMULASI PERGERAKAN ROBOT MULTI-SENDI

MUHAMMAD ADHITYA IRVANSYAH
NRP 05111540000143

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

Dosen Pembimbing II
Shintami Chusnul Hidayanti, S.Kom., M.Sc., Ph.D

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – IF184802

IMPLEMENTASI METODE REINFORCEMENT LEARNING PADA SIMULASI PERGERAKAN ROBOT MULTI-SENDI

MUHAMMAD ADHITYA IRVANSYAH
NRP 05111540000143

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

Dosen Pembimbing II
Shintami Chusnul Hidayati, S.Kom., M.Sc., Ph.D

Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS – IF184802

IMPLEMENTATION OF REINFORCEMENT LEARNING METHOD ON MULTI-JOINTS ROBOT MOVEMENT SIMULATION

MUHAMMAD ADHITYA IRVANSYAH
NRP 05111540000143

First Advisor
Dini Adni Navastara, S.Kom., M.Sc.

Second Advisor
Shintami Chusnul Hidayati, S.Kom., M.Sc., Ph.D

Department of Informatics Engineering
Faculty of Electronics Technology and Smart Informatics
Institut Teknologi Sepuluh Nopember
Surabaya 2020

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI METODE REINFORCEMENT LEARNING PADA SIMULASI PERGERAKAN ROBOT MULTI-SENDI

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Cerdas dan Visi
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh:
MUHAMMAD ADHITYA IRVANSYAH
NRP: 05111540000143

Disetujui oleh Pembimbing Tugas Akhir:

1. Dini Adni Navastara, S.Kom., M.Sc. ..... (Pembimbing 1)
   (NIP. 19851017 201504 2 001)

2. Shintami Chusnul H., S.Kom., M.Sc., Ph.D. ..... (Pembimbing 2)
   (NIP. 1987202012004)

SURABAYA
JUNI, 2020

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI METODE *REINFORCEMENT LEARNING* PADA SIMULASI PERGERAKAN ROBOT MULTI-SENDI

Nama Mahasiswa	**: Muhammad Adhitya Irvansyah**
NRP	**: 05111540000143**
Jurusan	**: Teknik Informatika, FTEIC**
Dosen Pembimbing 1	**: Dini Adni Navastara, S.Kom., M.Sc.**
Dosen Pembimbing 2	**: Shintami Chusnul Hidayati, S.Kom., M.Sc., Ph.D**

ABSTRAK

Reinforcement Learning (RL) merupakan salah satu cabang kecerdasan buatan yang sedang berkembang dan banyak diteliti. Perbedaan dari RL dengan cabang lain pada studi kecerdasan buatan adalah algoritma pada RL tidak belajar dari korelasi antara masukan dan keluaran yang seharusnya. Namun algoritma ini belajar dari feedback atau timbal balik yang dia dapat saat dia memberikan suatu keluaran atau aksi dari masukan yang ia terima. Dengan metode pembelajaran ini, algortima RL bisa menyelesaikan berbagai masalah yang penyelesaiannya tidak bisa diberikan langsung oleh manusia sehingga algoritma harus mencari sendiri jawaban dari permasalahannya, seperti kontrol pada robot, mengemudikan mobil, bermain gim dan sebagainya.

Pada tugas akhir ini, penulis mengimplementasikan sebuah algoritma RL menggunakan Augmented Random Search (ARS). Algoritma ARS bekerja dengan cara memberikan perubahan-perubahan acak pada agen dan menggunakan perubahan yang dinilai terbaik untuk melakukan perbaruan pada agen tersebut.

Algoritma ARS diimplementasikan pada serangkaian simulasi pergerakan robot multi-sendi untuk melihat dan menguji apakah algoritma ini dapat menyelesaikan goals atau tujuan dari simulasi tersebut. Dari hasil pengujian berbagai simulasi robot multi-sendi menunjukkan bahwa algoritma ARS mampu

menyelesaikan simulasi-simulasi tersebut dan bahkan memberikan performa yang menyaingi algoritma Asynchronus Actor Critic (A2C) yang memiliki sample complexity lebih besar, dengan mencapai hasil 243% lebih baik pada salah satu simulasi.

Kata kunci: *Reinforcement Learning*, *Random Search*, Simulasi Pergerakan Robot, *Augmented Random Search*

IMPLEMENTATION OF REINFORCEMENT LEARNING METHOD ON MULTI-JOINT ROBOT MOVEMENT SIMULATION

Student's Name : Muhammad Adhitya Irvansyah
Student's ID : 05111540000143
Department : Informatics Engineering , Faculty of Electics-ITS
First Advisor : Dini Adni Navastara, S.Kom., M.Sc.
Second Advisor : Shintami Chusnul Hidayati, S.Kom., M.Sc., Ph.D

ABSTRACT

Reinforcement Learning (RL) is one of the branch of Aritificial Intelligent that are being studied and researched. The different between RL and the other branches is how the algorithm learn to do its task. Reinforcement Learning algorithm doesn't learn the correlation between inputs and expected output. It learns from how the feedback from the outputs that they give. The output itself come from processing the observed data or the inputs from the system.With the way RL algorithm works, it can solve the problem that a normal algorithm cannot solve due to large solution space that we human cannot provide, such as robot control, self driving car, playing a game and so on.

On this undergraduate thesis, writer will implement an algorithm called Augmented Random Search (ARS) that published in the paper with title "Simple Random Search Provides a Competitive Approach to Reinforcement Learning". ARS algorithm works by giving random change in the models and see which changes gives the best result and use that change to update the model.

This algorithm will be implemented to run a series of multi-jointed robot movement simulation to test the ability of this algorithm to finish the task of the simulations. The result of these

test shows that ARS Algorith can solve many type of multi-joint robot simulation and have a good performance. With an optimal parameter setting, ARS can even compete and beat the more complex RL algorithm that uses Deep Learning with more than 243% better result in one of the simulation using same amount of sample data.

Keywords: ***Machine Learning, Reinforcement Learning, Random Search, Simulation, Augmented Random Search.***

KATA PENGANTAR

Puji syukur saya sampaikan kepada Tuhan yang Maha Esa karena berkat rahmat-Nya saya dapat melaksanakan Tugas Akhir yang berjudul:

"IMPLEMENTASI METODE *REINFORCEMENT LEARNING* PADA SIMULASI PERGERAKAN ROBOT MULTI-SENDI"

Terselesaikannya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan banyak pihak, oleh karena itu melalui lembar ini penulis ingin mengucapkan terima kasih dan penghormatan kepada:

1. Allah Subhanahu Wa Ta'ala, karena limpahan rahmat dan karunia-Nya penulis dapat menyelesaikan Tugas Akhir dan juga perkuliahan di Informatika ITS.
2. Orangtua penulis, dan anggota keluarga lainnya yang telah memberikan dukungan doa, moral, dan material kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Dini Adni Navastara, S.Kom., M.Sc. dan Shintami Nurul Hidayati, S.Kom, Ph.D selaku pembimbing I dan II yang telah membimbing dan memberikan motivasi, nasihat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
4. Dr. Eng. Chastine Fatichah, S.Kom,, M.Kom. selaku Ketua Departemen Informatika ITS dan seluruh dosen dan karyawan Departemen Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa kuliah di Informatika ITS.
5. Admin-admin Laboratorium Komputasi Cerdas & Visi (KCV) yang memberikan kesempatan penulis untuk fokus mengerjakan Tugas Akhir ini dan menyediakan tempat di laboratorium tersebut.

6. Atikah Ayu Widyadhana yang telah menemani dan memberi dukungan kepada penulis selama proses pengerjaan Tugas Akhir ini.
7. Seluruh mahasiswa Informatika ITS angkatan 2015 yang telah menjadi teman penulis selama menjalani masa kuliah di Informatika ITS.
8. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa laporan Tugas Akhir ini masih memiliki banyak kekurangan. Oleh karena itu dengan segala kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan penulis kedepannya. Selain itu, penulis berharap laporan Tugas Akhir ini dapat berguna bagi pembaca secara umum.

Surabaya, Juni 2020

DAFTAR ISI

LEMBAR PENGESAHAN..... Error! Bookmark not defined.
ABSTRAK.....ix
ABSTRACT.....xi
KATA PENGANTAR.....xiii
DAFTAR ISI.....xv
DAFTAR TABEL.....xix
DAFTAR KODE SUMBER.....xxi
DAFTAR GAMBAR.....xxiii
BAB I PENDAHULUAN.....25
1.1 Latar Belakang.....25
1.2 Rumusan Masalah.....26
1.3 Batasan Permasalahan.....27
1.4 Tujuan.....27
1.5 Manfaat.....27
1.6 Metodologi.....28
1.6.1 Penyusunan Proposal Tugas Akhir.....28
1.6.2 Studi Literatur.....28
1.6.3 Perancangan.....28
1.6.4 Implementasi Perangkat Lunak.....28
1.6.5 Pengujian dan Evaluasi.....29
1.6.6 Penyusunan Buku.....29
1.7 Sistematika Penulisan Laporan.....29
BAB II TINJAUAN PUSTAKA.....31
2.1 Reinforcement Learning.....31
2.2 Environment dalam Reinforcement Learning.....32
2.2.1 Environment OpenAI Gym.....33
2.2.2 MuJoco Locomotion.....33
2.3 Robot Multi-sendi.....35
2.4 Data Normalization.....35
2.5 Finite Differences.....35
2.6 Algoritma Augmented Random Search.....36

2.7 Pengukuran Performa Reinforcement Learning.....38
2.8 Python 38
2.9 Pybullet.....39
2.10 Python Multiprocessing.....39
2.11 Numpy 39
BAB III PERANCANGAN SISTEM.....41
3.1 Perancangan Data 41
3.1.1 Simulasi HalfCheetah 42
3.1.2 Simulasi Hopper 42
3.2 Desain Umum Sistem 43
3.2.1 *Hyperparameter*.....45
3.2.2 Generate Kandidat *Policy*.....45
3.2.3 Eksplorasi Kandidat Policy.....47
3.2.4 Pembaruan *Policy* Berdasarkan *Reward* Terbaik 48
3.2.5 Pengujian *Policy* 50
BAB IV IMPLEMENTASI.....51
4.1 Lingkungan Implementasi 51
4.1.1 Perangkat Keras 51
4.1.2 Perangkat Lunak.....51
4.2 Implementasi Kelas Hyperparameter.....51
4.3 Implementasi Normalizer 53
4.3.1 Menghitung Rata-rata dan Variance.....53
4.3.2 Normalisasi Hasil Observasi 53
4.4 Implementasi Policy 54
4.4.1 Implementasi Pengambilan Keputusan 54
4.4.2 Implementasi Pembuatan Matriks Pertrubation....55
4.4.3 Implementasi Proses Perbaruan Matriks Bobot.....55
4.5 Implementasi Modul Worker 56
4.5.1 Fungsi Eksplore 56
4.6 Modul ARS 57
BAB V UJI COBA DAN EVALUASI.....59
5.1 Lingkungan Uji Coba 59
5.2 Dataset 59

5.3 Target Reward Uji Coba.....60
5.4 Skenario Uji Coba 61
5.4.1 Uji Implementasi Berbagai Versi ARS 62
5.4.2 Uji Coba *Hyperparameter* dari ARS.....63
5.4.3 Uji Coba Performa ARS 67
5.4.4 Perbandingan Performa ARS dengan Algoritma A2C 70
5.5 Hasil dan Evaluasi 70
BAB VI KESIMPULAN DAN SARAN.....74
6.1 Kesimpulan 74
6.2 Saran.....75
DAFTAR PUSTAKA 76
LAMPIRAN.....78
BIODATA PENULIS.....82

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

Tabel 2.1 Tabel 2.1 Penjelasan notasi rumus pembaruan matriks bobot (2.2) 37
Tabel 2.2 Reward Maksimal dan Threshold untuk *Benchmarking* [11] 38
Tabel 3.1 Spesifikasi *halfcheetah* 42
Tabel 3.2 Spesifikasi *Hopper* 43
Tabel 4.1 Jenis-jenis *hyperparameter* 52
Tabel 5.1 *threshold reward* yang akan digunakan dalam pengujian 60
Tabel 5.2 Parameter awal yang digunakan dalam implementasi ARS 61
Tabel 5.3 *Hyperparameter* yang tidak akan berubah selama pengujian 64
Tabel 5.4 hasil percobaan variasi parameter *random directions* . 65
Tabel 5.5 Hasil pengujian implementasi ARS 69
Tabel 5.6 Pengujian Berdasarkan jumlah step pada *environment* 70

