PUI-AIHeS
(15 September 2023 S/D 12 Desember 2023)

KERJA PRAKTEK – EE184601

Simulator Robot Sepak Bola Beroda Berbasis Robot Operating System

Danendra Clevero Ananda NRP 5022201123

DOSEN PEMBIMBING
Ir. Tasripan,. M. T
NIP. 196204181990031004

JURUSAN TEKNIK ELEKTRO
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2023

PUI-AIHeS
(15 September 2023 S/D 12 Desember 2023)

KERJA PRAKTEK – EE184601

Simulator Robot Sepak Bola Beroda Berbasis Robot Operating System

DANENDRA CLEVERO ANANDA NRP 5022201123

DOSEN PEMBIMBING
Ir. Tasripan,. M. T
NIP. 196204181990031004

JURUSAN TEKNIK ELEKTRO
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

LEMBAR PENGESAHAN

SIMULATOR ROBOT SEPAK BOLA BERODA BERBASIS ROBOT OPERATING SYSTEM
di
PUI-AIHeS

Laporan Kerja Praktek ini Disusun untuk Memenuhi
Persyaratan Akademik di Departemen Teknik Elektro
FTEIC – ITS
Tempat Pengesahan di : Surabaya
Tanggal : 6 Desember 2023

Menyetujui,
Dosen Pembimbing,

Ir. Tasripan, M. T
NIP. 196204181990031004

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

Mengetahui,
Kepala

Dedet Candra Riawan, ST., M.Eng., Ph.D.
NIP. 19731119 200003 1 001

LEMBAR PENGESAHAN

SIMULATOR ROBOT SEPAK BOLA BERODA BERBASIS ROBOT OPERATING SYSTEM
di
PUI-AIHeS

Laporan Kerja Praktek ini Disusun untuk Memenuhi
Persyaratan Akademik di Departemen Teknik Elektro
FTEIC – ITS
Tempat Pengesahan di : Surabaya
Tanggal : 6 Desember 2023

Menyetujui,
Pembimbing Kerja Praktik,

Dr. Rudy Dikairono, S.T., M.T.
NIP. 198103252005011002

Mengetahui,
Manager Klaster ICTR

Dr. Ir. Endroyono, DEA
NIP. 196504041991021001

ABSTRAK

Saat ini pengembangan robot sangatlah cepat. Industri 4.0 membuat semua perkembangan jauh lebih cepat dari sebelumnya. Akan tetapi manufaktur perangkat keras dapat dibilang cukup lama. Maka dari itu diperlukannya sebuah sistem yang dapat melakukan fungsional robot seperti pada dunia nyata, yang tidak memerlukan waktu untuk mengembangkan. Gazebo membuat hal seperti ini sangat memungkinkan dengan mempertimbangkan faktor faktor pada dunia nyata, seperti gravitasi, gesekan, dll. Dengan perkembangan seperti ini, kita dapat mempercepat proses perkembangan di bidang industri lebih khususnya robotika.

Kata kunci : Robot, *Simulator*, *Multiple Machine*

ABSTRACT

Currently, robot development is advancing rapidly. Industry 4.0 has significantly accelerated progress compared to the past. However, manufacturing hardware can be quite time-consuming. Therefore, a system that can perform real-world robot functions virtually, without requiring development time, is necessary. Gazebo facilitates this by considering real-world factors such as gravity, friction, etc. With such developments, we can expedite progress in the industrial sector, especially in robotics.

Keywords : **Robot, *Simulator*, *Multiple Machine***

KATA PENGANTAR

Dalam kesempatan ini, penulis ingin mengucapkan rasa syukur dan terima kasih atas segala limpahan rahmat dan karunia yang telah diberikan oleh Allah SWT. Shalawat serta salam tak lupa saya haturkan kepada Nabi Muhammad SAW, yang telah menjadi rahmat bagi seluruh alam.

Laporan kerja praktik ini berjudul " Simulator Robot Sepak Bola Beroda Berbasis Robot Operating System" Fokus utama dalam kerja praktik ini adalah bagaimana membentuk sebuah simulasi yang dapat diintegrasikan dengan program robot.

Terlepas dari semua itu, penulis sadar bahwa terdapat kekurangan pada tugas akhir ini. Oleh karena itu, penulis dengan terbuka menerima kritik dan saran dari pembaca agar dapat memperbaiki tugas akhir ini. Semoga penelitian ini dapat memberikan manfaat untuk banyak orang dan dapat dikembangkan lebih baik lagi pada penelitian selanjutnya.

Surabaya, Juni 2023

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SIMBOL

BAB 1 PENDAHULUAN

Pada bab ini akan dijelaskan mengenai latar belakang, tujuan, manfaat dari kerja praktek, batasan dan asumsi yang digunakan serta sistematika penulisan yang digunakan untuk menyelesaikan laporan ini.

1.1 Latar Belakang

Seperti yang telah kita ketahui bahwa perkembangan teknologi dan ilmu pengetahuan yang terus meluas telah memberikan kemajuan di berbagai bidang, khususnya pada bidang industri. Tantangan dan permasalahan baru juga dapat muncul seiring dengan berkembangnya hal tersebut. Oleh karena itu, perguruan tinggi sebagai institusi pendidikan diharapkan mampu mencetak sumber daya manusia yang memiliki kecerdasan intelektual yang baik, tanggap terhadap kebutuhan dan pengembangan ilmu pengetahuan teknologi (IPTEK), serta memiliki kualitas dan kompetensi terutama pada bidangnya masing-masing.

Kami merupakan mahasiswa dari Departemen Teknik Elektro, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember Surabaya adalah salah satu perguruan tinggi negeri yang berorientasi pada ilmu pengetahuan dan teknologi memberikan kesempatan pada mahasiswa untuk mendapatkan pengalaman nyata secara langsung agar dapat mengembangkan diri dan mampu memahami dan menguasai perkembangan yang ada terutama di bidang industri. Salah satunya adalah dengan

mewajibkan seluruh mahasiswa untuk mengambil program kerja praktik dalam kurikulumnya.

Kerja praktik (KP) merupakan salah satu mata kuliah di Departemen Teknik Elektro, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember Surabaya, sebagai pelengkap teori yang dipelajari pada masa perkuliahan serta sebagai sarana pengembangan dan penerapan pengetahuan yang dimiliki mahasiswa. Sehingga mahasiswa dapat menjadi sumber daya manusia yang mampu menghadapi tantangan dan permasalahan yang muncul di era globalisasi.

1.2 Waktu dan Tempat Pelaksanaan

Kerja Praktik ini dilakukan pada 15 September 2023 s.d 12 Desember 2023 di Lab B 202 Departemen Teknik Elektro dan Workshop Robotika Institut Teknologi Sepuluh Nopember.

1.3 Permasalahan dan Batasan

Pada Kegiatan Kerja Praktik ini dilakukan pembentukan simulasi robot sepak bola beroda dengan menggunakan gazebo ros sebagai framework untuk pembentukan sistim. Hal ini dikarenakan keterbatasan tim IRIS dalam hal pengembangan *software* pada saat *hardware* sedang pada tahap riset.

1.4 Tujuan

1) Menciptakan suatu *platform* yang dapat menjadi dasar dari simulasi robot.
2) Mampu menciptakan simulasi tiga dimensi untuk robot sepak bola beroda.
3) Mampu mengintegrasikan program tim IRIS dengan simulator.

1.5 Sistematika Penulisan

1) BAB 1 PENDAHULUAN
   Bab ini berisikan latar belakang, waktu dan tempat pelaksanaan, tujuan, batasan masalah, metodologi pengumpulan data, dan sistematika penulisan laporan.
2) BAB 2 SEJARAH DAN PROFIL PERUSAHAAN
   Bab ini berisikan mengenai sejarah, profil perusahaan, struktur perusahaan, dan kegiatan mengenai perusahaan yang dijadikan tempat kerja praktek.
3) BAB 3 TINJAUAN PUSTAKA
   Bab ini berisi tentang metode CNN atau convolutional neural network yang digunakan untuk mengolah citra dari dataset yang telah dikumpulkan berupa potongan huruf-huruf braille menggunakan library Tensorflow. Selain itu juga dibahas tentang penggunaan software Android Studio menggunakan bahasa pemrograman Java yang juga terintegrasi dengan Tensorflow.
4) BAB 4 PEMBAHASAN
   Bab ini berisi tentang pembahasan utama untuk kegiatan Kerja Praktik yang berisi tentang perancangan sistem menggunakan Tensorflow untuk melatih model dari dataset berisi gambar dari potongan huruf yang telah di augmentasi. Kemudian model yang didapatkan di-deploy pada perangkat Android menggunakan software Android Studio.
5) BAB 5 KESIMPULAN DAN SARAN
   Pada bab ini berisi kesimpulan dan saran dari keseluruhan Kerja Praktik.

1.6 Manfaat

Manfaat pelaksanaan kerja praktik di Laboratorium AJ 304 Departemen Teknik Elektro, adalah sebagai berikut:

1) Bagi Perguruan Tinggi
   Sebagai tambahan referensi khususnya mengenai perkembangan teknologi di bidang machine learning terutama pada penguasaan deep learning di Indonesia dan dapat digunakan oleh civitas akademika perguruan tinggi.
2) Bagi Mahasiswa
   Mahasiswa dapat mengetahui secara lebih mendalam tentang aplikasi machine learning pada bidang elektronika sehingga diharapkan mampu menerapkan ilmu yang telah diperoleh di perkuliahan.

BAB 2 SEJARAH DAN PROFIL PERUSAHAAN

2.1 Sejarah PUI-AIHeS

PUI-AIHeS merupakan salah satu Lembaga riset yang terletak di Gedung Robotika, Institut Teknologi Sepuluh Nopember. PUI-AIHeS memberi sarana kepada para mahasiswa dalam mempelajari, mengembangkan, dan meneliti robot. PUI-AIHeS sendiri memiliki beberapa tim robot yang mana salah satunya merupakan Tim IRIS yang bergerak di bidang robot sepak bola beroda.

2.2 Logo Perusahaan

Gambar 2. 1 Logo PUI-AIHES

2.3 Visi dan Misi

2.3.1 Visi

Visi dari PUI-AIHeS adalah Menjadi pusat rujukan pengembangan teknologi AI di sektor kesehatan untuk memberi manfaat kepada masyarakat dalam skala nasional dan internasional.

2.3.2 Misi

Misi dari PUI-AIHeS adalah Menghasilkan produk perangkat keras atau perangkat lunak yang dikembangkan berdasarkan kebutuhan atau permintaan pengguna sambil

berkolaborasi dengan masyarakat dan pemerintah, bersama dengan industri bisnis kecil dan menengah.