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

Kode Sumber 4.1inisiasi kelas *hyperparameter*.....52
Kode Sumber 4.2 fungsi untuk menghitung rata-rata dan varians 53
Kode Sumber 4.3 fungsi untuk normalisasi input 54
Kode Sumber 4.4 fungsi *evaluate* untuk memproses masukan dan menghasilkan keluaran aksi.....54
Kode Sumber 4.5 fungsi *sample_deltas* untuk membuat kumpulan matriks *perturbation* random.....55
Kode Sumber 4.6 fungsi *update* untuk melakukan perbaruan matriks bobot.....55
Kode Sumber 4.7 fungsi *explore* untuk menjalankan simulasi sesuai matriks bobot dan *perturbation* yang ada.....56

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

Gambar 2.1 Hubungan antara *agent* dan *environment* reinforcement learning 31
Gambar 2.2 Simulasi robot HalfCheetah..... 34
Gambar 2.3 Simulasi robot humanoid..... 34
Gambar 3.1 simulasi *halfcheetah* 41
Gambar 3.2 Simulasi robot *hopper* 43
Gambar 3.3 Interaksi ARS dengan *Agent* dan *Environment* 43
Gambar 3.4 Diagram alir algoritma ARS..... 44
Gambar 3.5 Diagram alir proses generate kandidat *Policy* 46
Gambar 3.6 Diagram alir ekplorasi kandidat *Policy* 47
Gambar 3.7 Proses Pembaruan Policy 49
Gambar 5.1 Hasil pengujian berbagai versi implementasi dari ARS 63
Gambar 5.2 hasil percobaan variasi nilai *random directions* 64
Gambar 5.3 Hasil percobaan variasi nilai parameter *best directions* 66
Gambar 5.4 Hasil percobaan variasi nilai parameter *learning rates* 67
Gambar 5.5 Pengujian simulasi hopper..... 68
Gambar 5.6 Pengujian simulasi HalfCheetah..... 68
Gambar 5.7 Pengujian simulasi Walker2d 69

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 *Latar Belakang*

Machine Learning adalah salah satu pengaplikasian Kecerdasan Buatan yang dibuat agar suatu sistem dapat belajar dan berkembang tanpa harus di beri perintah secara spesifik. *Machine Learning* dapat diaplikasikan secara luas karena pada dasarnya kita hanya perlu memberikan *dataset* atau kumpulan data yang besar ke algoritma *Machine Learning* untuk menghasilkan "mesin" yang terlatih yang mampu menyelesaikan tugas yang ada [1]. Terdapat tiga kelas utama dari teknik *Machine Learning*, yaitu *Supervised learning*, *Unsupervised learning*, dan yang terakhir adalah *Reinforcement learning*.

Reinforcement Learning (RL) merupakan salah satu cabang dari *Machine Learning*. Algoritma pada RL berjalan dengan menerima masukan dari observasi sebuah *environment* dan memberikan aksi yang sesuai. Metode ini tidak belajar dari apa keluaran yang seharusnya ia keluarkan, namun metode ini belajar dari *feedback* yang diberikan dari *environment*nya.berbeda dengan *supervised learning* yang belajar dari data yang diberikan. Metode RL bertujuan untuk memaksimalkan *reward* atau *feedback* yang diberikan oleh *environment*. Apabila dianalogikan, pembelajaran pada RL bekerja seperti padahalnya anak kecil yang sedang belajar berjalan. Pada awalnya tentu anak kecil akan jatuh saat pertama kali belajar berjalan, dan akan menerima *feedback* berupa rasa sakit saat jatuh. Tentu anak kecil yang sedang belajar ini akan berusaha agar tidak mendapat rasa sakit tersebut. Sama halnya pada RL, agen akan berusaha memaksimalkan *reward* atau memperkecil *punishment* yang didapat.

Metode RL pada dasarnya diimplementasikan untuk menyelesaikan masalah yang kompleks dengan cara membiarkan program mencoba dan belajar sendiri cara menyelesaikan masalah tersebut. Hal ini dikarenakan semakin kompleks masalahnya maka akan semakin sulit juga bagi manusia untuk memberikan data yang

tepat dan konsisten agar program dapat belajar dari datanya [2]. Banyak masalah komplek yang bisa diselesaikan oleh metode RL beberapa contohnya adalah, memenangkan sebuah permainan [3], menentukan strategi bisnis yang tepat untuk memaksimalkan keuntungan [4], sistem pengendara tanpa awak di mobil, sistem pergerakan robot [5], dan lain-lain.

Terdapat berbagai algoritma RL yang sudah dipublikasikan dan sudah diaplikasikan di berbagai bidang. Salah satu yang paling terkenal adalah algoritma yang dibuat oleh google yang mampu bermain Go [6] dan mengalahkan pemain profesional. Karakteristik berbagai algoritma berbasis *Deep Learning* yang terkenal adalah membutuhkan sampel data yang besar atau kompleks.

Augmented Random Search atau ARS merupakan salah satu algoritma RL yang dibuat berdasarkan algoritma *Random Search*. ARS dinilai mampu mencapai *threshold reward* dengan jumlah sample data yang lebih sedikit, sehingga waktu yg dibutuhkan pada prosess *training* algoritmanya juga akan lebih sedikit. ARS juga memberikan pandangan lain tentang cara menyelesaikan permasalahan RL tanpa menggunakan dasar algoritma *Deep Learning.*

Pada tugas akhir ini akan diimplementasikan algoritma *Augmented Random Search* (ARS) yang akan menjadi agen yang menggerakkan robot multi-sendi yang akan dilakukan pada lingkungan simulasi. Dengan begitu kita akan mengetahui apakah algoritma ARS dapat diimplementasikan untuk menggerakkan robot dan bagaimana perbandingan performa ARS dengan algoritma RL yang lain.

1.2 *Rumusan Masalah*

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana kondisi lingkungan simulasi yang menjadi tempat belajar dan pengujian dari sistem *Reinforcement Learning?*
2. Bagaimana mengimplementasikan metode RL dalam simulasi pergerakan robot?
3. Bagaimana performa sistem dalam menyelesaikan simulasi pergerakan robot dengan menggunakan algoritma *Augmented Random Search?*

1.3 *Batasan Permasalahan*

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Algoritma ditulis dengan bahasa pemrograman python
2. Simulasi dijalankan menggunakan pustaka OpenAI Gym
3. Menggunakan simulasi *HalfCheetah* dan *Hopper* yang disediakan dalam pustaka OpenAI Gym

1.4 *Tujuan*

Tujuan dari pembuatan Tugas Akhir ini adalah mengimplementasikan metode RL dengan algoritma *Augmented Random Search* pada simulasi pergerakan robot multi-sendi.

1.5 *Manfaat*

Tugas akhir ini diharapkan dapat memberikan contoh implementasi dari algoritma *Augmented Random Search*. Hasil implementasi dapat memberikan pandangan dan perbandingan performanya. Diharapkan tugas akhir ini bisa menjadi acuan dalam implementasi algoritma ARS maupun algoritma RL lain dalam menyelesaikan berbagai masalah kontrol yang sekuensial (*continous control problems)* seperti kontrol pergerakan robot, sistem pengendaraan pada mobil tanpa awak, kecerdasan buatan dalam permainan, dan berbagai masalah lain.

1.6 *Metodologi*

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahapan awal dari Tugas Akhir ini adalah penyusunan Proposal Tugas Akhir yang berisi pendahuluan, deskripsi dan gagasan metode-metode yang dibuat dalam Tugas Akhir ini. Pendahuluan ini terdiri dari latar belakang diajukannya Tugas Akhir, rumusan masalah dan batasan masalah yang ditetapkan, serta manfaat dari hasil pembuatan Tugas Akhir ini. Selain itu, dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan Tugas Akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan Tugas Akhir.

1.6.2 Studi Literatur

Pada studi literatur ini akan dipelajari sejumlah referensi yang relevan terhadap pembuatan program yaitu mengenai bahasa pemrograman Python, dasar-dasar pada *Machine Learning*, pengetahuan dasar dan metode-metode pada RL, penggunaan *Toolkit* OpenAI Gym yang akan digunakan sebagai *toolkit* untuk menjadi penyedia simulasi dan *environment* untuk implementasi algoritma, dan Mujoco Locomotion yang akan menjadi jenis simulasi yang akan digunakan

1.6.3 Perancangan

Pada tahap ini penulis akan menganalisis dan merancang sistem yang akan dibuat. Hal ini bertujuan untuk dapat mengimplementasikan algortima dan *pseudo code* secara tepat, sehingga jalannya sistem sesuai dan dapat menyelesaikan tujuan dari tugas akhir ini.

1.6.4 Implementasi Perangkat Lunak

Pada tahap ini akan dilaksanakan implementasi metode dan algoritma yang telah direncanakan. Implementasi sistem

menggunakan *Python 3* sebagai bahasa pemrograman, *OpenAI Gym* sebagai *framework*, serta *library* pendukung lainya.

Pada tugas akhir ini akan diimplementasikan sistem sebagai berikut:

1. Implementasi versi pertama dari algoritma *Augmented Random Search*
2. Implementasi versi kedua dari algoritma *Augmented Random Search*
3. Implementasi variasi dari versi kedua algoritma *Augmented Random Search*

Versi yang akan diimplementasikan adalah versi yang ditulis dalam jurnal "*Simple random search provides a competitive approach to reinforcement learning".* Perbedaan dari ketiga implementasi ini akan dijabarkan pada Bab 3.

1.6.5 Pengujian dan Evaluasi

Pengujian akan dilakukan untuk melihat apakah implementasi yang sudah dibuat dapat menyelesaikan simulasi pergerakan robot multi-sendi, membandingkan performa dan hasil dari ketiga implementasi *Augmented Random Search*, serta membandingkan algoritma tersebut dengan algoritma RL lain yang terkenal seperti A2C dan TRPO.

1.6.6 Penyusunan Buku

Pada tahap ini dilakukan penyusunan buku yang menjelaskan seluruh konsep, teori dasar dari metode yang digunakan, implementasi, serta hasil yang telah dikerjakan sebagai dokumentasi dari pelaksanaan Tugas Akhir.

1.7 *Sistematika Penulisan Laporan*

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

Bab I Pendahuluan

Bab ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.

Bab II Tinjauan Pustaka

Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam penyusunan Tugas Akhir ini. Secara garis besar, bab ini berisi tentang RL, algoritma *Augmented Random Search*, simulasi pergerakan robot, dan *library* yang digunakan.

Bab III Perancangan Sistem

Bab ini berisi pembahasan mengenai perancangan dari Algoritma *Augmented Random Search* dan metode RL yang digunakan untuk menyelesaikan simulasi pergerakan robot multi-sendi.

Bab IV Implementasi

Bab ini membahas implementasi dari perancangan yang telah dibuat pada bab sebelumnya. Penjelasan berupa kode yang digunakan untuk proses implementasi.

Bab V Uji Coba Dan Evaluasi

Bab ini membahas tahapan uji coba, kemudian hasil uji coba dievaluasi terhadap kinerja dari sistem yang dibangun.

Bab VI Kesimpulan dan Saran

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

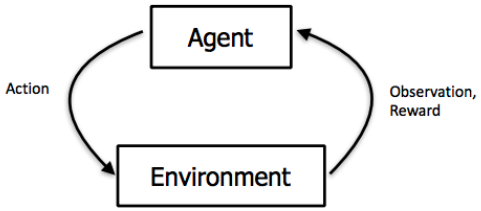
BAB II
TINJAUAN PUSTAKA

Bab ini membahas mengenai teori-teori dasar yang digunakan dalam Tugas Akhir. Teori-teori tersebut diantaranya adalah *Reinforcement Learning* (RL), *Environment* dalam RL, Mujoco simulation, dan beberapa teori lain yang mendukung pembuatan Tugas Akhir. Penjelasan ini bertujuan untuk memberikan gambaran umum dan diharapkan dapat mendukung sistem yang dibangun.

2.1 *Reinforcement Learning*

Reinforcement Learning (RL) merupakan salah satu cabang dari *Machine Learning*, yang berbeda dari dua cabang lain yaitu *supervised* dan *unsupervised learning.* Metode pada RL berfokus pada bagaimana membuat sebuah agen pintar yang dapat belajar dari apa yang agen itu lakukan pada sebuah lingkungan atau *environment* [2].

Agen berusaha mencapai sebuah tujuan dan setiap aksi yang dilakukan mendapat sebuah *feedback* berupa nilai *reward* atau *punishment* yang kemudian agen akan berusaha belajar untuk mendapat akumulasi *reward* yang maksimal atau dalam kasus lain mendapat akumulasi *punishment* yang minimum guna mencapai tujuannya.