2.4 Produk dan Jasa

2.5 Kegiatan PUI-AIHeS

Kegiatan dalam PUI-AIHeS meliputi :

1) Penelitian.
   Penelitian yang dilakukan di PUI-AIHeS antara lain :
   - Robot RAISA
   - Robot ICAR
   - Robot-robot lain seperti robot sepak bola beroda, robot sepak bola humanoid, robot bawah air, robot tari, dan lain lain.Wilayah Kerja

   PUI-AIHeS terletak di Workshop Robotika ITS. Adapun alamat dari PUI-AIHeS adalah Jl. Teknik Sipil No.2, Keputih, Kec. Sukolilo, Surabaya, Jawa Timur 60111.

2.6 Jam Kerja Perusahaan

Aktivitas seluruh sivitas akademika dari PUI-AIHeS dilakukan pada hari Senin hingga Jum'at, mulai pukul 07.00 WIB - 16.00 WIB.

BAB 3 TINJAUAN PUSTAKA

3.1 Robot Operating System

Robot Operating System (ROS) adalah framework terbuka yang digunakan untuk pengembangan perangkat lunak robot. ROS menyediakan layanan yang diharapkan dari sistem operasi, seperti abstraksi perangkat keras, kontrol perangkat lunak rendah, implementasi fungsionalitas umum, dan penanganan pesan antar proses. ROS juga mendukung penggunaan kode secara berulang-ulang dan berbagi dalam komunitas robotik. Ini memudahkan integrasi antarmuka sensor dan aktuator, serta pengembangan algoritma robotik yang kompleks. (J., Wheeler, R., & Ng, A. Y, 2009)

3.2 Gazebo

Gazebo adalah simulator robotik 3D yang digunakan dalam pengujian dan pengembangan algoritma robot, desain robot, dan simulasi interaksi robot dengan lingkungan yang kompleks. Gazebo menawarkan kemampuan fisika yang realistis, berbagai sensor dan objek lingkungan, serta rendering grafis berkualitas tinggi. Gazebo juga terintegrasi dengan baik dengan ROS, yang memungkinkan simulasi yang mendekati kondisi dunia nyata. (Koenig, N., & Howard, A, 2004)

3.3 CPP

C++ adalah bahasa pemrograman yang sering digunakan dalam pengembangan perangkat lunak robotik. Dikenal karena efisiensinya, C++ memungkinkan kontrol tingkat rendah atas sumber daya sistem sambil tetap menawarkan tingkat abstraksi tinggi. Ini adalah pilihan yang umum untuk implementasi algoritma yang memerlukan

pengolahan data yang cepat dan efisien. C++ juga sering digunakan dalam integrasi dengan ROS dan Gazebo untuk pengembangan aplikasi robotik. (Stroustrup, B. 2013)

BAB 4 PEMBAHASAN

4.1 Perangkat Yang Digunakan

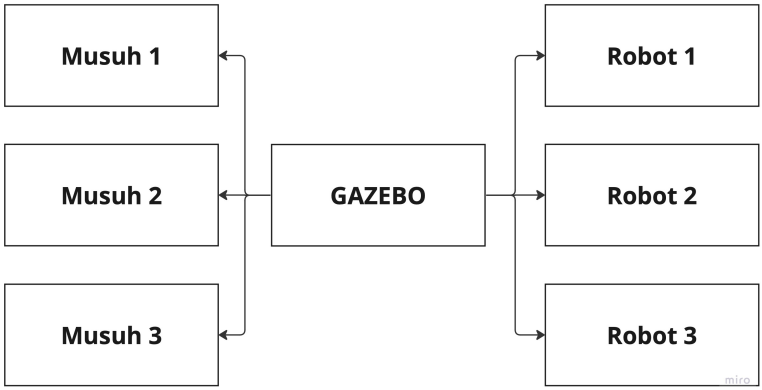
Adapun perangkat yang digunakan dalam pengembangan kerja praktik ini adalah:

1) Ubuntu.
2) Visual Studio Code.
3) Gazebo.

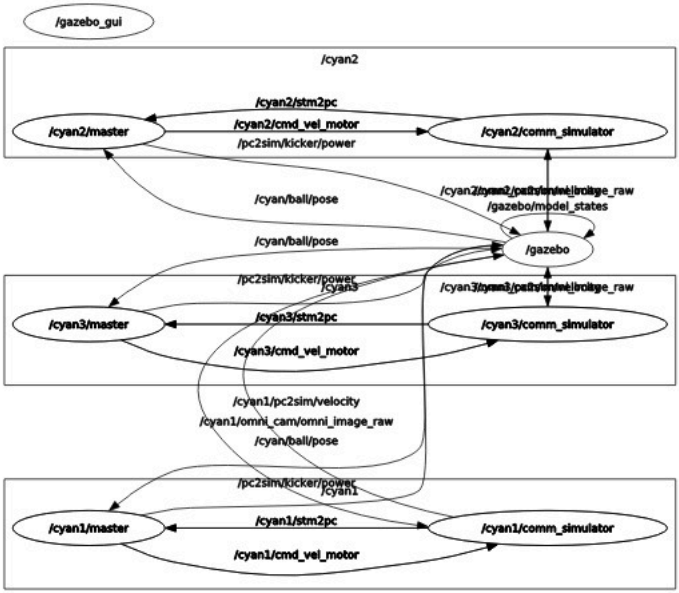
4.2 Perancangan Sistem

Pelaksanaan kerja praktek mengikuti langkah – langkah diantaranya pembuatan desain robot, percobaan pembuatan gazebo, dan integrasi gazebo dengan program.

4.2.1 Diagram Blok Sistem



Gambar 4. 1 Diagram Blok Sistem



Gambar 4. 2 Diagram Alir Data

4.2.2 Sistem Kerja Gazebo

Sistem gazebo sesuai dengan aliran data, dimana gazebo adalah satu aplikasi *standalone* yang nantinya akan berdiri sendiri akan tetapi berada pada satu *workspace* yang sama dengan program ROS robot, dan program ROS robot akan dijalankan pada terminal lainnya.

4.2.3 Realisasi Sistem

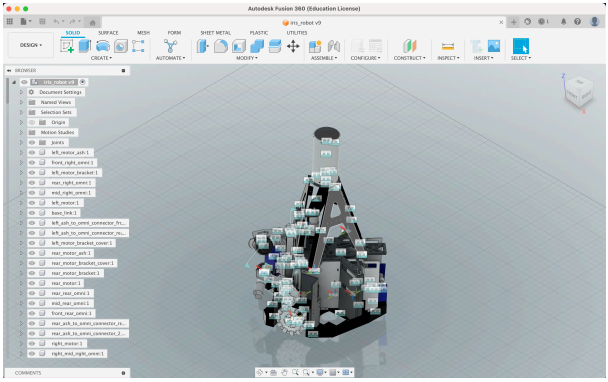
Pada Realisasi Sistem pada perancangan sistem ini, terdiri dari perancangan perancangan desain robot. Untuk perancangan robot penulis akan menggunakan aplikasi Fusion 360 untuk membuat bentuk dari robot tersebut. Setelahnya untuk perancangan lapangan dan asset lainnya,

penulis akan menggunakan asset yang telah disediakan oleh tim Nu-Bot dengan sedikit konfigurasi tambahan.

4.2.4 Rancangan desain robot

1) Perancangan model dengan menggunakan Fusion 360.

   Hasil dari perancangan model ini dapat dilihat pada gambar berikut

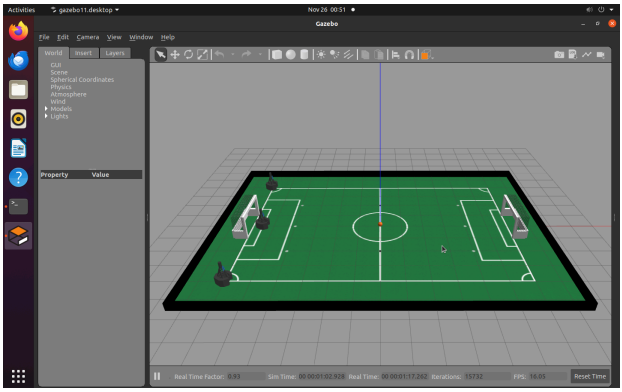


Gambar 4. 3 Gambar Robot pada Fusion 360

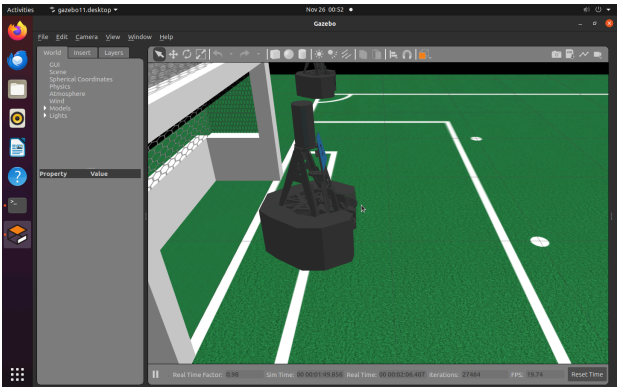
Akan tetapi desain ini nantinya tidak akan digunakan pada kerja praktik ini, melainkan desain ini akan digunakan pada pengembangan lebih lanjut.

2) Visualisasi semua asset pada gazebo.

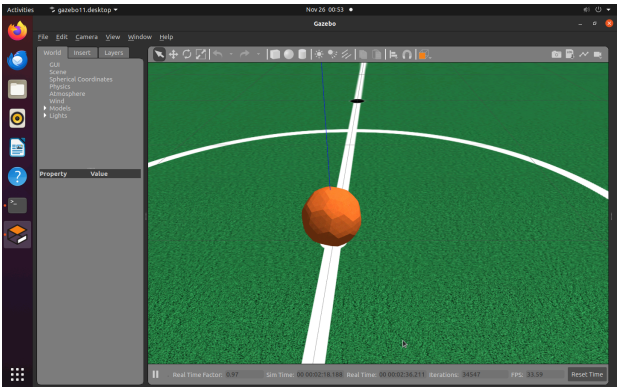
   Setelah membentuk desain yang sesuai penulis akan menggunakan gazebo untuk melakukan visualisasi hasilnya.



Gambar 4. 4 Visualisasi Robot pada Gazebo



Gambar 4. 5 Visualisasi Robot pada Gazebo (zoom)

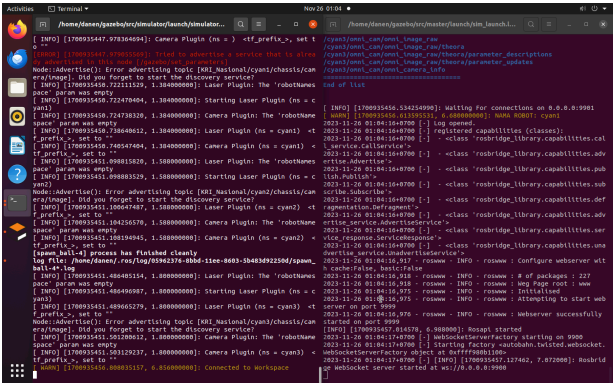


Gambar 4. 6 Visualisasi Bola pada Gazebo

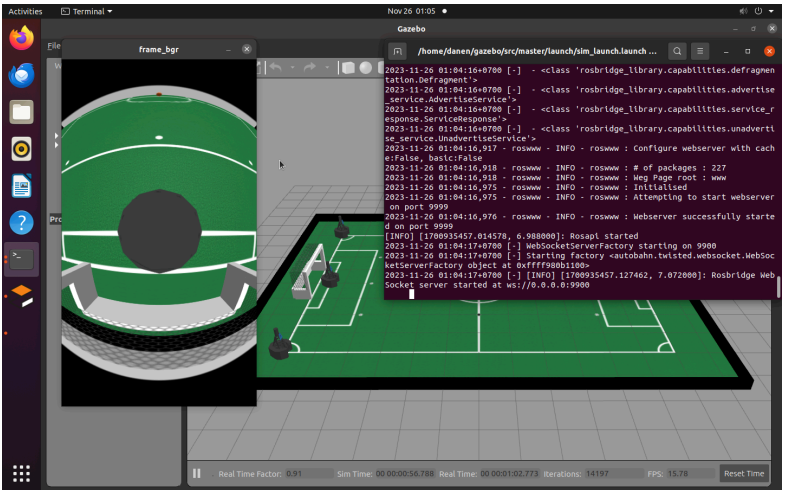
Asset yang digunakan telah tervisualisasi dengan baik kedalam gazebo serta dapat digunakan. Selanjutnya penulis akan melakukan tahap pengujian sistem.

4.3 Pengujian Sistem

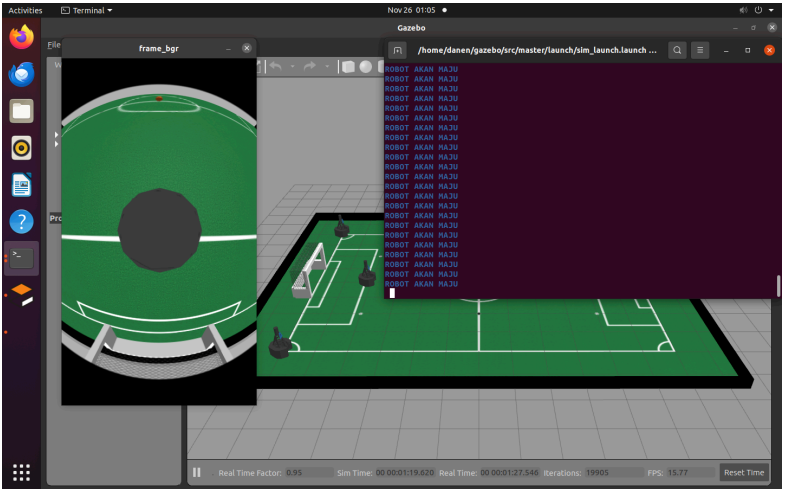
4.3.1 Pengujian koneksi gazebo dengan satu *workspace* robot



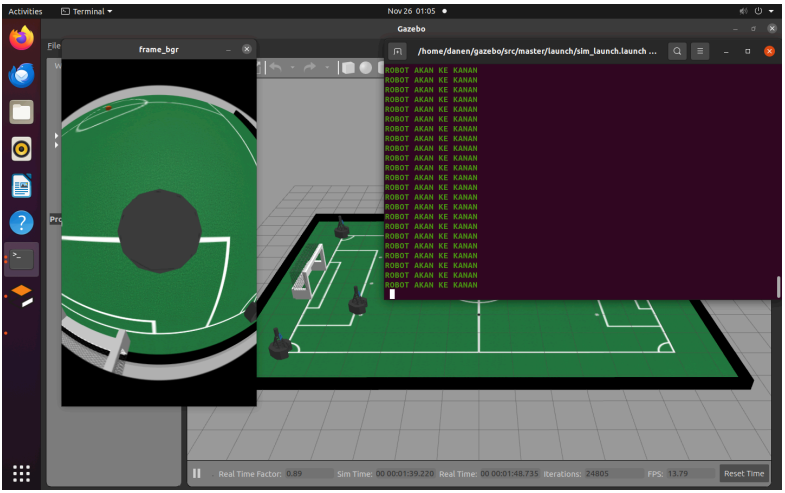
Gambar 4.1 Status Koneksi Robot ke Gazebo



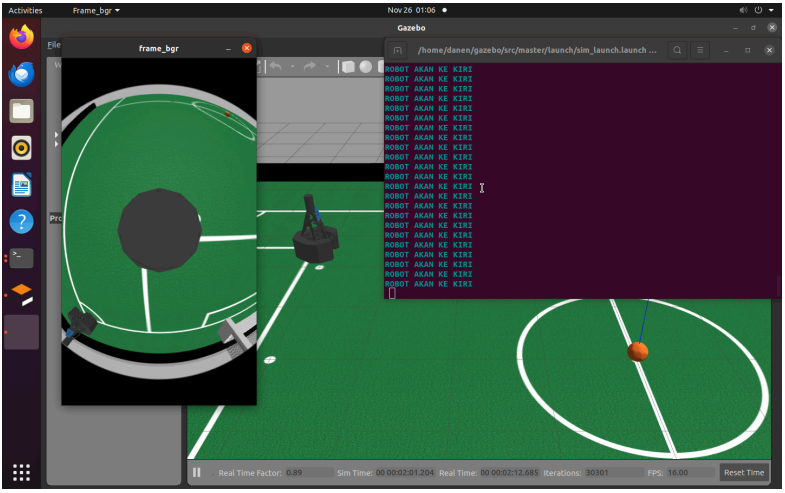
Gambar 4.2 Visualisasi Kamera Robot yang Terkoneksi



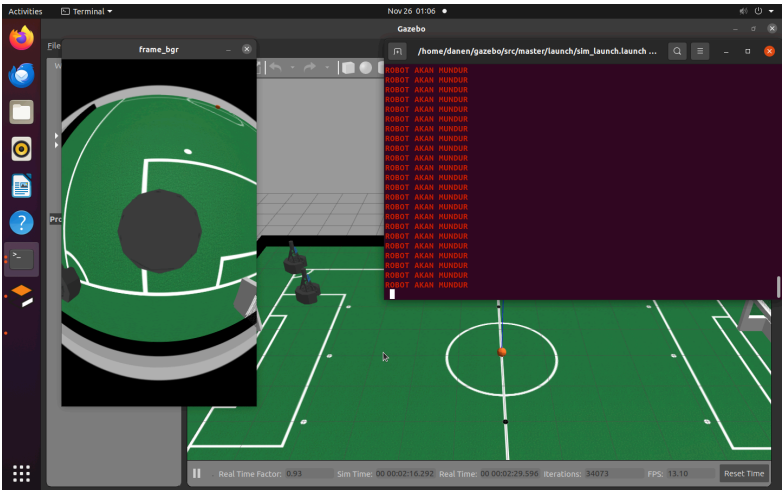
Gambar 4.3 Visualisasi Pergerakan Robot (maju)



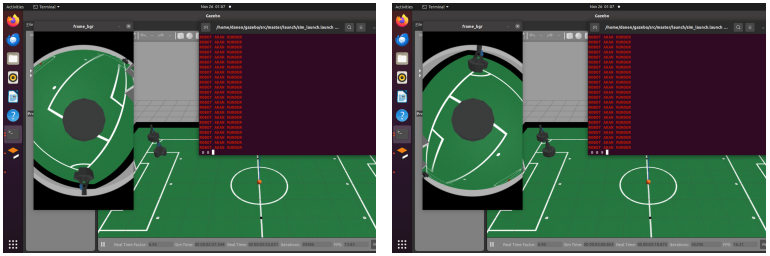
Gambar 4. 7 Visualisasi Pergerakan Robot (kanan)



Gambar 4. 8 Visualisasi Pergerakan Robot (kiri)



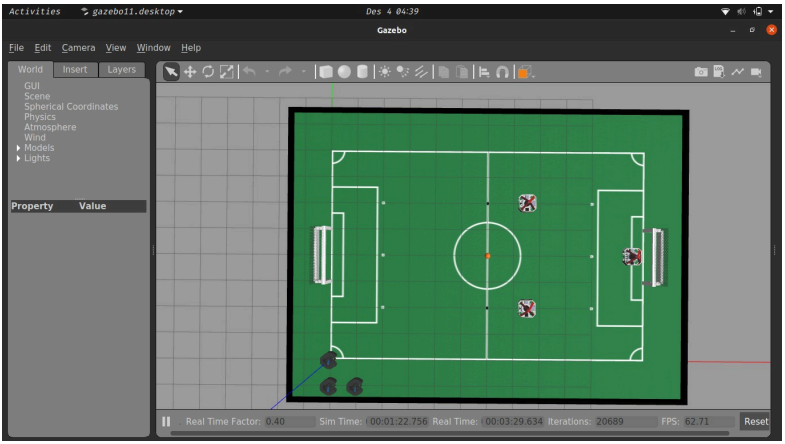
Gambar 4. 9 Visualisasi Pergerakan Robot (mundur)



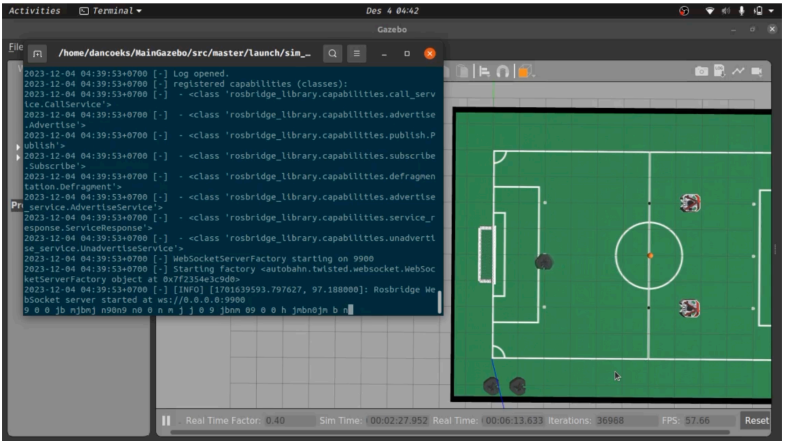
Gambar 4. 10 Visualisasi Pergerakan Robot (rotasi kanan kiri)

Program dari robot dapat terkoneksi dengan program dari gazebo. Program robot dapat mendapatkan data data seperti data posisi robot, kamera robot. Serta gazebo dapat menerima data kecepatan yang diinginkan untuk suatu robot yang spesifik.