Gambar 2.1 Hubungan antara *agent* dan *environment* reinforcement learning

Dalam prosesnya, RL bekerja secara sekuensial karena pada dasarnya untuk mengambil keputusan selanjutnya bergantung kepada *feedback* dari hasil keluaran atau *reward* sebelumnya seperti pada Gambar 2.1. Dalam mebuat metode RL, pembuat sistem tidak perlu memberi tahu agen atau bagaimana cara mencapai suatu tujuan. Agen akan mencari sendiri bagaimana cara mencapai tujuan tersebut tanpa perlu adanya masukan dari manusia atau bahkan tanpa perlu tahu apa jawaban yang benar dari suatu masalah, agen hanya memerlukan *reward/punishment* feedback dari apa yang ia lakukan.

Metode RL cocok untuk memecahkanmasalah dimana tidak mungkin manusia memberikan jawaban dan data yang tepat. Sebagai contohnya untuk membuat sistem yang mengatur berjalannya robot. Tidak mungkin kita sebagai manusia memberikan seluruh pengaturan cara berjalan untuk setiap kasus atau *state* yang mungkin ditemui robot saat berjalan, maka dari itu lebih baik kita membuat sistem yang bisa belajar melalui *trial and error* mengenai cara berjalan.

2.2 *Environment dalam Reinforcement Learning*

Environment merupakan komponen penting dalam pengembangan RL. *Environment* adalah komponen yang bertugas memberikan hasil observasi dari lingkungan simulasi kepada agen RL, menerima dan menjalankan output aksi yang diberikan oleh agen, menilai aksi yang dilakukan oleh agen dan memberikan nilai tersebut sebagai *reward feeback* ke pada agen atas aksi yang dilakukan [2].

Dalam pengembangan berbagai algoritma RL, sangat diperlukan komponen *environment* karena tidak mungkin dalam tahap pengembangan, sebuah algoritma langsung di implementasikan kedalam lingkukan fisik. Dalam lingkungan fisik,seperti implementasi langsung pada robot, akan sangan sulit menguji dan melihat hasil performa dari algoritman yang dikembangkan. Hal ini terjadi karena keterbatasan pada

lingkunganfisik, yaitu mahalnya biaya implementasi, terlalu banyaknya *variable* atau faktor yang bisa mempengaruhi hasi percobaan dan kendala fisik yang bisa memperlambat proses perulangan saat *training* agen.

2.2.1 Environment OpenAI Gym

OpenAI Gym merupakan pustaka pengembangan algoritma RL yang bersifat *open source* yang yang dibuat oleh tim OpenAI. Pustaka ini ditulis dalam bahasa pemrograman python yang banyak digunakna oleh peneliti *machine learning* dan *data sciences* untuk mengembangkan dan mengimplementasikan berbagai algoritma.

Gym dibuat untuk memudahkan pengembangan algoritma RL, karena salah satu kendala yang dialami oleh banyak peneliti adalah tidak adanya standara dalam pengujian dan pembandingan dari algoritma yang dibuat. Dengan tidak adanya *environment* standar, akansangat sulit mencoba dan mengulang hasil penitian yang dibuat oleh peneliti lalin. Pembuatan *environment* sendiri juga akan menambah waktu penelitian dan menghabiskan *resource* yang ada.

Dalam Gym terdapat berbagai *envirotnment* standar yang bisa langsung digunakan peneliti dan pengembangan algoritma RL. Gym juga menyediakan *interface* yang bisa digunakan untuk membuat *environment* baru apabila dibutuhkan. Sub bab 2.2.2 akan memberi contoh berbagai contoh *environment* simulasi yang disediakan oleh pustaka Gym.

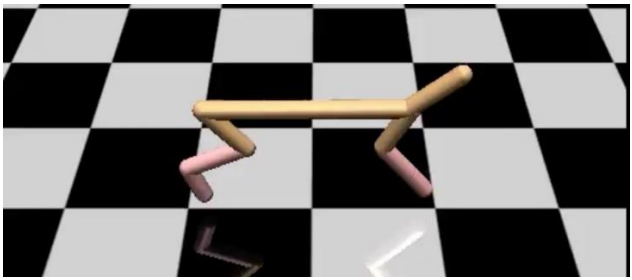
2.2.2 MuJoco Locomotion

MuJoCo merupakan singkatan dari *Multi Joint dynamics with Contact.* Mujoco merupakan *physics engine* yang digunakan untuk mensimulasikan pergerakan robot multi-sendi. *Engine* mujoco dapat memberikan simulasi fisika untuk berbagai bidang.

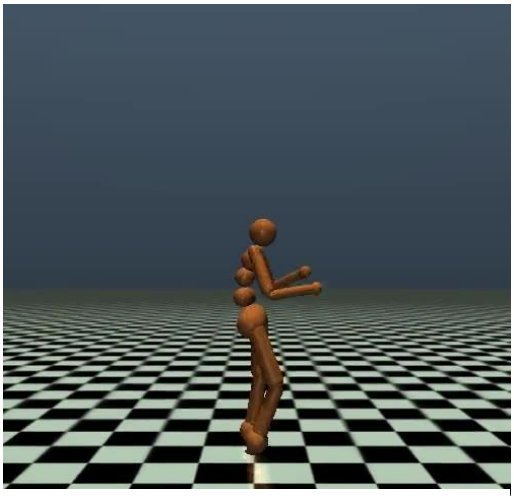
Dalam membuat tugas akhir ini, simulasi yang akan digunakan adalah simulasi pergerakan robot multi-sendi. Mujoco bertugas memberikan simulasi fisika dari pergerakan yang dilakukan oleh robot dalam lingkungan simulasi. Implementasi

algoritma *Augmented Random Search* akan diuji dalam beberapa bentuk simulasi, mulai dari bentuk robot sederhana yang bergerak seperti hewan, sampai bentuk robot yang berjalan dengan dua kaki seperti manusia

Dapat dilihat berdasarkan Gambar 2.2 dan Gambar 2.3, agen RL yang dibuat akan mempelajari bagaimana menggerakan robot dengan banyak sendi, bagaimana efek pergerakan setiap sendi dan bagaimana mencapai tujuannya.



Gambar 2.2 Simulasi robot HalfCheetah



Gambar 2.3 Simulasi robot humanoid

2.3 *Robot Multi-sendi*

Robot multi-sendi merupakan robot yang memiliki banyak sendi pada lengan geraknya [7]. Dengan adanya banyak sendi pada robot akan membuat gerakan yang mampu dilakukan oleh robot atau mesin menjadi jauh lebih fleksibel. Gerakan yang bisa dilakukan juga akan menjadi lebih presisi dan tepat. Kekurangannya mengontrol robot dengan banyak sendi akan jauh lebih kompleks dan sulit karena banyaknya bagian bisa digerakkan dan perlu pengaturan agar semua sendi dan lengan pada robot bisa bergerak sesuai yang diinginkan.

2.4 *Data Normalization*

Data Normalization atau nomalisasi data adalah teknik yang diaplikasikan saat menyiapkan data yang akan menjadi masukan suatu program *machine learning*. Normalisasi merupakan proses penyetaraan data tanpa merubah distribusi maupun rentang jarak nilai yang ada. Hal ini dilakukan agar seluruh bagian dari data memiliki bobot yang sama tanpa ada bias dari ukuran nilainya [8].

Dari berbagai teknik normalisasi, Algoritma ARS akan menggunakan teknik normalisasi *Z-Score* atau yang biasa disebut *standard score* [9]. Hasil Normalisasi (z) menggunakan *z-score* pada persamaan 2.1 memerlukan informasi data yang akan dinormalkan (x), rata-rata (μ) dan standar deviasi (σ) dari keseluruhan data yang ada.

$$z = \frac{x - \mu}{\sigma} \tag{2.1}$$

2.5 *Finite Differences*

Metode *Finite Differences* merupakan algoritma optimasi alternatif dari metode *Gradient Descent* yang biasa digunakan oleh algoritma *deep learning*. Pada dasarnya Metode ini bekerja dengan mencari perbedaan dari hasil derivatif dari nilai dikrit yang ada untuk mendapat perkiraan bentuk analisis dari suatu fungsi.

Metode *Finite Difference* dapat digunakan untuk mencari suatu perkiraan nilai optimum, sama seperti *Gradient Descent* [10].

Dalam algoritma ARS, metode *Finite Differences* akan digunakan dalam proses perbaruan matriks bobot. Rumus perbaruan matriks bobot algoritma ARS merupakan hasil *finite differences* dari reward *random search* yang ada dikalikan dengan konstanta dan dibagi dengan standard deviasinya.

2.6 *Algoritma Augmented Random Search*

Algoritma ARS merupakan pengembangan dari algoritma *Basic Random Search* yang dibuat untuk membuktikan algoritma sederhana juga mampu menyelesaikan permasalahan RL dan menyaingi performa algoritma-algoritma lain yang jauh lebih kompleks.

Pertama-tama jalannya algoritma ARS dimulai dengan menginisiai berbagai variable dan konstanta yang dibutuhkan. Akan dibuat juga matriks bobot yang berukuran sesuai dengan data masukan dan keluaran aksi yang dibutuhkan.

Kemudian algoritma akan membuat berbagai perubahan acak pada matriks bobot dan akan dinilai seberapa baik perubahan tersebut melalui *feedback* yang diberikan dari *environment* simulasi. Hal ini lah yang menjadi bagian *Random Search* dalam algoritma ARS.

Dalam usaha mendapatkan *feedback* dari *environment,* ARS menggunakan metode normalisasi *z-score* pada masukan hasil observasi *environment*nya.

Setelah mendapat *reward* dari berbagai perubahan yang dicoba, ARS akan mengurutkan perubahan dari yang terbaik ke yang terburuk dilihat dari nilai *reward*nya. Setelah terurut, ARS akan mengambil beberapa perubahan acak yang terbaik untuk melakukan perbaruan pada matriks bobotnya.

Algoritma ARS menggunakan modifikasi metode *finite differences* pada seluruh reward yang sudah diurutkan untuk menentukan pembaruan pada matrix *weight/policy* yang ada.

$$M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)}, \qquad (2.2)$$

Persamaan (2.2) merupakan fungsi untuk melakukan pembaruan pada matrix *weight*. Proses pembaruan berjalan dengan menghitung total dari hasil *finite differences* dari kumpulan reward yang didapatkan, dikali dengan konstanta *learning rates (alpha)* kemudian dibagi dengan standar deviasinya.

Tabel 2.1 Tabel 2.2 Penjelasan notasi rumus pembaruan matriks bobot (2.2)

Notasi	**Penjelasan**
M_{j+1}	Matriks bobot yang telah diperbarui
M_j	Matriks bobot
$r(\pi_{j,(k),+})$	Reward dari hasil *perturbation* positif
$r(\pi_{j,(k),-})$	Reward dari hasil *perturbation* negatif
$\delta_{(k)}$	*Perturbation*
σ_R	Standart deviasi
α	Konstanta
b	Banyaknya perturbation terbaik yang digunakan untuk perbaruan matriks bobot

2.7 *Pengukuran Performa Reinforcement Learning*

Pengukuran performa dari algoritma RL dapat diukur dengan menjalankan algoritma pada simulasi environment Gym. Gym akan memberikan nilai reward pada setiap aksi yang dilakukan pada simulasi, yang dimana reward tersebut akan dikumpulkan oleh program. Performa dari algoritma bisa diukur dalam jumlah iterasi untuk mencapai target rata-rata reward tertentu dan lama waktu per-iterasinya.

Target atau *Threshold* dari rata-rata rewardnya didapatkan dengan menggunakan 90% dari nilai akhir yang didapatkan dengan menjalankan *stochastic linear policy*. Tabel 2.2 merupakan hasil akhir *stochastic linear policy* yang dijalankan pada environment simulasi Gym [11].

Tabel 2.3 Reward Maksimal dan Threshold untuk *Benchmarking* [11]

Mujoco Task	**Final Reward**	**Threshold**
Swimmer	362	325
Hopper	3466	3120
Cheetah	3810	3430
Walker	4881	4390
Ant	3980	3580
Humanoid	5873	5280

Perlu diperhatikan bahwa reward pada Tabel 2.1 menggunakan pustaka simulasi yang berbeda dengan implementasi pada tugas akhir ini, sehingga threshold nilai pada pengujian akan disesuaikan.

2.8 *Python*

Python adalah bahasa pemograman yang populer. *Python* sering dimanfaatkan dalam pengembangan web, perangkat lunak, penelitian, dan *system scripting*. Python dapat digunakan untuk

menangani data besar dan melakukan operasi matematika yang kompleks. Python bekerja di berbagai *platform* seperti Windows, Mac, Linux, Raspberry Pi, dan lain-lain. Python dirancang untuk mudah dibaca, yaitu memiliki sintaks yang sederhana dan menggunakan bahasa inggris [12].