4.3.2 Pengujian satu robot terkoneksi



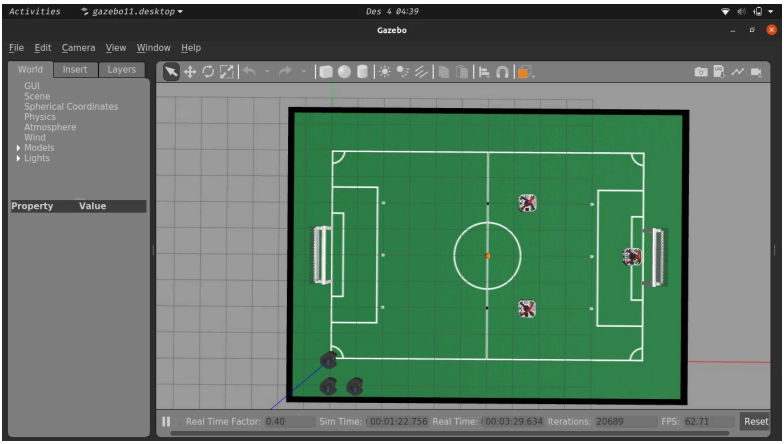
Gambar 4. 11 Posisi Robot Mula Mula



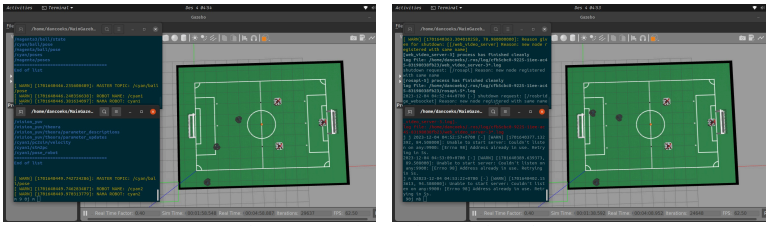
Gambar 4. 12 Koneksi Satu Robot

Robot dapat melakukan translasi sesuai dengan perintah yang diberikan.

4.3.3 Pengujian dua robot terkoneksi



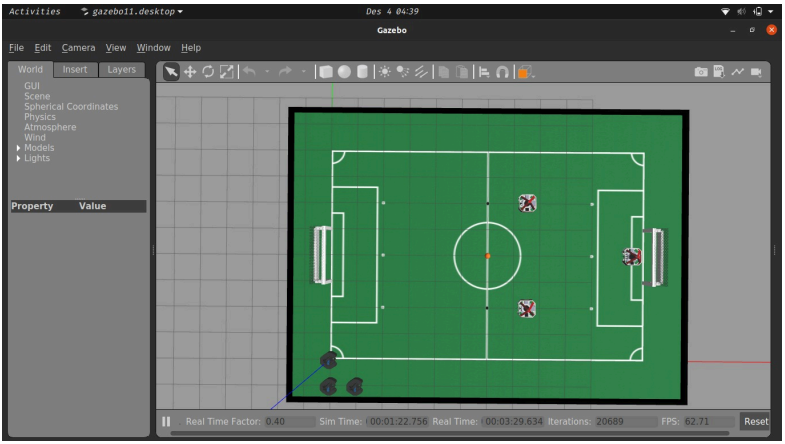
Gambar 4. 13 Posisi Robot Mula Mula



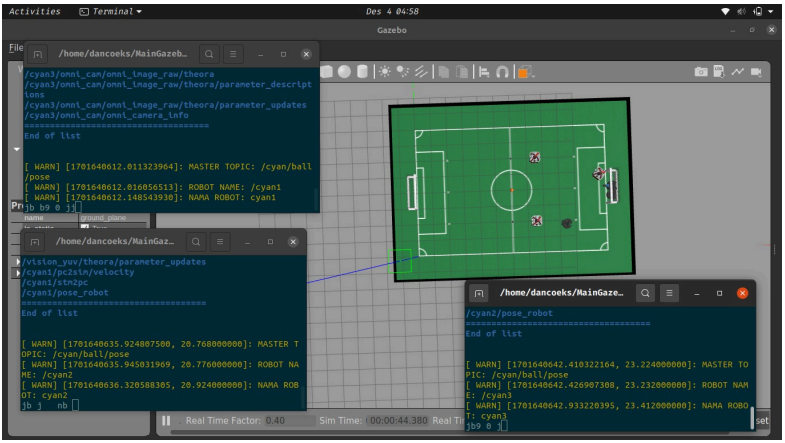
Gambar 4. 14 Dua Robot Terkoneksi

Robot dapat melakukan translasi sesuai dengan perintah yang diberikan untuk masing masing robot yang spesifik (terdapat dua terminal, yang masing masing menjalankan program untuk tiap robot).

4.3.4 Pengujian tiga robot terkoneksi



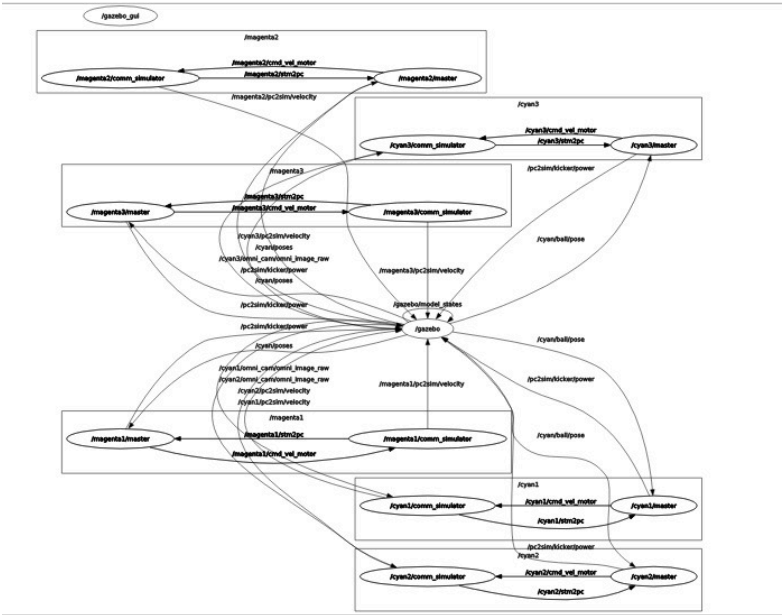
Gambar 4. 15 Posisi Robot Mula Mula



Gambar 4. 16 Tiga Robot Terkoneksi

Robot dapat melakukan translasi sesuai dengan perintah yang diberikan untuk masing masing robot yang spesifik (terdapat tiga terminal, yang masing masing menjalankan program untuk tiap robot).

4.3.5 Semua Robot Dijalankan



Gambar 4. 17 Alir Data Saat Semua Robot Dijalankan

Kondisi alir data saat keenam robot dijalankan.

BAB 5 PENUTUP

5.1 Kesimpulan

Berdasarkan hasil penelitian Simulator Robot Sepak Bola Beroda ini didapatkan kesimpulan seperti berikut.

1) Sistem simulator dapat berjalan sesuai yang diinginkan.
2) Sistem simulator dapat terintegrasi dengan program ros robot.
3) Sistem simulator dapat terkoneksi dengan banyak robot.

5.2 Saran

Berdasarkan hasil penelitian Simulator Robot Sepak Bola Beroda, saran yang dapat diberikan penulis adalah sebagai berikut.

1) Model robot terbaru yang telah penulis buat dapat digunakan pada penelitian selanjutnya.

DAFTAR PUSTAKA

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software*, 3(3.2), 5.

Koenig, N., & Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Vol. 3, pp. 2149-2154). IEEE.

Stroustrup, B. (2013). The C++ programming language (4th ed.). Addison-Wesley

LAMPIRAN

Lampiran A : Program Pengolahan Robot di Gazebo (header)

```
#ifndef ROBOT_GAZEBO_NEW_CONTROL_HH
#define ROBOT_GAZEBO_NEW_CONTROL_HH

#include <functional>
#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/common/common.hh>

// Gazebo versi 9
#include <ignition/math/Vector3.hh>
#include <ignition/math/Pose3.hh>
#include <ignition/math/Quaternion.hh>

#include <angles/angles.h>
#include <ros/ros.h>
#include <ros/callback_queue.h>
#include <ros/subscribe_options.h>
#include <gazebo_msgs/ModelStates.h>
#include <geometry_msgs/Pose2D.h>
#include <geometry_msgs/Twist.h>
#include <std_msgs/UInt8.h>
#include <std_msgs/Float32.h>
#include <std_msgs/Float32MultiArray.h>
#include <std_msgs/Int16MultiArray.h>
#include <sensor_msgs/LaserScan.h>
#include <boost/thread/mutex.hpp>
#include <std_msgs/Char.h>
#include <iostream>
#include "simulator/ball_gazebo.hh"
```

```
#include "iris_msgs/robot2sim.h"
#include "iris_msgs/robotdata.h"

#include <logger/logger.hpp>

#define DEG2RAD 0.017452925
#define RAD2DEG 57.295780
#define DIV180 0.005555556

#define TOTAL_CYAN 3
#define TOTAL_MAGENTA 3

logger::Logger logger_sim;

using namespace gazebo;
using namespace ignition;
using namespace std;

//-----Model Name
#define MODEL_BALL "bola"
#define MODEL_CYAN1 "cyan1"
#define MODEL_CYAN2 "cyan2"
#define MODEL_CYAN3 "cyan3"
#define MODEL_CYAN4 "cyan4"
#define MODEL_CYAN5 "cyan5"
#define MODEL_MAGENTA1 "magenta1"
#define MODEL_MAGENTA2 "magenta2"
#define MODEL_MAGENTA3 "magenta3"
#define MODEL_MAGENTA4 "magenta4"
#define MODEL_MAGENTA5 "magenta5"

//-----Team Identifier
#define CYAN_TEAM 'c'
```

```
#define MAGENTA_TEAM 'm'

#define BALL_DIST 35

math::Pose3d cyan_pose[6];
math::Pose3d magenta_pose[6];

typedef struct
{
    float x;
    float y;
    float th;
} Vect3;

namespace gazebo
{
    class RobotGazeboSecond : public ModelPlugin
    {
    public:
        RobotGazeboSecond();
        ~RobotGazeboSecond();
        void Load(physics::ModelPtr _model,
sdf::ElementPtr);
        void OnUpdate();

        //---Ros Subscribers
        void CllbckSubGazeboModel(const
gazebo_msgs::ModelStates::ConstPtr &msg);
        void CllbckSubVelocity(const
iris_msgs::robot2sim::ConstPtr &msg);
        void CllbckSubKickerPower(const
std_msgs::Float32ConstPtr &msg);
```

```
        //---Utils
        /**
         * @brief The RobotGazebo::QueueThread()
function is intended to run in a separate
         * thread and manages the processing of
incoming ROS messages.
         */
        void QueueThread();

    private:
        //---Gazebo variables
        physics::WorldPtr world;
        physics::ModelPtr robot_model;
        std::string robot_model_name;
        event::ConnectionPtr update_connection;
        physics::ModelPtr ball_model;
        unsigned int ball_index;

        //---ROS NodeHandle
        std::unique_ptr<ros::NodeHandle> ros_node;

        //---ROS Subscriber
        ros::Subscriber sub_gazebo_model;
        ros::Subscriber sub_kicker_power;
        ros::Subscriber sub_vel_cyan[4];
        ros::Subscriber sub_vel_magenta[4];

        //---ROS Publisher
        ros::Publisher pub_cyan_pose[4];
        ros::Publisher pub_cyan_ball_state[4];
        ros::Publisher pub_magenta_pose[4];
        ros::Publisher pub_magenta_ball_state[4];
        ros::Publisher pub_ball_pose_cyan;
```

```
        ros::Publisher pub_ball_pose_magenta;
        ros::Publisher pub_friend_poses;
        ros::Publisher pub_enemy_poses;
        ros::Publisher pub_total_active_cyan;
        ros::Publisher pub_total_active_magenta;

        //---Query
        ros::CallbackQueue ros_queue;
        std::thread ros_queue_thread;

        /**
         * @brief First array is the robot number
         * @brief Second array is the motor number
         * @param second_array is [left, right,
rear]
         */
        Vect3 vel_cyan_motor[TOTAL_CYAN + 1];
        Vect3 final_vel_cyan[TOTAL_CYAN + 1];
        Vect3 cyan_pose[TOTAL_CYAN + 1];

        /**
         * @brief First array is the robot number
         * @brief Second array is the motor number
         * @param second_array is [left, right,
rear]
         */
        Vect3 vel_magenta_motor[TOTAL_MAGENTA + 1];
        Vect3 final_vel_magenta[TOTAL_MAGENTA + 1];
        Vect3 magenta_pose[TOTAL_MAGENTA + 1];

        //---Status
        /**
```

```
         * @brief To decide the robot in that index
is connected with robot code or not
         * if not, it would not give velocity to the
robot
         * @param first_array is the robot number
         */
        uint8_t cyan_connection_stat[TOTAL_CYAN +
1];
        float cyan_connection_timer[TOTAL_CYAN + 1];
        /**
         * @brief To decide the robot in that index
is connected with robot code or not
         * if not, it would not give velocity to the
robot
         * @param first_array is the robot number
         */
        uint8_t
magenta_connection_stat[TOTAL_MAGENTA + 1];
        float magenta_connection_timer[TOTAL_MAGENTA
+ 1];

        /**
         * Decide how much robot is active to tell
workspace to decide between double and single
         */
        uint8_t active_cyan;
        uint8_t active_magenta;

        Vect3 ball_pose;
        Vect3 ball_pose_on_real;
        uint8_t ball_status;
        float kicker_power;
```

```
        //---Functions
        void RobotControl();
        void ComputeRobotPosition();
        /**
         * @brief Remember that X in our real world
format is diffrent
         * from X in gazebo
         *
         * In gazebo
         *
         * y
         *  _
         * / \
         *  |
         *  |
         *  |
         *  |__________ x
         *
         * In robot
         *
         * x
         *  _
         * / \
         *  |
         *  |
         *  |
         *  |__________ y
         *
         */
        void ComputeRobotVelocity();
        void ComputeBallPosition();
        void ComputeBallVelocity();
        void PublishData();
```

```
        void MoveRobot(uint8_t _coderobot, float
_vx, float _vy, float _vsudut);
        bool CheckBallStatus(Vect3 robot_pose);
        void BallDribbling(Vect3 robot_pose, string
robot_model_name);
        void Kick(uint8_t mode_, float force, Vect3
robot_pose);
    };
}

#endif
```

Lampiran B : Program Pengolahan Robot di Gazebo (cpp)

```
#include <simulator/robot_gazebo_second.hpp>

unsigned int FIELD_X = 1200;
unsigned int FIELD_Y = 800;

GZ_REGISTER_MODEL_PLUGIN(RobotGazeboSecond)

RobotGazeboSecond::RobotGazeboSecond()
{
}

RobotGazeboSecond::~RobotGazeboSecond()
{
    update_connection.reset();
    ros_node->shutdown();
}

void RobotGazeboSecond::Load(physics::ModelPtr
_model, sdf::ElementPtr)
{
```

```
    // Store the pointer to the model
    robot_model = _model;

    // Condition where the pointer is null
    // The gazebo will retry to load the model until
it finds it
    if (!robot_model)
    {
        ROS_ERROR("Null pointer access: robot_model
is null.");
        return;
    }

    // Get the world pointer
    world = _model->GetWorld();

    // Get the name of the model ex: cyan1
    robot_model_name = _model->GetName();

    // Get the ball model by the ball model name,
"ball"
    ball_model = world->ModelByName(MODEL_BALL);

    // Condition where the ball model is null
    if (!ball_model)
    {
        ROS_FATAL("Robot %s can't detect the ball",
robot_model_name.c_str());
        return;
    }

    // Iterate through all the model to find the
ball
```

```
    // We do this so the index of the ball can be
found
    // And it would help the post processing to be
faster
    for (int i = 0; i < world->ModelCount(); i++)
    {
        // Getting the model by index
        // Reference: https://osrf-
distributions.s3.amazonaws.com/gazebo/api/9.0.0/clas
sgazebo_1_1physics_1_1World.html#aa74c0e9b03a030b69b
d5c475b1367b2e
        if (world->ModelByIndex(i) == ball_model)
        {
            ball_index = i;
            break;
        }
    }

    for (int i = 0; i < TOTAL_CYAN + 1; i++)
        cyan_connection_stat[i] = false;
    for (int i = 0; i < TOTAL_MAGENTA + 1; i++)
        magenta_connection_stat[i] = false;

    // This event is broadcast every simulation
iteration
    // Reference: https://osrf-
distributions.s3.amazonaws.com/gazebo/api/9.0.0/clas
sgazebo_1_1event_1_1Events.html#a0c8c7e6a33ad46f8be4
5b6a33d279fdc
    update_connection =
event::Events::ConnectWorldUpdateBegin(
        // std::bind is used to bind the function to
the class
```

```
        // Reference:
https://en.cppreference.com/w/cpp/utility/functional
/bind
        std::bind(&RobotGazeboSecond::OnUpdate,
this));

    // Check if ROS is not being initialized
    // If not then initialize it
    if (!ros::isInitialized())
    {
        int argc = 0;
        char **argv = NULL;
        // Ros Init Option NoSigintHandler means
that the program will not terminate when ctrl + c is
pressed
        // Reference:
https://docs.ros.org/en/api/roscpp/html/namespaceros
_1_1init__options.html
        ros::init(argc, argv, "robot_gazebo",
ros::init_options::NoSigintHandler);
    }

    /**
     * This line is NECESSARY for a gazebo plugins
     * Node Handle Creation: The ros::NodeHandle is
an essential object in ROS that handles
     * the initialization and shutdown of the node
within the ROS environment. It's used for
     * subscribing to topics, advertising services,
and interacting with parameters on the parameter
server.
     *
```

```
     * Scoped Node Handle: When you use reset on a
boost::shared_ptr or std::shared_ptr (depending on
your version of ROS),
     * you are effectively saying that you want to
replace the managed object with a new one. In this
case, it replaces the
     * current ros::NodeHandle object with a new
instance of ros::NodeHandle. This is important for
ensuring that the node handle
     * is properly scoped within the lifecycle of
the plugin. When the boost::shared_ptr (or
std::shared_ptr) goes out of scope,
     * it ensures the destruction of the
ros::NodeHandle, which will cleanly shutdown all the
ROS communications set up through this handle.
     *
     * Unique Namespace: By specifying
"robot_gazebo" as the namespace for the node handle,
you are ensuring that this plugin's node handle
     * interacts with ROS under a specific
namespace, which can help in managing topics,
services, and parameters, especially when you have
     * multiple instances of the same plugin or
nodes in a larger system.
     *
     * Lazily Initialized ROS Node: The check if
(!ros::isInitialized()) before initializing the ROS
node is there to prevent re-initialization
     * if the ROS system is already up and running.
This is a safety measure to ensure that you do not
accidentally re-initialize ROS, which would
```

```
     * be an error. The ros::NodeHandle constructor
does not initialize ROS; it assumes ROS has already
been initialized, hence the check before
     * the node handle is created.
     *
     * Note: Explained by CHATGPT
     */
    ros_node.reset(new
ros::NodeHandle("robot_gazebo"));

    //---Ros Subscribers

    ros::SubscribeOptions so_gazebo_model =

ros::SubscribeOptions::create<gazebo_msgs::ModelStat
es>("/gazebo/model_states",

16,

boost::bind(&RobotGazeboSecond::CllbckSubGazeboModel
, this, _1),

ros::VoidPtr(),

&ros_queue);

    sub_gazebo_model = ros_node-
>subscribe(so_gazebo_model);

    ros::SubscribeOptions so_kicker_power =

ros::SubscribeOptions::create<std_msgs::Float32>("/p
c2sim/kicker/power",
```

```
16,

boost::bind(&RobotGazeboSecond::CllbckSubKickerPower
, this, _1),

ros::VoidPtr(),

&ros_queue);

    sub_kicker_power = ros_node-
>subscribe(so_kicker_power);

    for (uint8_t i = 1; i < TOTAL_CYAN + 1; i++)
    {
        string topic_name = "/cyan" +
std::to_string(i) + "/pc2sim/velocity";
        ros::SubscribeOptions so_vel_cyan =

ros::SubscribeOptions::create<iris_msgs::robot2sim>(
topic_name,

16,

boost::bind(&RobotGazeboSecond::CllbckSubVelocity,
this, _1),

ros::VoidPtr(),

&ros_queue);

        sub_vel_cyan[i] = ros_node-
>subscribe(so_vel_cyan);
```

```
        topic_name = "/cyan" + to_string(i) +
"/odom";
        pub_cyan_pose[i] = ros_node-
>advertise<geometry_msgs::Pose2D>(topic_name, 16);

        topic_name = "/cyan" + to_string(i) +
"/ball/state";
        pub_cyan_ball_state[i] = ros_node-
>advertise<std_msgs::UInt8>(topic_name, 16);
    }

    for (uint8_t i = 1; i < TOTAL_MAGENTA + 1; i++)
    {
        string topic_name = "/magenta" +
std::to_string(i) + "/pc2sim/velocity";
        ros::SubscribeOptions so_vel_magenta =

ros::SubscribeOptions::create<iris_msgs::robot2sim>(
topic_name,

16,

boost::bind(&RobotGazeboSecond::CllbckSubVelocity,
this, _1),

ros::VoidPtr(),

&ros_queue);

        sub_vel_magenta[i] = ros_node-
>subscribe(so_vel_magenta);
```

```
        topic_name = "magenta" + to_string(i) +
"/odom";
        pub_magenta_pose[i] = ros_node-
>advertise<geometry_msgs::Pose2D>(topic_name, 16);

        topic_name = "/magenta" + to_string(i) +
"/ball/state";
        pub_magenta_ball_state[i] = ros_node-
>advertise<std_msgs::UInt8>(topic_name, 16);
    }

    pub_ball_pose_cyan = ros_node-
>advertise<geometry_msgs::Pose2D>("/cyan/ball/pose",
16);
    pub_ball_pose_magenta = ros_node-
>advertise<geometry_msgs::Pose2D>("/magenta/ball/pos
e", 16);

    pub_friend_poses = ros_node-
>advertise<std_msgs::Float32MultiArray>("/cyan/poses
", 16);
    pub_enemy_poses = ros_node-
>advertise<std_msgs::Float32MultiArray>("/magenta/po
ses", 16);

    pub_total_active_cyan = ros_node-
>advertise<std_msgs::UInt8>("/cyan/total/robot/activ
e", 16);
    pub_total_active_magenta = ros_node-
>advertise<std_msgs::UInt8>("/magenta/total/robot/ac
tive", 16);

    /**
```

```
     * When
std::bind(&RobotGazeboSecond::QueueThread, this) is
passed to the std::thread constructor,
     * it tells the new thread to execute the
QueueThread member function on the current instance
of the
     * RobotGazeboSecond class. Essentially, it sets
up a thread that will handle tasks defined within
     * the QueueThread function, which is often
related to managing a queue of messages or events in
the context of a ROS node.
     *
     * Reference to thread:
https://en.cppreference.com/w/cpp/thread/thread
     */
    ros_queue_thread =
std::thread(std::bind(&RobotGazeboSecond::QueueThrea
d, this));
}

// Main Loop
void RobotGazeboSecond::OnUpdate()
{
    if (world->ModelCount() > 6)
    {
        RobotControl();

        for (int i = 1; i < TOTAL_CYAN + 1; i++)
            if (ros::Time::now().toSec() -
cyan_connection_timer[i] > 1)
                cyan_connection_stat[i] = false;

        for (int i = 1; i < TOTAL_MAGENTA + 1; i++)
```

```
            if (ros::Time::now().toSec() -
magenta_connection_timer[i] > 1)
                magenta_connection_stat[i] = false;
    }
}

// Thread
void RobotGazeboSecond::QueueThread()
{
    static const double timeout = 0.01;
    while (ros_node->ok())
    {

ros_queue.callAvailable(ros::WallDuration(timeout));
    }
}

//---Ros Callbacks
void RobotGazeboSecond::CllbckSubGazeboModel(const
gazebo_msgs::ModelStates::ConstPtr &msg)
{
    if (ball_index >= msg->pose.size())
        return;

    if (world->ModelCount())
    {
        // Virtual ball position
        geometry_msgs::Pose pose = msg-
>pose[ball_index];
        math::Vector3d position(pose.position.x,
pose.position.y, pose.position.z);
        ball_pose.x = position.X();
        ball_pose.y = position.Y();
```

```
        // Virtual velocity
        geometry_msgs::Twist twist = msg-
>twist[ball_index];
    }
}

void RobotGazeboSecond::CllbckSubVelocity(const
iris_msgs::robot2sim::ConstPtr &msg)
{
    int motor_index = msg->coderobot - 10;
    if (motor_index >= 1 && motor_index <= 5)
    {
        vel_cyan_motor[motor_index].x = msg->vx;
        vel_cyan_motor[motor_index].y = msg->vy;
        vel_cyan_motor[motor_index].th = msg->vth;
        cyan_connection_stat[motor_index] = true;