2.9 *Pybullet*

Pybullet adalah modul python yang berguna untuk menyediakan simulasi fisika, robotika dan *deep reinforcement learning* yang dibuat berdasarkan *Bullet Physics SDK*. dapat memuat model tubuh dari URDF, SDF dan format file lainnya. Pybullet menyediakan simulasi dinamika maju, perhitungan dinamika terbalik, kinematika maju dan terbalik dan deteksi tabrakan dan query persimpangan Ray. Selain simulasi fisika, Pybullet mendukung rendering, dengan renderer CPU dan visualisasi OpenGL dan dukungan untuk headset realitas virtual [13].

2.10 *Python Multiprocessing*

Ray merupakan salah satu modul tambahan dari bahasa pemrograman python. Modul ini berguna untuk membuat *process* tambahan saat program python dijalankan yang dimana akan digunakan untuk membagi tugas yang dikerjakan sehingga waktu proses menjadi lebih cepat dan juga lebih efisien dengan memanfaatkan berbagai *core* yang tidak dipakai dalam *processor* yang memiliki *multi-core* [14].

2.11 *Numpy*

Numpy adalah *library Python* yang mendukung pengolahan data pada *array* dan matriks multidimensi yang besar. Numpy menyediakan kumpulan fungsi matematika, seperti aljabar linear, transformasi Fourier, pembuatan angka acak, dan lain-lain. *Numpy* bersifat *open source* sehingga banyak dimanfaatkan dalam pengolahan data peneilitian [15].

(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN SISTEM

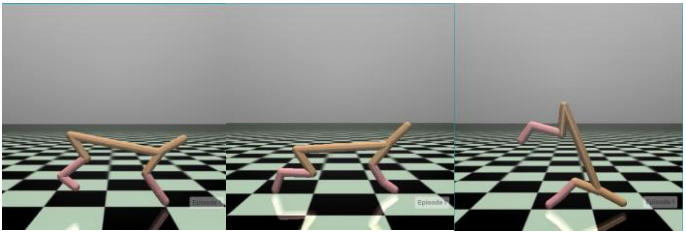
Bab ini menjelaskan mengenai perancangan data dan sistem RL dengan algoritma ARS yang mampu menyelesaikan simulasi pergerakan robot. Bab ini juga akan menjelaskan gambaran umum sistem dalam bentuk diagram alir.

3.1 *Perancangan Data*

Data yang akan digunakan sebagai pengujian awal dari sistem pergerakan robot adalah data simulasi yang disediakan oleh pustaka machine learning Gym. Salah satu simulasi yang akan digunakan adalah simulasi pergerakan robot berkaki dua yang bergerak seperti cheetah, yang berjudul *HalfCheetah.* Gambar 3.1 merupakan tampilan simulasi *halfCheetah.*

Sistem akan menerima hasil observasi dari simulasi yang akan berbentuk matriks yang memberikan informasi mengenai posisi dan kondisi model pada dunia simulasi, yang kemudian akan di proses dan menghasilkan nilai keluaran berupa aksi yang akan dilakukan robot. *Environment* akan memberikan nilai *feedback* dari aksi yang sudah dilakukan robot kepada agen/sistem.

Perhitungan nilai *reward* yang akan dilakukan oleh *environment* akan meliputi beberapa aspek berikut, yaitu nilai *survival* dari robot, nilai progres pergerakan robotnya, *electricity cost* yang disimulasikan dari pergerakan sendi, nilai penalti untuk sendi yang *stuck* atau tersangkut, dan juga nilai pinalti apa bila kaki



Gambar 3.1 simulasi *halfcheetah*

dari robotnya menyentuh objek lain atau badannya sendiri. Nilai-nilai ini akan diakumulasikan dan menghasilkan total reward dari setiap aksi atau pergerakan yang dilakukan oleh robot.

Spesifikasi dari simulasi yang akan dijalankan terdiri dari 3 hal, yaitu panjang simulasi, jumlah observasi, dan jumlah sendi. Panjang simulasi diukur dalam berapa banyak aksi yang dilakukan (episode atau frame). Jumlah observasi adalah banyaknya nilai yang diberikan oleh *environment* dari hasil observasi suatu simulasi. Spesifikasi jumlah sendi adalah banyaknya sendi pada robot yang bisa beri nilai untuk digerakan.

3.1.1 Simulasi HalfCheetah

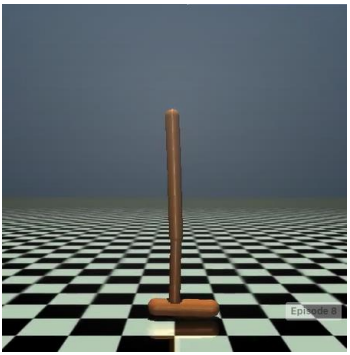
Tujuan dari simulasi *HalfCheetah* adalah agar robot bisa tidak jatuh dan juga dapat bergerak kearah kanan/depan secepat mungkin. Semakin cepat robot bergerak maka *reward* atau *feedback* yang diberikan *environment* akan semakin tinggi. Spesifikasi dari simulasi *HalfCheetah* bisa dilihat pada Tabel 3.1.

Tabel 3.1 Spesifikasi *halfcheetah*

Keterangan	**Spesifikasi**
Panjang Simulasi	1000 episode
Jumlah observasi	26 nilai
Jumlah sendi	6 sendi

3.1.2 Simulasi Hopper

Tujuan simulasi *hopper* hampir sama dengan *halfcheetah,* yaitu untuk robot bergerak maju secepat dan se-efisien mungkin. Yang membedakan adalah cara robot ini bergerak. Simulasi robot *hopper* berbentuk sebuah kaki yang bergerak maju dengan cara melompat yang dapat dilihat pada Gambar 3.1. Spesifikasi dari simulasi *Hoppper* bisa dilihat pada Tabel 3.2.

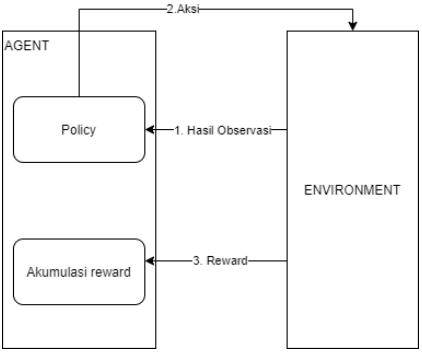


Gambar 3.2 Simulasi robot *hopper*

Tabel 3.2 Spesifikasi *Hopper*

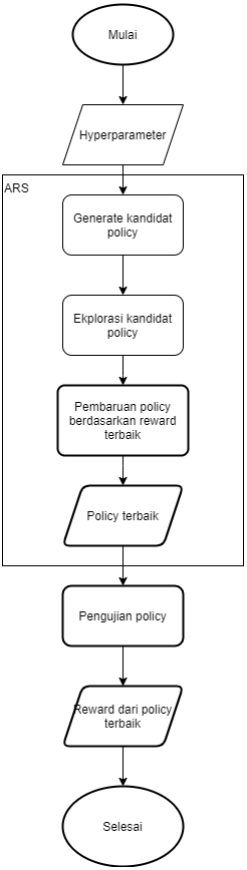
Keterangan	**Spesifikasi**
Panjang Simulasi	1000 episode
Jumlah observasi	15 nilai
Jumlah sendi	3 sendi

3.2 *Desain Umum Sistem*



Gambar 3.3 Interaksi ARS dengan *Agent* dan *Environment*

Proses yang terdapat pada pada Gambar 3.3 merupakan gambaran umum interaksi dari sistem ARS yang dibuat dengan *environment* simulasinya. *Agent* yang di dalamnya terdapat *policy* dari ARS menerima masukan hasil observasi dari lingkungan simulasi. Setelah menerima hasil observasi, *policy* akan memberikan keluaran aksi yang sesuai dengan masukan yang



Gambar 3.4 Diagram alir algoritma ARS

diterima. Aksi akan diterima oleh *environment* yang kemudian aksi

tersebut akan dijalankan pada lingkungan simulasi. Setiap aksi yang dilakukan pada lingkungan simulasi akan ada *feedback* nilai yang menunjukan apakah aksi tersebut sesuai dengan tujuan simulasi atau tidak. *Feedback* yang dikumpulkan oleh ARS guna melakukan perbaruan *policy* yang ada

Sistem implementasi algoritma ARS yang dibangun memiliki proses utama diantaranya memberi masukan *hyperparameter*, Generate kandidat *policy*, ekplorasi kandidat *policy*, dan pembaruan *policy* berdasarkan reward terbaik. Diagram alir yang lebih spesifik menjelaskan proses algoritma ARS dapat dilihat pada diagram alir Gambar 3.4.

3.2.1 *Hyperparameter*

Algoritma ARS membutuhkan berbagai parameter sebelum algoritma dapat berjalan dengan baik. Parameter yang dibutuhkan salah satunya, jumlah iterasi, jumlah *random perturbation* yang akan dicoba, tipe simulasi yang akan dijalankan dan juga berbagai *variable* lain yang diperlukan. Penjelasan lebih jelas mengenai *Hyperparameter* akan ada pada bagian implementasi ARS.

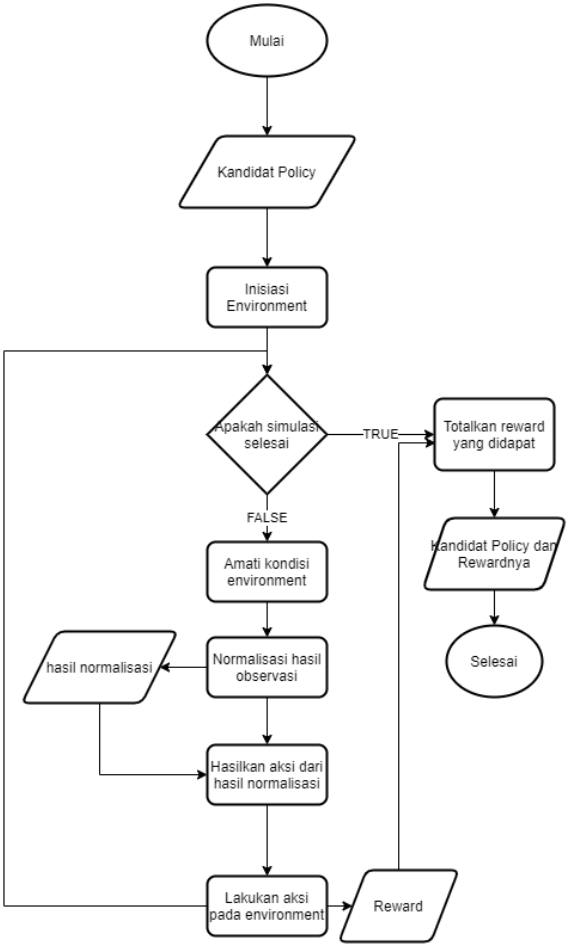
3.2.2 Generate Kandidat *Policy*

Dasar dari algoritma ARS adalah *Basic Random Search*, jadi algoritma ARS belajar melalui berbagai percobaan acak yang dilakukan. Maka dari itu dalam algoritma ARS, setiap iterasi akan dibuat matrix dengan jumlah yang sudah ditentukan yang berukuran sama seperti matrix *policy* yang berisi nilai random dengan distribusi normal seperti pada diagram Gambar 3.5.

Setiap iterasi akan dilakukan membuat kandidat *Policy* dengan menambahkan maupun mengurangkan matrix *policy* dengan matrix acak yang sudah dibuat yang kemudian akan di evaluasi hasilnya.

Gambar 3.5 Diagram alir proses generate kandidat *Policy*

3.2.3 Eksplorasi Kandidat Policy



Gambar 3.6 Diagram alir ekplorasi kandidat *Policy*

Untuk mendapatkan reward dan mengevaluasi matrix acaknya, simulasi robot harus dijalankan dengan matrix policy yang sudah dimodifikasi dengan matrix acak. Setiap langkah atau

gerakan yang dilakukan robot akan dinilai oleh *environment* dan kemudian akan memberikannya kepada sistem. Diagram 3.6 menggambarkan jalannya proses eksplorasi *policy* atau matriks bobot pada lingkungan simulasi.

Penilaian dan perhitungan reward dari setiap aksi yang dilakukan akan ditangani oleh *environment*-nya, karena setiap simulasi menggunakan model dan perhitungan yang berbeda. Simulasi akan dijalankan sampai robot terjatuh atau mencapai limit dari simulasinya.