        cyan_connection_timer[motor_index] =
ros::Time::now().toSec();
    }
    else if (motor_index >= 11 && motor_index <= 15)
    {
        motor_index -= 10;
        vel_magenta_motor[motor_index].x = msg->vx;
        vel_magenta_motor[motor_index].y = msg->vy;
        vel_magenta_motor[motor_index].th = msg-
>vth;
        magenta_connection_stat[motor_index] = true;
        magenta_connection_timer[motor_index] =
ros::Time::now().toSec();
    }
}
```

```
void RobotGazeboSecond::CllbckSubKickerPower(const
std_msgs::Float32ConstPtr &msg)
{
    kicker_power = msg->data;
}

//---Robot Control
void RobotGazeboSecond::RobotControl()
{
    std_msgs::UInt8 total_robot_msg;
    uint8_t robot_code = 10;
    active_cyan = 0;
    active_magenta = 0;
    for (int i = 1; i <= TOTAL_CYAN; i++)
    {
        std_msgs::UInt8 ball_status_msg;
        robot_code++;
        ball_status = 1;
        if (cyan_connection_stat[i])
        {
            active_cyan++;
            ComputeRobotPosition();
            ComputeRobotVelocity();
            ComputeBallPosition();
            if (CheckBallStatus(cyan_pose[i]) &&
kicker_power == 0)
            {
                BallDribbling(cyan_pose[i], "cyan" +
to_string(i));
                ball_status = 2;
            }
            if (kicker_power != 0)
            {
```

```
                Kick(1, kicker_power, cyan_pose[i]);
            }

            MoveRobot(robot_code,
final_vel_cyan[i].x, final_vel_cyan[i].y,
final_vel_cyan[i].th);

            PublishData();
        }
        ball_status_msg.data = ball_status;

pub_cyan_ball_state[i].publish(ball_status_msg);
    }
    total_robot_msg.data = active_cyan;
    pub_total_active_cyan.publish(total_robot_msg);
    robot_code = 20;
    for (int i = 1; i <= TOTAL_MAGENTA; i++)
    {
        std_msgs::UInt8 ball_status_msg;
        robot_code++;
        ball_status = 1;
        if (magenta_connection_stat[i])
        {
            active_magenta++;
            ComputeRobotPosition();
            ComputeRobotVelocity();
            ComputeBallPosition();
            if (CheckBallStatus(magenta_pose[i]) &&
kicker_power == 0)
            {
                BallDribbling(magenta_pose[i],
"magenta" + to_string(i));
                ball_status = 2;
```

```
            }
            if (kicker_power != 0)
            {
                Kick(1, kicker_power,
magenta_pose[i]);
            }

            MoveRobot(robot_code,
final_vel_magenta[i].x, final_vel_magenta[i].y,
final_vel_magenta[i].th);

            PublishData();
        }
        ball_status_msg.data = ball_status;

pub_magenta_ball_state[i].publish(ball_status_msg);
    }
    total_robot_msg.data = active_magenta;

pub_total_active_magenta.publish(total_robot_msg);
}

void RobotGazeboSecond::ComputeRobotPosition()
{
    math::Pose3d pose_cyan[TOTAL_CYAN + 1];
    math::Pose3d pose_magenta[TOTAL_MAGENTA + 1];

    for (uint8_t i = 1; i < TOTAL_CYAN + 1; i++)
    {
        string what_robot = "cyan" + to_string(i);
        auto model = world->ModelByName(what_robot);
        if (!model)
            continue;
```

```
        pose_cyan[i] = model->RelativePose();
        cyan_pose[i].x = (pose_cyan[i].Pos().Y() *
100);
        cyan_pose[i].y = (pose_cyan[i].Pos().X() *
100);
        cyan_pose[i].th = -
angles::to_degrees(pose_cyan[i].Rot().Yaw());

        while (cyan_pose[i].th > 180)
            cyan_pose[i].th -= 360;
        while (cyan_pose[i].th < -180)
            cyan_pose[i].th += 360;
    }
    for (uint8_t i = 1; i < TOTAL_MAGENTA + 1; i++)
    {
        string what_robot = "magenta" +
to_string(i);
        auto model = world->ModelByName(what_robot);

        if (!model)
            continue;

        pose_magenta[i] = model->RelativePose();
        magenta_pose[i].x =
(pose_magenta[i].Pos().Y() * 100);
        magenta_pose[i].y =
(pose_magenta[i].Pos().X() * 100);
        magenta_pose[i].th = -
angles::to_degrees(pose_magenta[i].Rot().Yaw());

        while (magenta_pose[i].th > 180)
            magenta_pose[i].th -= 360;
```

```
        while (magenta_pose[i].th < -180)
            magenta_pose[i].th += 360;
    }
}

void RobotGazeboSecond::PublishData()
{
    geometry_msgs::Pose2D robot_data;
    std_msgs::Float32MultiArray cyan_positions;
    std_msgs::Float32MultiArray magenta_positions;

    for (int i = 1; i <= TOTAL_CYAN; i++)
    {
        robot_data.x = cyan_pose[i].x;

cyan_positions.data.push_back(cyan_pose[i].x);
        robot_data.y = cyan_pose[i].y;

cyan_positions.data.push_back(cyan_pose[i].y);
        robot_data.theta = cyan_pose[i].th;
        pub_cyan_pose[i].publish(robot_data);
    }

    for (int i = 1; i <= TOTAL_MAGENTA; i++)
    {
        robot_data.x = magenta_pose[i].x;

magenta_positions.data.push_back(magenta_pose[i].x);
        robot_data.y = magenta_pose[i].y;

magenta_positions.data.push_back(magenta_pose[i].y);
        robot_data.theta = magenta_pose[i].th;
        pub_magenta_pose[i].publish(robot_data);
```

```
    }
    pub_friend_poses.publish(cyan_positions);
    pub_enemy_poses.publish(magenta_positions);

    geometry_msgs::Pose2D pose;
    pose.x = ball_pose_on_real.x;
    pose.y = ball_pose_on_real.y;
    pub_ball_pose_cyan.publish(pose);
}

void RobotGazeboSecond::ComputeRobotVelocity()
{
    for (int i = 1; i <= TOTAL_CYAN; i++)
    {
        final_vel_cyan[i].x = (vel_cyan_motor[i].y *
0.05 * cos(angles::from_degrees(cyan_pose[i].th)) +
                               vel_cyan_motor[i].x *
0.05 * sin(angles::from_degrees(cyan_pose[i].th)));
        final_vel_cyan[i].y = -(vel_cyan_motor[i].y
* 0.05 * sin(angles::from_degrees(cyan_pose[i].th))
+
                                vel_cyan_motor[i].x
* 0.05 *
cos(angles::from_degrees(cyan_pose[i].th)));
        final_vel_cyan[i].th = -vel_cyan_motor[i].th
* 0.05;
    }

    for (int i = 1; i <= TOTAL_MAGENTA; i++)
    {
        final_vel_magenta[i].x = -
(vel_magenta_motor[i].y * 0.05 *
cos(angles::from_degrees(magenta_pose[i].th)) +
```

```
vel_magenta_motor[i].x * 0.05 *
sin(angles::from_degrees(magenta_pose[i].th)));
        final_vel_magenta[i].y =
(vel_magenta_motor[i].y * 0.05 *
sin(angles::from_degrees(magenta_pose[i].th)) +

vel_magenta_motor[i].x * 0.05 *
cos(angles::from_degrees(magenta_pose[i].th)));
        final_vel_magenta[i].th = -
vel_magenta_motor[i].th * 0.05;
    }
}

/**
 * @brief This system should be faster than the
previous method.
 */
void RobotGazeboSecond::MoveRobot(uint8_t
_code_robot, float _vx, float _vy, float _vsudut)
{
    std::string model_name;
    switch (_code_robot)
    {
    case 11:
        model_name = "cyan1";
        break;
    case 12:
        model_name = "cyan2";
        break;
    case 13:
        model_name = "cyan3";
        break;
```

```
    case 21:
        model_name = "magenta1";
        break;
    case 22:
        model_name = "magenta2";
        break;
    case 23:
        model_name = "magenta3";
        break;
    default:
        return;
    }

    gazebo::physics::ModelPtr model = world-
>ModelByName(model_name);
    if (model)
    {
        model->SetLinearVel(math::Vector3d(_vx, _vy,
0.0));
        model->SetAngularVel(math::Vector3d(0.0,
0.0, _vsudut));
    }
}

void RobotGazeboSecond::ComputeBallPosition()
{
    ball_pose_on_real.x = ball_pose.y * 100;
    ball_pose_on_real.y = ball_pose.x * 100;
}

bool RobotGazeboSecond::CheckBallStatus(Vect3
robot_pose)
{
```

```
    float dist_to_ball = sqrt(pow(robot_pose.x -
ball_pose_on_real.x, 2) + pow(robot_pose.y -
ball_pose_on_real.y, 2));
    float robot_th = robot_pose.th + 90;

    while (robot_th > 180)
        robot_th -= 360;
    while (robot_th < -180)
        robot_th += 360;

    float theta_to_ball = atan2(ball_pose_on_real.y
- robot_pose.y, ball_pose_on_real.x - robot_pose.x)
* RAD2DEG;
    while (theta_to_ball > 180)
        theta_to_ball -= 360;
    while (theta_to_ball < -180)
        theta_to_ball += 360;

    float error_theta = theta_to_ball - robot_th;

    if (dist_to_ball < 35 && fabs(error_theta) < 10)
        return true;

    return false;
}

void RobotGazeboSecond::BallDribbling(Vect3
robot_pose, string robot_model_name)
{
    math::Pose3d pose;
    math::Quaterniond orientation;
    math::Vector3d position;
```

```
    pose = world->ModelByName(robot_model_name)-
>RelativePose();
    position = pose.Pos();
    orientation = pose.Rot();

    float target_x = 0.26 *
cos(angles::from_degrees(robot_pose.th));
    float target_y = 0.26 *
sin(angles::from_degrees(robot_pose.th));

    target_y = -target_y;

    math::Vector3d target_pos(target_x, target_y,
0.0);
    math::Pose3d target_pose(position + target_pos,
orientation);
    ball_model->SetLinearVel(math::Vector3d::Zero);
    ball_model->SetRelativePose(target_pose);
}

void RobotGazeboSecond::Kick(uint8_t mode_, float
force, Vect3 robot_pose)
{
    if (!ball_status)
        return;

    Vect3 ball_vel;
    float ball_velocity;

    float robot_th = robot_pose.th + 90;

    while (robot_th > 180)
        robot_th -= 360;
```

```
    while (robot_th < -180)
        robot_th += 360;

    float theta_to_ball = atan2(ball_pose_on_real.y
- robot_pose.y, ball_pose_on_real.x - robot_pose.x)
* RAD2DEG;
    while (theta_to_ball > 180)
        theta_to_ball -= 360;
    while (theta_to_ball < -180)
        theta_to_ball += 360;

    // Convert the force to velocity
    ball_velocity = force;

    ball_vel.x = ball_velocity *
sin(angles::from_degrees(theta_to_ball));
    ball_vel.y = ball_velocity *
cos(angles::from_degrees(theta_to_ball));

    math::Vector3d target_velocity;
    target_velocity.Set(ball_vel.x, ball_vel.y,
0.0);
    ball_model->SetLinearVel(target_velocity);
}
```