Hasil dari observasi *environment* perlu dilakukan normalisasi. Normalisasi diperlukan untuk menyetarakan bobot dari setiap komponen hasil observasi, karena *range* nilai yang berbeda-beda dari setiap komponen mampu mendominasi perhitungan saat *training* menghasilkan bias kepada *policy* yang sedang di *training.*

Teknik yang digunakan dalam algoritma ARS untuk normalisasi adalah mengurangi hasil observasi dengan rata-rata seluruh hasil observasi dan kemudian dibagi dengan standart deviasinya. Tahap normalisasi ini hanya akan di implementasikan pada versi 2 dan variasi dari versi 2 algoritma ARS

3.2.4 Pembaruan *Policy* Berdasarkan *Reward* Terbaik

Hasil dari banyak percobaan akan dikumpulkan dan diurutkan dari reward yang terbesar ke nilai yang terkecil (*descending)*. Hasil pengurutan ini kemudian akan dipasangkan dengan matrix acak yang sesuai.

Dalam algoritma ARS , akan diambil beberapa nilai terbaik dari hasil yang sudah terurut. Hal ini dilakukan untuk membuang hasil percobaan yang buruk agar proses pembaruan matrix *policy* menjadi lebih efisien.

Pada dasarnya, algoritma RL merupakan algoritma yang dibuat agar *agent* atau sistem pengambilan keputusan dapat belajar melalui *feedback* dari aksi yang sudah dilakukan. Sesuai pada

Gambar 3.7, proses ini membutuhkan kandidat *Policy* yang sudah terurut sesuai dengan *reward* yang didapat pada proses eksplorasi. Pada tahap ini lah algoritma ARS belajar dari aksi yang sudah dilakukan.

Dalam tugas akhir ini, hanya implementasi dari algoritma ARS versi 2 dan variasi dari versi 2 yang akan dilakukan pembagian berdasarkan standar deviasi pada tahap perbaruan matriks bobot.

Gambar 3.7 Proses Pembaruan Policy

3.2.5 Pengujian *Policy*

Proses pengujian *policy* bertujuan agar bisa melihat bagaimana perkembangan *policy* selama proses *training*. *Policy* yang sudah diperbarui akan dijalankan pada lingkungan simulasi yang ada. Simulasi akan dijalankan sampai batas simulasi tercapai atau sampai robot dalam simulasi terjatuh. *Reward* dari setiap aksi yang dilakukan akan diakumulasikan, dengan begitu akan terlihat apakah *policy* berkembang menjadi lebih baik atau tidak.

BAB IV
IMPLEMENTASI

Bab ini menjelaskan mengenai implementasi perangkat lunak dari rancangan sistem yang telah dibahas pada Bab 3 meliputi kode program dalam perangkat lunak. Selain itu, implementasi dari tiap proses, parameter masukan, keluaran, dan beberapa keterangan yang berhubungan dengan program juga dijelaskan.

4.1 *Lingkungan Implementasi*

Dalam mengimplementasikan algoritma *Augmented Random Search Reinforcement Learning* diperlukan beberapa perangkat pendukung sebagai berikut.

4.1.1 Perangkat Keras

Implementasi tugas akhir ini menggunakan Laptop Asus X450-JN. Sistem operasi yang digunakan adalah Windows 10 64-bit. PC yang digunakan memiliki spesifikasi Intel Core i7-4710HQ dengan kecepatan 3,6 GHz, *Random Access Memory* (RAM) sebesar 8 GB, dan mempunyai *Graphics Processing Unit* (GPU) yaitu NVIDIA GeForce 840M sebesar 2 GB.

4.1.2 Perangkat Lunak

PC dari sisi perangkat lunak memiliki spesifikasi antara lain menggunakan bahasa pemograman Python 3.6, dilengkapi dengan *library* antara lain Gym, Pybullet, Ray, dan Numpy

4.2 *Implementasi Kelas Hyperparameter*

Untuk mengatur penggunaan berbagai parameter-parameter penting yang akan digunakan oleh berbagai modul, maka dibuat sebuah kelas yang menampung keseluruhan *Hyperparameter* yang ada. Objek dari kelas ini akan di gunakan oleh seluruh modul yang

membutuhkan.

```
1.  def __init__(self, nb_steps = 5, episode_length = 1000, learning_rate= 0.02, nb_direc-
    tions = 16, nb_best_directions = 16, noise = 0.03, seed = 1, env_name = "HalfChee-
    tahBulletEnv-v0", shift = 0):
2.      self.nb_steps = nb_steps
3.      self.episode_length = episode_length
4.      self.learning_rate = learning_rate
5.      self.nb_directions = nb_directions
6.      self.nb_best_directions = nb_best_directions
7.      assert self.nb_directions >= self.nb_best_directions
8.      self.noise = noise
9.      self.seed = seed
10.     self.env_name = env_name
11.     self.shift = shift
```

Kode Sumber 4.1inisiasi kelas *hyperparameter*

Kode sumber 4.1 merupakan fungsi yang akan digunakan untuk menginisiasi *hyperparameter* yang akan digunakan sistem. Pada Tabel 4.1 akan menjelaskan berbagai *hyperparameter* yang digunakan beserta kegunaannya.

Tabel 4.1 Jenis-jenis *hyperparameter*

Hyperparameter	**Penjelasan**
Nb_steps	Jumlah perulangan yang dilakukan selama training
Episode_length	Batas maksimum simulasi
Nb_directions	Jumlah random *perturbation* yang dibuat setiap iterasi
nb_best_directions	Banyaknya *perturbation* terbaik yang diambil untuk pembaruan matriks bobot
noise	Konstanta random yang dikalikan dengan *perturbation*
Seed	Seed untuk memilih random
Env_name	Nama *environment* atau nama simulasi yang dijalankan

Hyperparameter	Penjelasan
Shift	Angka untuk meniadakan *survival reward* pada simulasi

4.3 *Implementasi Normalizer*

Pada subbab ini akan dijabarkan implementasi Normalizer yang berfungsi untuk melakukan normalisasi setiap data hasil observasi *environment* yang masuk.

4.3.1 Menghitung Rata-rata dan Variance

Data hasil observasi yang masuk pertama-tama perlu dihitung ulang rata-rata dari keseluruhan data yang sudah pernah di observasi, kemudian perbedaan dari kedua rata-rata tersebut akan digunakan untuk variannya.

```
1.  def observe(self, x):
2.          """ observe new input value and calculate appropriate mean and variance"""
3.      self.n += 1.
4.      last_mean = self.mean.copy()
5.      self.mean += (x - self.mean) /self.n
6.      self.mean_diff += (x - last_mean) * (x - self.mean)
7.      self.var = (self.mean_diff / self.n).clip(min = 1e-2)
```

Kode Sumber 4.2 fungsi untuk menghitung rata-rata dan varians

Pada kode sumber 4.2 baris 4 dan 7 berfungsi untuk menghitung rata-rata dan menghitung varians dari seluruh data yang masuk pada sistem.

4.3.2 Normalisasi Hasil Observasi

Dengan nilai rata-rata dan varians yang sudah dihitung pada tahap 4.2.1, rumus yang digunakan untuk normalisasi datanya adalah mengurangi hasil observasi dengan nilai rata-ratanya dan kemudian membaginya dengan nilai varians yang sudah didapatkan.

```
1. def normalize(self, inputs):
2.     """ normalize input"""
3.     obs_mean = self.mean
4.     obs_std = np.sqrt(self.var)
5.     return (inputs - obs_mean) / obs_std
```

Kode Sumber 4.3 fungsi untuk normalisasi input

Baris ke 4 berfungsi untuk menghitung standar deviasi dari data yang masuk. Sedangkan baris 5 merupakan implementasi dari rumus normalisasi *z-score*.

4.4 *Implementasi Policy*

Kelas Policy bertanggung jawab untuk melakukan pengambilan keputusan aksi yang dilakukan berdasarkan input yang masuk, membuat kumpulan matrix *perturbation*, dan melakukan perbaruan pada matrix bobot pengambilan keputusannya.

Kelas Policy di inisiasi dengan membuat matriks bobot yang kosong dengan ukuran yang sesuai dengan masukan dan keluaran yang dibutuhkan.

4.4.1 Implementasi Pengambilan Keputusan

Pada tahap ini kelas Policy akan mengevaluasi masukan hasil observasi yang sudah di normalisasi dan akan memberikan keluaran yang dibutuhkan.

Keluaran aksi yang dibutuhkan akan didapatkan dengan cara melakukan perkalian matriks masukan dengan matriks bobot atau *policy* yang ada yang diimplementasikan pada baris 4, 6, dan 8.

```
1. def evaluate(self, input, delta = None, direction = None):
2.     """ Evaluate input value against theta/weight matrix and return output matrix"""
3.     if direction is None:
4.         return self.tetha.dot(input)
5.     elif direction == 'positive':
6.         return (self.tetha + self.hp.noise * delta).dot(input)
7.     else:
8.         return (self.tetha - self.hp.noise * delta).dot(input)
```

Kode Sumber 4.4 fungsi *evaluate* untuk memproses masukan dan menghasilkan keluaran aksi.

Parameter *delta* akan diisi dengan matriks *perturbation* yang akan ada pada tahap *random search* dari training yang dilakukan.

4.4.2 Implementasi Pembuatan Matriks Pertrubation

Matriks *perturbation* akan dibuat sejumlah angka yang sudah kita tentukan pada awal program. Matriks ini akan dibuat dengan ukuran yang sama seperti matriks bobot yang ada. Matriks *perturbation* akan berisi angka random dengan distribusi normal.

```
1. def sample_deltas(self):
2.     """ generate random samples"""
3.     return [ np.random.randn(*self.tetha.shape) for _ in range(self.hp.nb_direc-
   tions)]
```

Kode Sumber 4.5 fungsi *sample_deltas* untuk membuat kumpulan matriks *perturbation* random.

Fungsi *sample_deltas* pada baris 1 merupakan fungsi sederhana untuk membuat matriks acak dengan ukuran dimensi matriks yang sama dengan matriks bobot

4.4.3 Implementasi Proses Perbaruan Matriks Bobot

Sesuai dengan rumus pada sub-bab 3.2.5, perbaruan pada matriks bobot akan dimulai dengan menghitung total dari reward dikali dengan matriks *perturbation*-nya kemudian matriks bobot akan ditambahkan dari kalkulasi dari *learning rates* dikali total perhitungan sebelumnya dan dibagi dengan standart deviasinya.

```
1. def update(self, rollouts, sigma_r):
2.     """updating theta or weight matrix value"""
3.     step = np.zeros(self.tetha.shape)
4.
5.     for r_pos, r_neg, d in rollouts:
6.         step +=   (r_pos - r_neg) * d
7.
8.     self.tetha += self.hp.learning_rate / (self.hp.nb_best_direc-
   tions * sigma_r) * step
```

Kode Sumber 4.6 fungsi *update* untuk melakukan perbaruan matriks bobot

Baris 8 adalah implementasi dari rumus 2.2 yang berfungsi untuk melakukan perbaruan pada matriks *policy* menggunakan hasil eksplorasi yang sudah dilakukan.

4.5 *Implementasi Modul Worker*

Kelas Worker akan bertanggung jawab mengenai proses eksplorasi *environment* dan juga mengumpulkan reward yang didapat. Dengan kelas ini, pengaturan untuk *multiprocessing* akan lebih mudah. Kelas ini dapat digunakan saat evaluasi *perturbation* dan juga evualuasi dari hasil perubahaan matriks bobot.

4.5.1 Fungsi Eksplore

Dalam kelas Worker, terdapat fungsi explore yang memiliki fungsi untuk melakukan eksplorasi di *environment* dan mendapatkan akumulasi rewardnya sampai batas dari *environment* yang kita tentukan tercapai atau robot dalam simulasi terjatuh.

```
1.  def explore(self,normalizer, policy, direction = None, delta = None):
2.          """explore environment to one specific pertubation value"""
3.          state = self.env.reset()
4.          done = False
5.          num_plays = 0.
6.          sum_rewards = 0
7.          while not done and num_plays < self.hp.episode_length:
8.              state = ray.get(normalizer.observe.remote(state))
9.              action = policy.evaluate(state, delta, direction)
10.             state, reward, done, _ = self.env.step(action)
11.
12.             sum_rewards += (reward - self.hp.shift)
13.             num_plays += 1
14.         return sum_rewards
```

Kode Sumber 4.7 fungsi *explore* untuk menjalankan simulasi sesuai matriks bobot dan *perturbation* yang ada.

Di fungsi explore pada baris 1, kita juga dapat menentukan kearah mana *perturbation*nya akan dilakukan, *negative* (mengurangi matriks bobot dengan *perturbation)* atau *positive (* menambahkan matriks bobot dengan *perturbation)*. Hasil observasi atau data masukan akan dinormalisasi pada baris 8 menggunakan *normalizer* yang sudah diimplementasikan.

4.6 *Modul ARS*

Modul ARS merupakan modul utama yang akan menjalankan proses *training* dalam algoritma *Augmented Random Search.* Modul ini akan membuat dan menjalankan objek dari modul-modul yang dibutuhkan selama proses berjalannya program.

Kelas ARS pertama-tama akan melakukan inisiasi berbagai variable serta modul yang dibutuhkan. Kemudian akan menjalankan proses *training* seperti yang ada pada diagram alir 3-3. Kelas ini juga akan bertanggung jawab tentang pencatatan hasil *training* dan juga pecatatan *reward* pada setiap iterasi yang berguna untuk evaluasi.

(Halaman ini sengaja dikosongkan)

BAB V
UJI COBA DAN EVALUASI

Bab ini akan membahas mengenai hasil uji coba sistem yang telah dirancang dan dibuat. Uji coba dilakukan untuk mengetahui kinerja sistem dengan lingkungan uji coba yang telah ditentukan.

5.1 *Lingkungan Uji Coba*

Lingkungan uji coba pada tugas akhir ini adalah sebuah laptop Asus X450-JN. Sistem operasi yang digunakan adalah Windows 10 64-bit. PC yang digunakan memiliki spesifikasi perangkat keras Intel Core i7-4710HQ dengan kecepatan 3,6 GHz, *Random Access Memory* (RAM) sebesar 8 GB, dan mempunyai *Graphics Processing Unit* (GPU) yaitu NVIDIA GeForce 840M sebesar 2 GB. Pada sisi perangkat lunak, uji coba pada tugas akhir ini dilakukan dengan menggunakan bahasa pemograman Python 3.6 dilengkapi dengan *library* antara lain Gym, Pybullet, Ray, dan Numpy.

5.2 *Dataset*

Pada tugas akhir ini, data pengujian yang akan digunakan adalah berbagai simulasi yang disediakan oleh pustaka gym. Dari berbagai bentuk simulasi yang disediakan, tugas akhir ini akan menggunakan 3 simulasi yang ada, yaitu *Hopper, HalfCheetah* dan *Walker2D*. Pemilihan simulasi dilihat dari kompleksitas dan waktu yang dibutuhkan untuk melakukan *training* dari sistem yang telah dibuat. *Hopper* dan *HalfCheetah* merupakan simulasi yang relativ sederhana, karena itu pengujian akan menggunakan target *threshold* tertentu dan melihat berapa rata-rata episode atau iterasi yang dibutuhkan untuk mencapai target tersebut. Sedangkan *Walker2d* merupakan simulasi yang lebih kompleks dan membutuhkan jauh lebih banyak iterasi, sehingga pengujian yang dilakukan hanya akan mencatat berapa rata-rata *reward* yang didapatkan dalam jumlah iterasi tertentu.

5.3 *Target Reward Uji Coba*

Target *reward* yang perlu dicapai untuk menyatakan bahwa sistem berhasil menyelesaikan simulasi akan didasarkan pada Tabel 2.1. Namun perlu diperhatikan bahwa pada tugas akhir ini akan menggunakan *physics library* yang berbeda (*Pybullet*) sehingga perhitungan *reward* dari simulasi juga akan dan lebih sulit [16].

Algoritma ARS yang dibuat juga tidak akan menghitung *survival reward* yang diberikan oleh *environment. Threshold reward* akan disesuaikan dengan cara mengurangi *threshold* sebanyak total *survival reward* yang bisa didapatkan pada setiap jenis simulasi, dan juga mengurangi *threshold* sebanyak 500 poin untuk mengakomodasi tambahan kesulitan yang ada.

Threshold goal pada Tabel 5.1 yang digunakan didapatkan langsung dari kode sumber pustaka *pybullet* yang digunakan. Proses pengujian akan dibuat batasan pada jumlah iterasi yang dijalankan untuk menghemat *resource* komputer dan waktu yang ada. Apa bila dalam 2000 iterasi hasil *training* belum mencapai *adjusted reward* maka akan dilihat bagaimana progress yang telah pencapaian dari tujuan simulasi yang dapat dilihat dari video rekaman simulasinya.

Tabel 5.1 *threshold reward* yang akan digunakan dalam pengujian

Simulasi	***Goal***	***Threshold***	***Survival Bonus***	***Adjusted reward***	**batas jumlah iterasi**
Hopper	2500	2250	1000	1250	2000
Cheetah	3000	2700	1000	1700	2000
Walker2D	2500	2250	1000	1250	2000

5.4 *Skenario Uji Coba*

Tabel 5.2 Parameter awal yang digunakan dalam implementasi ARS

Keterangan	**Parameter**
Jumlah *episode(*iterasi)	1000
Panjang simulasi	1000
Noise	0,03
Random Directions	*10*
Best directions	*10*
Learning rate	0,02

Proses uji coba berguna untuk menemukan parameter yang menghasilkan performa dari algoritma ARS yang paling optimal. Parameter yang tepat akan memberikan hasil yang lebih baik pada saat proses uji coba. Hasil terbaik dari suatu skenario uji coba akan digunakan untuk skenario uji coba berikutnya. Ada 3 macam skenario uji coba yang akan dijalankan pada implementasi algoritma ARS yang telah dibuat. Skenario uji coba yang akan dilakukan yaitu:

1. Uji Coba implementasi berbagai versi ARS
2. Uji Coba *Hyperparameter* dari ARS
3. Uji Coba performa ARS
4. Perbandingan Algoritma ARS dengan Algoritma A2C

Keterangan	**Parameter**
Jumlah *episode(*iterasi)	1000
Panjang simulasi	1000
Noise	0,03
Random Directions	*10*
Best directions	*10*

Learning rate	0,02

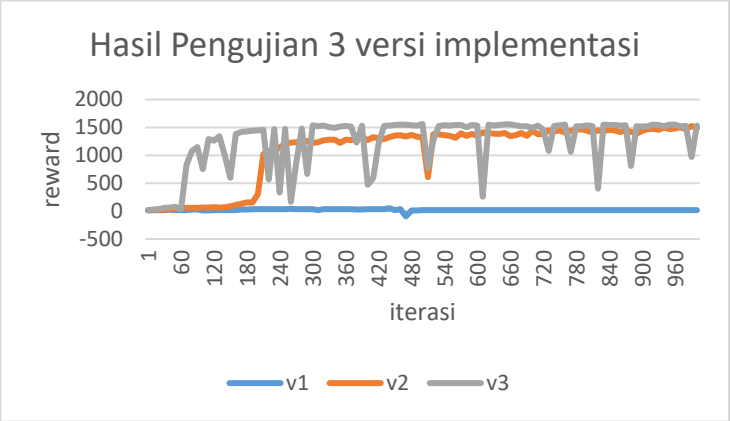
Tabel 5.2 berisi parameter-parameter awal implementasi dari ARS yang digunakan dan dapat berubah di setiap uji coba yang dilakukan. Pada setiap skenario uji coba akan ditetapkan nilai parameter yang dapat meningkatkan kinerja arsitektur.

5.4.1 Uji Implementasi Berbagai Versi ARS

Uji coba pertama adalah uji coba berbagai versi ARS yang telah diimplementasikan. Dari berbagai versi yang ada akan dilihat apakah dapat menyelesaikan simulasi yang ada dan bagaimana performanya dibanding dengan versi implementasi yang lain. Akan ada 3 versi dari algoritma ARS yang diimplementasikan, yaitu:

1. Versi pertama adalah versi paling sederhana dari algoritma ARS. Dimana versi ini tidak akan menggunakan *Normalizer* dalam proses *training*nya.
2. Versi Kedua adalah penyempurnaan dari versi pertama. Perbedaannya ada pada ditambahkannya *Normalizer* dari data hasil observasi yang masuk dan juga proses pembaruan matriks bobot yang akan dibagi dengan deviasi standarnya.
3. Versi ketiga merupakan variasi dari versi kedua. Hanya ada sedikit perubahan pada versi ini. Pada versi ini hanya akan diambil sebagian *random perturbation* terbaik pada setiap iterasinya untuk digunakan dalam proses pembaruan matriks. Tidak seperti versi kedua yang akan menggunakan keseluruhan dari *random perturbation* yang dicoba.

Simulasi yang akan digunakan dalam menguji ketiga versi yang diimplementasikan adalah *Hopper*. Implementasi ini dipilih karena *Hopper* tergolong sederhana dan hanya memerlukan waktu yang lebih sedikit untuk *training policy* atau matriks bobotnya.



Gambar 5.1 Hasil pengujian berbagai versi implementasi dari ARS

Dapat dilihat berdasarkan grafik pada Gambar 5.1, performa versi 1 algoritma ARS sangatlah buruk dan membutuhkan jauh lebih banyak iterasi untuk mencapai target *reward* yang besar. Sedangkan versi 2 dan 3 tidak terlalu berbeda namun versi 3 mencapai *reward* yang tinggi sedikit lebih cepat dibandingkan versi 2. Hal ini karena versi 3 menggunakan metode yang lebih efisien dalam melakukan proses pembaruan matriks bobot.

5.4.2 Uji Coba *Hyperparameter* dari ARS

Uji Coba ini akan menguji bagaimana pengaruh berbagai *hyperparameter* yang ada pada implementasi ARS ini dalam segi performa dan hasil *training*nya. *Hyperparameter* yang akan dicoba untuk divariasikan antara lain:

1. *Random Directions*

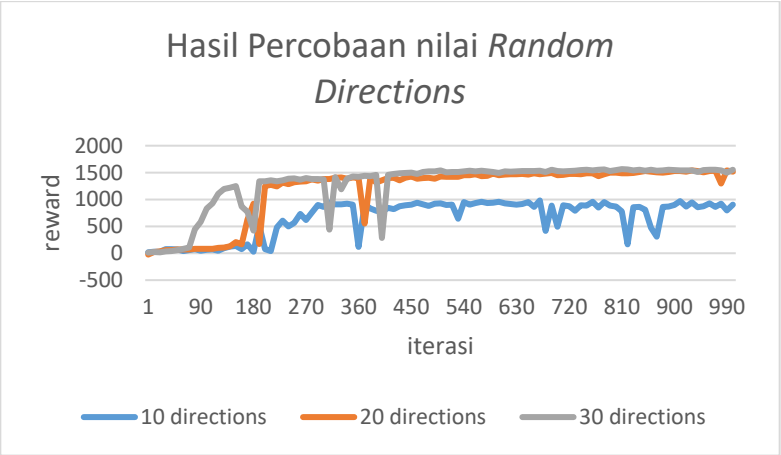
2. *Best Directions*
3. *Learning Rate*

Tabel 5.3 berisi *hyperparmeter* apa saja yang tidak akan berubah selama pengujian berserta nilainya.

Tabel 5.3 *Hyperparameter* yang tidak akan berubah selama pengujian

Keterangan	**Parameter**
Jumlah *episode(*iterasi)	1000
Panjang simulasi	1000
Noise	0,03
Simulasi	*Hopper*

Pengujian *random directions* akan melihat bagaimana perubahan jumlah *random perturbation* dengan perbandingan jumlah *best directions* yang tetap dapat mempengaruhi hasil dan waktu dari *training.* Jumlah *best directions* yang digunakan disini adalah setengah dari jumlah *random perturbation* yang diujikan.



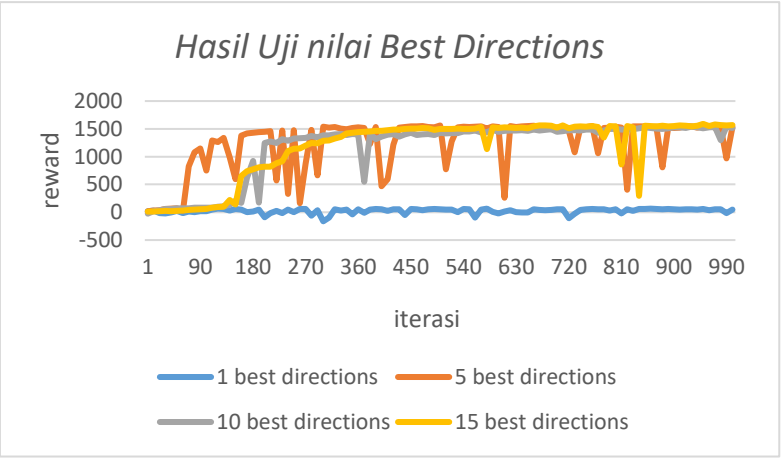
Gambar 5.2 hasil percobaan variasi nilai *random directions*

Tabel 5.4 hasil percobaan variasi parameter *random directions*

Jumlah *directions*	Rata-rata 100 reward terakhir	Waktu training (Jam)
10	824.946	2.86
20	1499.38	5.98
30	1524.12	9.8

Dapat dilihat pada Tabel 5.4 dan diagram pada Gambar 5.2 bahwa penambahan jumlah *random directions* dapat memberikan hasil yang lebih baik namun menambah waktu yang dibutuhkan untuk *training*. Hal ini dikarenakan jumlah percobaan yang dilakukan pada setiap iterasi akan bertambah. Lama dari waktu *training* ini bisa dikurangi dengan cara memberikan sumber daya CPU tambahan pada program, karena dengan begitu percobaan dari *random directions*nya bisa diparalel antar CPU yang ada.