Lampiran C : Program Referee Box (header)

```
#ifndef REFEREE_HPP
#define REFEREE_HPP

#include <functional>
#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/common/common.hh>
```

```
#include <cstdlib>
#include <fstream>
#include <string.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <string>

#include <ros/ros.h>
#include <ros/callback_queue.h>
#include <ros/subscribe_options.h>
#include <std_msgs/UInt8.h>

#include <logger/logger.hpp>

logger::Logger logger_sim;

using namespace gazebo;
using namespace ignition;
using namespace std;

namespace gazebo
{
    class RefereeBox : public WorldPlugin
    {
    public:
        RefereeBox();
        ~RefereeBox();
        void Load(physics::WorldPtr _world,
sdf::ElementPtr _sdf);
        void OnUpdate();

        void QueueThread();
```

```
        ros::Publisher pub_command_referee;

    private:
        event::ConnectionPtr update_connection;
        std::unique_ptr<ros::NodeHandle> ros_node;
        std::thread ros_queue_thread;
        ros::CallbackQueue ros_queue;

        uint8_t command_referee;

        int Kbhit();
        void KeyboardHandler();
        void PublishData();
    };
};

#endif
```

Lampiran D : Program Referee Box (cpp)

```
#include <simulator/referee.hpp>

GZ_REGISTER_WORLD_PLUGIN(RefereeBox)

RefereeBox::RefereeBox()
{
}

RefereeBox::~RefereeBox()
{
    update_connection.reset();
    ros_node->shutdown();
}
```

```
void RefereeBox::Load(physics::WorldPtr _world,
sdf::ElementPtr _sdf)
{
    command_referee = 'S';

    update_connection =
event::Events::ConnectWorldUpdateBegin(
        std::bind(&RefereeBox::OnUpdate, this));

    if (!ros::isInitialized())
    {
        int argc = 0;
        char **argv = NULL;
        // Ros Init Option NoSigintHandler means
that the program will not terminate when ctrl + c is
pressed
        // Reference:
https://docs.ros.org/en/api/roscpp/html/namespaceros
_1_1init__options.html
        ros::init(argc, argv, "referee_box",
ros::init_options::NoSigintHandler);
    }

    ros_node.reset(new
ros::NodeHandle("referee_box"));

    pub_command_referee = ros_node-
>advertise<std_msgs::UInt8>("/referee/sim/command",
16);
```

```
    ros_queue_thread =
std::thread(std::bind(&RefereeBox::QueueThread,
this));
}

void RefereeBox::OnUpdate()
{
    KeyboardHandler();
    PublishData();
}

void RefereeBox::QueueThread()
{
    static const double timeout = 0.01;
    while (ros_node->ok())
    {

ros_queue.callAvailable(ros::WallDuration(timeout));
    }
}

int RefereeBox::Kbhit()
{
    static const int STDIN = 0;
    static bool initialized = false;

    if (!initialized)
    {
        termios term;
        tcgetattr(STDIN, &term);
        term.c_lflag &= ~ICANON;
        tcsetattr(STDIN, TCSANOW, &term);
        setbuf(stdin, NULL);
```

```
        initialized = true;
    }

    int bytesWaiting;

    if (ioctl(STDIN, FIONREAD, &bytesWaiting) == -1)
        return 0;

    return bytesWaiting;
}

void RefereeBox::KeyboardHandler()
{
    if (Kbhit() > 0)
    {
        char key = std::cin.get();

        command_referee = key;
    }
}

void RefereeBox::PublishData()
{
    std_msgs::UInt8 msg;
    msg.data = command_referee;
    pub_command_referee.publish(msg);
}
```

Lampiran E : Program Komunikasi Robot Ke Gazebo (header)

```
#ifndef COMM_SIMULATOR_H
#define COMM_SIMULATOR_H
```

```
#include <ros/ros.h>

#include <sys/socket.h>
#include <arpa/inet.h>
#include <arpa/inet.h>
#include <sys/types.h>

#include "geometry_msgs/Twist.h"
#include "geometry_msgs/Pose2D.h"

#include "iris_msgs/pc2stm.h"
#include "iris_msgs/stm2pc.h"
#include "iris_msgs/robot2sim.h"
#include "std_msgs/Int16MultiArray.h"

#include <cv_bridge/cv_bridge.h>
#include <image_transport/image_transport.h>
#include <opencv2/highgui.hpp>
#include <opencv2/opencv.hpp>

#include <boost/thread/mutex.hpp>

using namespace cv;
using namespace std;

uint16_t g_res_x = 360;
uint16_t g_res_y = 640;

//---Timer
ros::Timer main_timer;

ros::Subscriber sub_sim_pose;
```

```
ros::Publisher pub_odometry;
ros::Publisher pub_vel_pc2sim;
ros::Publisher pub_final_pose;

image_transport::Subscriber sub_sim_cam;
image_transport::Publisher pub_raw_frame;

std::string robotns;
int8_t coderobot;
std::string robot_name;

boost::mutex mutex_frame_sim;

Mat frame_bgr;
Mat frame_yuv;

void CllbckSend2Pc(const ros::TimerEvent &);
void CllbckSimPose(const
geometry_msgs::Pose2DConstPtr &msg);
void CllbckSimCam(const sensor_msgs::ImageConstPtr
&msg);

#endif
```

Lampiran F : Program Komunikasi Robot Ke Gazebo (cpp)

```
#include "comm/comm_stm.h"
#include "comm/comm_simulator2.h"

#include <stdio.h>
using namespace std;

int main(int argc, char **argv)
{
```

```
    ros::init(argc, argv, "comm_simulator");
    ros::NodeHandle NH;
    ros::MultiThreadedSpinner spinner(0);
    ros::param::get("robot_name", namarobot);
    robotns = ros::this_node::getNamespace();

    std::string topic_pc2sim = "/" + robotns +
"/pc2sim/velocity";
    robotns = ros::this_node::getNamespace();

    image_transport::ImageTransport IT(NH);

    for (int i = 1; i <= 5; i++)
    {
        if (robotns.compare("/cyan" + to_string(i))
== 0)
        {
            coderobot = 10 + i;
            namarobot = "cyan" + to_string(i);
        }
        else if (robotns.compare("/magenta" +
to_string(i)) == 0)
        {
            coderobot = 20 + i;
            namarobot = "magenta" + to_string(i);
        }
    }

    send2pc_timer =
NH.createTimer(ros::Duration(0.01), CllbckSend2Pc);
    // pub_odometry =
NH.advertise<geometry_msgs::Pose2D>(topic_pc2sim,
16);
```

```
    sub_camera_simulator = IT.subscribe("/" +
namarobot + "/omni_cam/omni_image_raw", 16,
CllbckCameraSimulator);
    pub_raw_frame = IT.advertise("/vision_yuv", 32);

    pub_vel_pc2sim =
NH.advertise<iris_msgs::robot2sim>(topic_pc2sim,
16);
    pub_stm2pc =
NH.advertise<iris_msgs::stm2pc>("stm2pc", 16);
    pub_final_pose =
NH.advertise<geometry_msgs::Pose2D>("/" + namarobot
+ "/pose_robot", 16);
    // send_master_timer =
NH.createTimer(ros::Duration(0.01),
CllbckSendMaster);
    // send_timer =
NH.createTimer(ros::Duration(0.01), CllbckSend);
    sub_posisi_simulator = NH.subscribe("/" +
namarobot + "/odom", 16, CllbckPosisiSimulator);
    sub_vel_data = NH.subscribe("/" + namarobot +
"/cmd_vel_motor", 3, CllbckVelMotor);
    sub_pc2stm = NH.subscribe("/pc2stm", 16,
CllbckPc2Stm);
    // sub_dribble_vel = NH.subscribe("dribble_vel",
2, CllbckVelDribble);

    ROS_WARN("NAMA ROBOT: %s", namarobot.c_str());

    spinner.spin();
    send2pc_timer.stop();

    return 0;
```

```
}

void CllbckCameraSimulator(const
sensor_msgs::ImageConstPtr &msg)
{
    try
    {
        mutex_frame_sim.lock();
        frame_bgr = cv_bridge::toCvShare(msg,
"bgr8")->image.clone();
        cvtColor(frame_bgr, frame_published,
CV_BGR2HSV);
        mutex_frame_sim.unlock();
    }
    catch (cv_bridge::Exception &e)
    {
        ROS_ERROR("Could not convert from '%s' to
'bgr8'.", msg->encoding.c_str());
    }

    mutex_frame_sim.lock();
    resize(frame_published, frame_published,
Size(g_res_x, g_res_y));
    mutex_frame_sim.unlock();

    sensor_msgs::ImagePtr msg_yuv_sim =
cv_bridge::CvImage(std_msgs::Header(), "bgr8",
frame_published).toImageMsg();
    pub_raw_frame.publish(msg_yuv_sim);
}

void CllbckPosisiSimulator(const
geometry_msgs::Pose2DConstPtr &msg)
```

```
{
    robot_pos[0] = msg->x;
    robot_pos[1] = msg->y;
    robot_pos[2] = msg->theta + 90;
    // TODO: The theta should be inverted, the 0
become 180 and the 180 become -180
    while (robot_pos[2] > 180)
        robot_pos[2] -= 360;
    while (robot_pos[2] < -180)
        robot_pos[2] += 360;
}

void CllbckPc2Stm(const iris_msgs::pc2stmConstPtr
&msg)
{
    odom_offset[0] = msg->odom_offset_x;
    odom_offset[1] = msg->odom_offset_y;
    odom_offset[2] = msg->odom_offset_th;
    kicker_mode = msg->kicker_mode;
    kicker_power = msg->kicker_power;
    kicker_position = msg->kicker_position;
    buzzer_cnt = msg->buzzer_cnt;
    buzzer_time = msg->buzzer_time;
}

void CllbckSend2Pc(const ros::TimerEvent &event)
{
    iris_msgs::stm2pc msg;
    msg.odom_buffer_x = robot_pos[0];
    msg.odom_buffer_y = robot_pos[1];
    msg.odom_buffer_th = robot_pos[2];
    msg.line_sensors = 0;
    msg.gp_sensors.push_back(0);
```

```
    msg.gp_sensors.push_back(0);
    msg.ethernet_communication = 0;
    pub_stm2pc.publish(msg);

    geometry_msgs::Pose2D msgpose;
    msgpose.x = robot_pos[0];
    msgpose.y = robot_pos[1];
    msgpose.theta = robot_pos[2];
    pub_final_pose.publish(msgpose);

    iris_msgs::robot2sim msg2sim;
    msg2sim.coderobot = coderobot;
    msg2sim.vx = velocity_robot[0];
    msg2sim.vy = velocity_robot[1];
    msg2sim.vth = velocity_robot[2];

    pub_vel_pc2sim.publish(msg2sim);
}
void CllbckSend(const ros::TimerEvent &event)
{
}

void CllbckVelMotor(const
geometry_msgs::TwistConstPtr &msg)
{

    velocity_robot[0] = (msg->linear.x);
    velocity_robot[1] = (msg->linear.y);
    velocity_robot[2] = (msg->angular.z);

    // printf("%d \n", velocity_robot[0]);
}
```

```
void CllbckVelDribble(const
std_msgs::Int16MultiArrayConstPtr &msg)
{
    left_dribble_speed = msg->data.at(0);
    right_dribble_speed = msg->data.at(1);
}
```

Lampiran G : Program Perintah Spawn Robot dan Bola (bash)

```
#!/bin/bash

jml_cyan=3
jml_magenta=3

cyan_x=(0.0 0.0 0.0 1.0)
cyan_y=(0.0 -1.0 0.0 -1.0)
magenta_x=(0.0 11.5 4.5 9.0)
magenta_y=(0.0 4.0 2.0 6.0)

sleep 1


### spawn cyan robots
for ((i=1; i<=$jml_cyan; ++i))
do
    rosrun gazebo_ros spawn_model -file $(rospack
find simulator)/models/iris_cyan/model.sdf -sdf \
                                    -model cyan${i} \
                                    -x ${cyan_x[$i]} -
y ${cyan_y[$i]} -z 0.1 &
    sleep 0.5
done
```

```
## spawn magenta robots
for ((i=1; i<=$jml_magenta; ++i))
do
    rosrun gazebo_ros spawn_model -file $(rospack
find simulator)/models/robot_magenta/model.sdf -sdf
\
                                -model magenta${i}
\
                                -x
${magenta_x[$i]} -y ${magenta_y[$i]} -z 1 &
    sleep 0.5
done
```

Lampiran H : Pengolahan Data Pada Workspace Robot 1 (cpp)

```
if (is_sim)
    {
        string robot_type =
robot_namespace.substr(0, 5);
        string topic_name = robot_type +
"/ball/pose";
        ROS_WARN("MASTER TOPIC: %s",
topic_name.c_str());
        sub_sim_ball_pose = NH.subscribe(topic_name,
1, CllbckSubBallSimPose);
        pub_kicker_power =
NH.advertise<std_msgs::Float32>("/pc2sim/kicker/powe
r", 1);
        topic_name = robot_namespace +
"/ball/state";
```

```
        ROS_WARN("ROBOT NAME: %s",
robot_namespace.c_str());
        sub_sim_ball_state =
NH.subscribe(topic_name, 1, CllbckSubBallSimStat);
        sub_sim_friend_pose =
NH.subscribe("/cyan/poses", 1, CllbckSubFriendPose);
        sub_sim_enemy_pose =
NH.subscribe("/magenta/poses", 1,
CllbckSubEnemyPose);
        // sub_sim_active_cyan = NH
        ROS_ERROR("ROBOT NAMESPACE %s",
robot_namespace.c_str());
        if (robot_namespace.find("cyan") !=
std::string::npos)
        {
            ROS_ERROR("ROBOT IS CYAN TEAM");
            sub_sim_active_cyan =
NH.subscribe("/cyan/total/robot/active", 1,
CllbckSubTotalActiveCyanSim);
        }
        else
        {
            ROS_ERROR("ROBOT IS MAGENTA TEAM");
            sub_sim_active_magenta =
NH.subscribe("/magenta/total/robot/active", 1,
CllbckSubTotalActiveMagentaSim);
        }

        sub_sim_referee =
NH.subscribe("/referee/sim/command", 1,
CllbckSubRefereeBoxSim);
    }
```

Lampiran I : Pengolahan Data Pada Workspace Robot 2 (cpp)

```
void CllbckSubBallSimPose(const
geometry_msgs::Pose2DConstPtr &msg)
{
    ball_on_field[0] = msg->x;
    ball_on_field[1] = msg->y;
    ball_on_field[2] = atan2(pos_robot[1] -
ball_on_field[1], pos_robot[0] - ball_on_field[0]) *
RAD2DEG - 180;

    while (ball_on_field[2] > 180)
        ball_on_field[2] -= 360;

    while (ball_on_field[2] < -180)
        ball_on_field[2] += 360;
}

void CllbckSubBallSimStat(const
std_msgs::UInt8ConstPtr &msg)
{
    ball_status = msg->data;
}

void CllbckSubFriendPose(const
std_msgs::Float32MultiArrayConstPtr &msg)
{
    // first index is x, second is y
    // third index is x, fourth is y
    uint8_t iter = 0;
```

```
    for (uint8_t i = 0; i < msg->data.size(); i +=
2)
    {
        const float dx = msg->data[i] -
pos_robot[0];
        const float dy = msg->data[i + 1] -
pos_robot[1];
        const float distance = __builtin_sqrtf(dx *
dx + dy * dy);
        if (distance < 20)
            continue;

        friend_pose[iter] = {msg->data[i], msg-
>data[i + 1]};
        iter++;
    }
}
void CllbckSubEnemyPose(const
std_msgs::Float32MultiArrayConstPtr &msg)
{
    // first index is x, second is y
    // third index is x, fourth is y
    uint8_t iter = 0;
    for (uint8_t i = 0; i < msg->data.size(); i +=
2)
    {
        enemy_pose[iter].x = msg->data[i];
        enemy_pose[iter].y = msg->data[i + 1];
        iter++;
    }
}
```

```
void CllbckSubTotalActiveCyanSim(const
std_msgs::UInt8ConstPtr &msg)
{
    n_active_robot = msg->data;
    robot_role = 3;
    if (n_active_robot > 1)
    {
        if (robot_namespace.find("1") !=
std::string::npos)
            robot_role = 0;
        else if (robot_namespace.find("2") !=
std::string::npos)
            robot_role = 1;
        else if (robot_namespace.find("3") !=
std::string::npos)
            robot_role = 3;
    }
}

void CllbckSubTotalActiveMagentaSim(const
std_msgs::UInt8ConstPtr &msg)
{
    n_active_robot = msg->data;
    robot_role = 1;
    if (n_active_robot > 1)
    {
        if (robot_namespace.find("1") !=
std::string::npos)
            robot_role = 0;
        else if (robot_namespace.find("2") !=
std::string::npos)
            robot_role = 1;
```

```
        else if (robot_namespace.find("3") !=
std::string::npos)
            robot_role = 3;
    }
}

void CllbckSubRefereeBoxSim(const
std_msgs::UInt8ConstPtr &msg)
{
    static uint8_t prev_key = 0;
    if (prev_key == msg->data)
        return;
    switch (msg->data)
    {
    case 'S':
        if (game_status > 127 && game_status <= 255)
            game_status -= 128;
        game_status = 0;
        break;
    case ' ':
        if (game_status > 127 && game_status <= 255)
            game_status -= 128;
        game_status = 0;
        break;
    case 's':
        if (game_status > 0 && game_status <= 127)
            game_status += 128;
        break;
    case '2':
        game_status = 69;
        break;
    case '#':
        game_status = status_callibration;
```

```
            break;
        case 'K':
            game_status =
status_preparation_kickoff_home;
            break;
        case 'F':
            game_status =
status_preparation_freekick_home;
            break;
        case 'C':
            game_status =
status_preparation_cornerkick_home;
            break;
        case 'G':
            game_status =
status_preparation_goalkick_home;
            break;
        case 'P':
            game_status =
status_preparation_penaltykick_home;
            break;
        case 'T':
            game_status =
status_preparation_throwin_home;
            break;
        case 'k':
            game_status =
status_preparation_kickoff_away;
            break;
        case 'f':
            game_status =
status_preparation_freekick_away;
            break;
```

```
    case 'c':
        game_status =
status_preparation_cornerkick_away;
        break;
    case 'g':
        game_status =
status_preparation_goalkick_away;
        break;
    case 'p':
        game_status =
status_preparation_penaltykick_away;
        break;
    case 't':
        game_status =
status_preparation_throwin_away;
        break;
    }
    robot_base_action = (msg->data != 'S' && msg-
>data != ' ');
    prev_key = msg->data;
}
```