Pengujian *Best Directions* melihat bagaimana pengaruh perbandingan antara *random directions* dan *best directions* pada performa sistem. Menggunakan nilai *random directions 20* yang

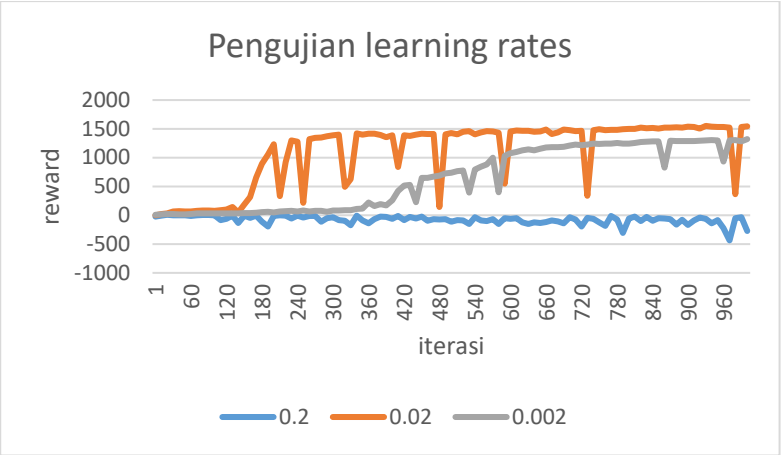


Gambar 5.3 Hasil percobaan variasi nilai parameter *best directions*

dinilai paling efisien dan sesuai dengan *resource* yang digunakan saat pengujian.

Hasil dari pengujian pada Gambar 5.3 menunjukan bahwa dengan mengambil hanya satu *random perturbation* yang terbaik akan membuat performa sistem sangat buruk dan mengambil hampir dari keseluruhan *random perturbation* (3/4 dari jumlah *random directions)* juga akan performanya memburuk. Dengan nilai *best directions* setengah atau kurang dari *random* directions akan menghasilkan performa yang baik pada sistem. Perlu diperhatikan bahwa dengan nilai *best directions* 1 performa sistem menjadi sangatlah buruk dikarenakan proses pembaruan matriks tidak mendapat cukup informasi untuk memberikan hasil yang baik.

Percobaan yang terakhir adalah parameter *Learning Rates* yang menetukan seberapa signifikan perubahan pada matriks bobot pada saat proses pembaruannya. Akan dicoba beberapa variasi *learning rate* yaitu 0.2, 0.02, dan 0.002.

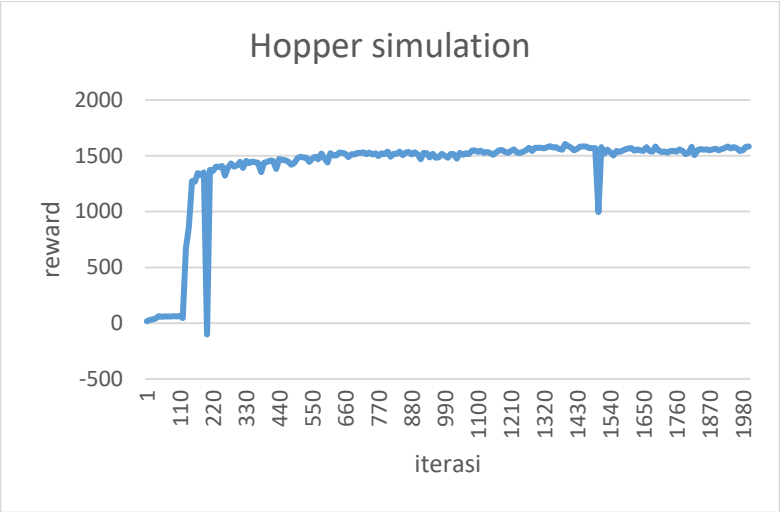


Gambar 5.4 Hasil percobaan variasi nilai parameter *learning rates*

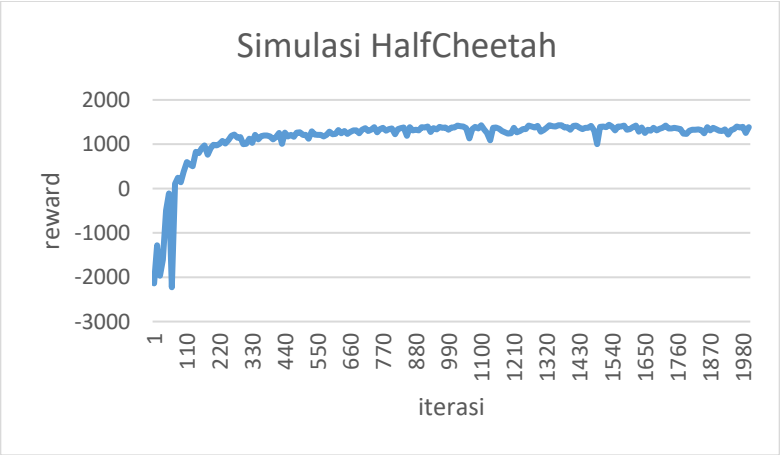
Dari hasil pengujian variasi nilai parameter *learning rates* dapat dilihat pada Gambar 5.4 bahwa pemilihan nilai parameter ini mempunyai pengaruh yang cukup besar pada hasil *training*. Pemilihan nilai yang terlalu kecil dapat membuat *training* membutuhkan iterasi yang lebih banyak untuk mencapai nilai tertentu. Begitu juga sebaliknya, dengan memilih nilai yang terlalu besar, proses pembaruan matriks akan menjadi terlalu bias pada setiap iterasinya dan membuat setiap iterasi terlalu memberikan pengaruh besar pada matriks bobotnya yang membuat sistem hanya terfokus pada *short-term goal*.

5.4.3 Uji Coba Performa ARS

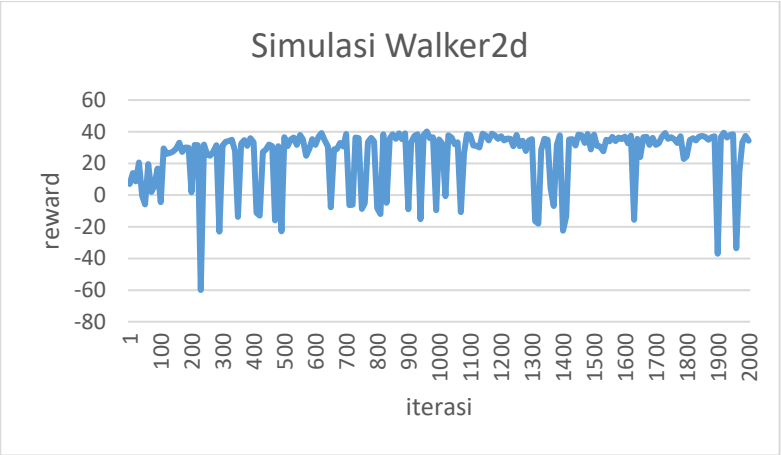
Uji coba performa ARS akan menggunakan hasil terbaik dari percobaan-percobaan sebelumnya dan akan dibandingkan dengan algoritma *Reinforcement Learning* lain yang terkenal yaitu A2C atau *Synchronus Advantage Actor Critic*. Pengujian akan dilakukan pada beberapa simulasi yaitu *Hopper*, *Halfcheetah* dan *Walker2D*.



Gambar 5.5 Pengujian simulasi hopper



Gambar 5.6 Pengujian simulasi HalfCheetah



Gambar 5.7 Pengujian simulasi Walker2d

Tabel 5.5 Hasil pengujian implementasi ARS

Simulasi	***target reward***	**Hasil pengujian rata-rata 10 iterasi terakhir**	**batas jumlah iterasi**
Hopper	1250	1570.63	2000
HalfCheetah	1700	1328.42	2000
Walker2D	1250	10.89	2000

Berdasarkan hasil pengujian yg dijelaskan pada Tabel 5.5 dan Gambar 5.5, 5.6 dan 5.7, target *reward* dari simulasi hopper berhasil tercapai. Namun pada simulasi Halfcheetah, rekaman video hasil *training* pada simulasi menujukan bahwa robot berhasil berjalan dan bergerak maju tapi belum mencapai kecepatan yang diinginkan yang membuat hasil *reward* belum mencapai target. Dalam Gambar 5.5 simulasi walker2d, terlihat bahwa dibutuhkan lebih banyak iterasi agar mencapai target yang diinginkan.

5.4.4 Perbandingan Performa ARS dengan Algoritma A2C

Pengujian terakhir akan dilakukan untuk melihat perbandingan performa ARS dengan algoritma RL lain yaitu A2C. Algoritma A2C menggunakan metode yang lebih kompleks *deep learning* dalam proses *training*nya. Pengujian akan dilakukan dengan memberikan batas jumlah *step* atau aksi yang dilakukan agen pada *environment* selama proses *training*.

Tabel 5.6 Pengujian Berdasarkan jumlah step pada *environment*

Simulasi	**Jumlah step pada simulasi**	**Reward ARS**	**Reward A2C**
Hopper	1,000,000	2093.51	860.35
HalfCheetah	1,000,000	226.36	953.21
Walker2D	1,000,000	153.53	266.22

Perbandingan ARS dengan algoritma RL lain ini dilakukan dengan melakukan perbandingan rata-rata reward yang bisa didapat dengan menggunakan model yang telah di*training.* Tabel 5.6 menunjukan bahwa simulasi Hopper algoritma ARS memberikan hasil jauh lebih baik dari algoritma A2C, sedangkan pada simulasi HalfCheetah dan Walker2D masih belum sebaik algoritma A2C.

5.5 *Hasil dan Evaluasi*

Pada uji coba versi implementasi ARS, dapat diketahui bahwa versi 3 yang merupakan modifikasi kecil dari implementasi versi 2 memberikan performa yang terbaik. Implementasi versi 3 memberikan hasil *training* yang stabil dan lebih cepat untuk mencapai hasil *reward* yang tinggi. Hal ini dikarenakan dengan menggunakan hanya beberapa *random perturbation* yang terbaik, proses pembaruan matriks bobot akan lebih efisien dan tidak akan terpengaruh oleh *random perturbation* yang buruk. Implementasi

versi 3 juga memiliki kelebihan yang sama seperti versi 2 jika dibandingkan dengan versi 1, yaitu penggunaan *normalizer* dan juga *scaling* menggunakan standar deviasi pada proses pembaruan matriks bobot. Penggunaan *normalizer* pada hasil observasi berguna untuk menghilangkan bias dan menyetarakan bobot pada setiap komponen dari hasil observasi. *Scaling* pembaruan matriks menggunakan standard deviasi memastikan tidak adanya perubahan matriks bobot yang ekstrim pada proses pembaruannya.

Hasil uji coba parameter *random directions* dan *best directions* memberikan gambaran bagaimana memilih kombinasi nilai yang tepat untuk kedua parameter tersebut. Apabila kita memiliki *resource* CPU yang banyak, kita dapat memberbesar nilai parameter *random directions* dan menerapkan *multiprocessing* pada program yang dibuat. Dengan nilai *best directions* yang paling tidak lebih kecil dari setengah nilai *random directions* kita akan mendapatkan hasil *training* yang optimal. Dengan pemilihan kombinasi nilai yang sesuai dengan *resource* yang ada, kita dapat mempercepat proses *training* untuk mencapai hasil yang diinginkan.

Pengujian Algoritma ARS pada berbagai simulasi yang berbeda menunjukan hasil yang baik pada hasil implementasi pada tugas akhir ini. Namun terbukti pada hasil percobaan simulasi HalfCheetah dan Walker2D yang belum optimal, menunjukan bahwa perlu adanya penyesuaian *Hyperparameter* yang berbeda untuk setiap *Environment* simulasi dan jumlah iterasi yang harus lebih besar untuk melihat hasil yang diinginkan.

Berdasarkan hasil perbandingan ARS dan algoritma A2C, ARS dapat mengalahkan performa algoritma A2C di simulasi Hopper. Hasil pengujian pada simulasi Hopper, algoritma ARS mencapai rata-rata *reward* sebesar 2093.51, sedangkan A2C sebesar 860.35. Hal ini menunjukan algoritma ARS mencapai *reward* 243% atau dua kali lebih besar dibandingkan algoritma A2C.

Dari hasil keseluruhan pengujian yang dilakukan, sistem dapat belajar untuk mencapai *goals* atau tujuan dari simulasi yang diujikan, namun sistem perlu diuji lebih lanjut pada simulasi-simulasi lain untuk membuktikan bagaimana ketahanan sistem pada perubahan simulasi tanpa perubahan yang signifikan pada algoritma ARS yang diimplementasikan. Penelitian lebih lanjut juga diperlukan untuk menguji hasil *training* ARS pada sistem pergerakan robot atau berbagai masalah RL di dunia nyata karena lingkungan simulasi jauh lebih sederhana dibandingkan dengan dunia nyata.

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas tentang kesimpulan yang didasari oleh hasil uji coba yang telah dilakukan pada bab sebelumnya. Kesimpulan nantinya sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut di masa depan.

6.1 *Kesimpulan*

Dalam pengerjaan Tugas Akhir ini setelah melalui tahap perancangan aplikasi, implementasi metode, serta uji coba, diperoleh kesimpulan sebagai berikut:

1. Dalam menyiapkan *environment* simulasi tempat belajarnya sistem RL perlu memilih pustaka yang tepat. Dengan pustaka yang tepat, pengukuran performa, pencatatan hasil, dan pemilihan simulasinya juga akan sangat dimudahkan. Pemilihan simulasi yang sesuai dengan kemampuan dan tujuan algoritma juga akan berpengaruh pada proses pengembangan dan implementasi algoritma RL.
2. Algoritma ARS dapat diimplementasikan pada sistem dengan bahasa pemrograman python. Hasil *training* dari sistem yang dibuat dapat menghasilkan sistem pergerakan robot multi-sendi yang efisien dengan mencapai batas *benchmark* dari simulasi yang dijalankan.
3. Performa algoritma ARS cukup bersaing dengan performa algoritma A2C yang apabila dilihat dari kompleksitas *sample* datanya jauh diatas algoritma ARS. Dengan pemilihan nilai *Hyperparameter* yang tepat, algoritma ARS mampu mengalahkan performa algoritma A2C dengan rata-rata reward pada hasil training 243% lebih besar dari algoritma A2C.

6.2 *Saran*

Saran yang diberikan untuk implementasi metode RL dengan algoritm *Augmented Random Search* pada simulasi pergerakan robot multi-sendi, yaitu:

1. Implementasi algoritma ARS pada jenis simulasi lain.
2. Implementasi algoritma ARS dalam permasalahan yang ada pada dunia nyata.
3. Melakukan eksplorasi parameter yang lebih optimal seperti *episode length*, *noise* dan penggunaan teknik normalisasi yang lain.

DAFTAR PUSTAKA

[1] O. Simeone, “A Very Brief Introduction to Machine Learning With Applications to Communication Systems,” *Information Theory,* p. 2, 2018.

[2] S. J. Russel dan P. Norvig, Artificial Intelligence : A Modern Approach, New Jersey: Alan Apt, 1995.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra dan M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *DeepMind Technologies,* p. 1, 2013.

[4] J. Jin, C. Song, H. Li, K. Gai, J. Wang dan W. Zhang, “Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising,” *CIKM '18 Proceedings of the 27th ACM International Conference on Information and Knowledge Management,* p. 1, 2018.

[5] J. Kober, J. A. Bagnell dan J. Peters, “Reinforcement Learning in Robotics: A Survey,” *The International Journal of Robotics Research,* p. 1, 2013.

[6] D. Silver, C. Madison, L. Sifre, A. Huang dan J. Schrittwieser, “Mastering the game of Go with deep neural networks and tree search,” *Nature,* no. 529, p. 1, 2016.

[7] Y. Toyoda dan F. Yano, “Optimizing Movement of A Multi-Joint Robot Arm with Existence of Obstacles Using Multi-Purpose Genetic Algorithm,” p. 1, 2004.

[8] S. Gopal Krishna Patro dan K. K. Sahu, “Normalization: A Preprocessing Stage,” *arXiv:1503.06462,* 2015.

[9] E. Kreyszig, Advanced Engineering Mathematics, Wiley, 1979, 1979.

[10] V. S. Y. Nesterov, “Random gradient-free minimization of convex functions,” *Foundations of Computational Mathematics,* vol. 17, no. 2, pp. 527-566, 2017.

[11] A. R. K. L. . E. T. dan . S. K. , “Towards Generalization and Simplicity in Continuous Control,” *arXiv:1703.02660,* 2017.

[12] “About Python,” Python, [Online]. Available: https://www.python.org/about/. [Diakses 30 November 2018].

[13] “Pybullet.org,” [Online]. Available: https://pybullet.org/. [Diakses 28 4 2020].

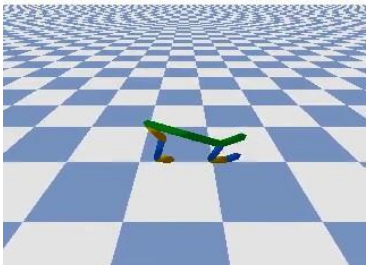
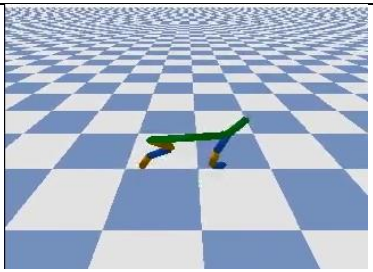
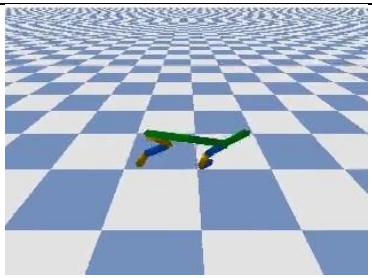
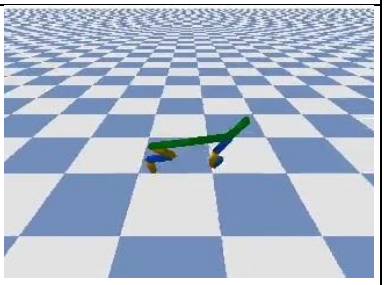
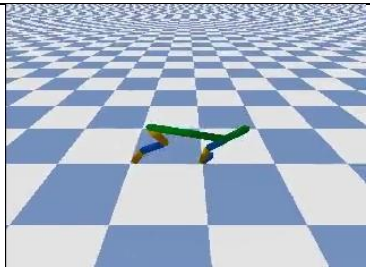
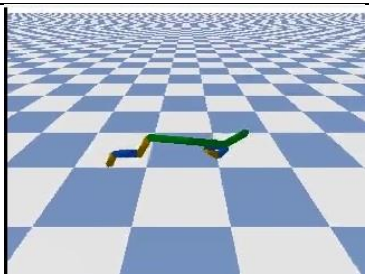
[14] “Ray Python,” [Online]. Available: https://github.com/ray-project/ray. [Diakses 23 5 2020].

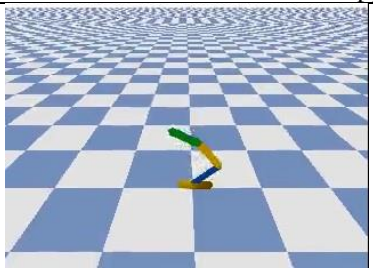
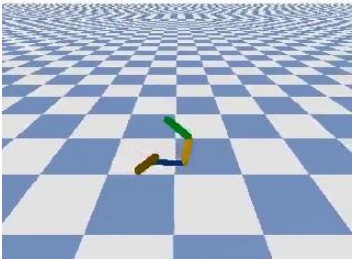
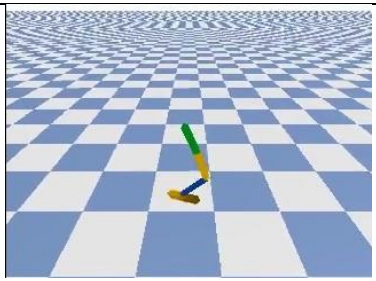
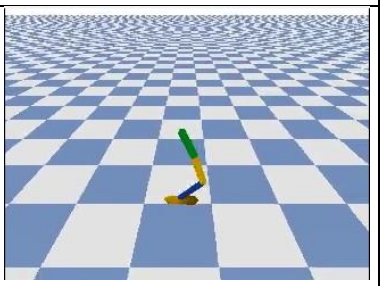
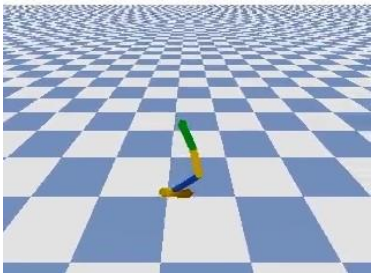
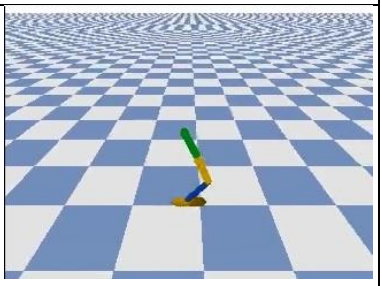
[15] “NumPy,” NumPy, [Online]. Available: http://www.numpy.org/. [Diakses 30 November 2018].

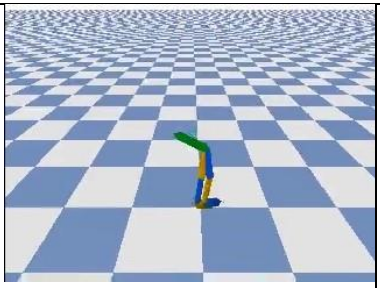
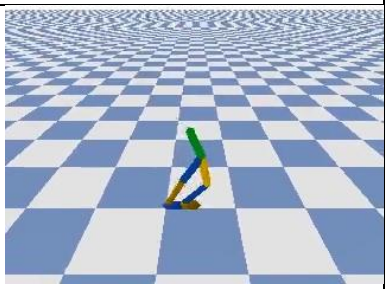
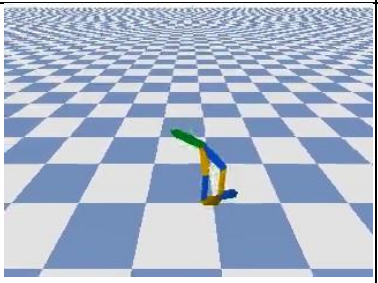
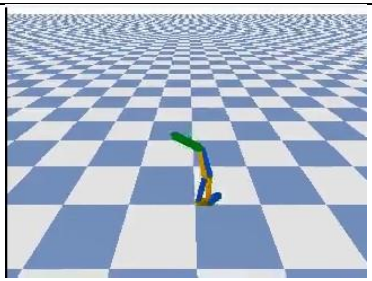
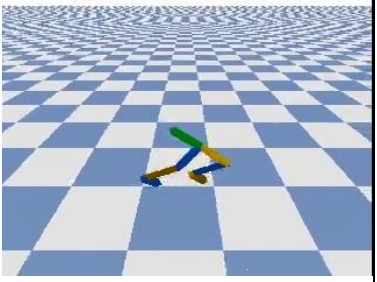
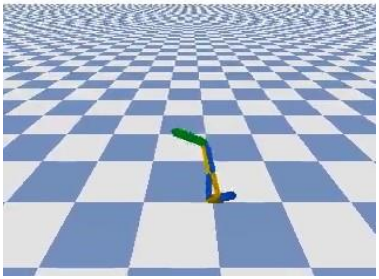
[16] “Gym Roboschool,” Gym OpenAI, [Online]. Available: https://openai.com/blog/roboschool/. [Diakses 3 6 2020].

[17] H. Mania, A. Guy dan B. Recht, “Simple Random Search Provides Competitive Approach to Reinforcement Learning,” *arXiv:1803.07055v1,* 2018.

LAMPIRAN

HalfCheetah	
 Iterasi 1	 Iterasi 50
 Iterasi 200	 Iterasi 800
 Iterasi 1500	 Iterasi 2000

Hopper2d	
 Iterasi 1	 Iterasi 50
 Iterasi 200	 Iterasi 800
 Iterasi 1500	 Iterasi 2000

Walker2D	
 Iterasi 1	 Iterasi 50
 Iterasi 200	 Iterasi 800
 Iterasi 1500	 Iterasi 2000

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Muhammad Adhitya Irvansyah, lahir di Bekasi pada tanggal 12 Februari 1998. Penulis menempuh pendidikan di SMAN 81 Jakarta (2012-2015) dan sekarang sedang menjalani pendidikan S1 Informatika di ITS. Penulis aktif dalam organisasi dan kepanitiaan Unit Kegiatan Mahasiswa ITS Billiard dan Schematics. Diantaranya adalah menjadi staff Departemen *Technical Development* ITS Billiard 2016-2017, staff ahli Departemen *Technical Development* ITS Billiard 2017-2018, staff Departemen REEVA Schematics ITS 2016 dan staff ahli Departemen REEVA Schematics ITS 2017. Komunikasi dengan penulis dapat melalui telepon: +6282330093097 dan *email*: **adhityairvan@yahoo.com**.