MANIPULATOR ROBOT SIMULATOR AS A LEARNING TOOL ON ROBOTICS COURSE

Ariadie Chandra Nugraha
Department of Electrical Engineering Education, Yogyakarta State University
ariadie@yahoo.com

Abstract

To address the lack of manipulator robot availability as a practice tool in robotics learning, we propose a robot simulation software, which we named SIRUPP (Simulator Robot Manipulator untuk Pembelajaran dan Perancangan/ Manipulator Robot Simulator for Learning and Designing). SIRUPP is a simulator application that enable user to simulate manipulator robot movement in a virtual work environment and visualize the simulation in 3D. One of main components of SIRUPP is physics simulator module. The objective of this effort is to design and implement physics simulator module based on robot kinematics, using Denavit-Hartenberg (D-H) notation, and dynamics, using link's mass and joint's friction parameters. This module will be built based on a physics engine, i.e. Open Dynamics Engine (ODE).

Functionally, the module is capable of modelling kinematics and dynamics of manipulator robot, translating kinematics and dynamics parameters to a virtual robot, controlling the virtual robot with a controlling method, and saving the simulation data to text file for further analysis. In validation test, we compare the movement of virtual robot with the movement of real robot, i.e. PUMA 260, when we control them with equivalent controlling methods. The results showed that simulator module is capable to simulate real robot movement with small error, so that the simulator can be used as a learning tool in robotics.

Keywords: robot simulation, manipulator robot, physics engine, robotics learning, SIRUPP, ODE, PUMA

A. INTRODUCTION

In robotics learning, a manipulator robot is often used as an example to explain the concept of robot kinematics and dynamics. However, as a learning tool, a manipulator robot or its trainer is quite expensive. This problem can be addressed by using a robot simulator application. Today, many robot simulators can be downloaded from the Internet and is free. Several manipulator robot simulators have been developed. Some of them are: "RoboSim" by Dr. Thomas

Braunl [2], “Perangkat Lunak Visualisasi Tiga Dimensi Kinematika Lengan Robot” by Rachmat Hariyanto [5], and “Modelling dan Simulasi Suatu Sistem Lengan Robot” by Ir. Wahidin Wahab, MSc, PhD [7]. Still there is no manipulator robot simulator that can meet the needs for learning kinematics and dynamics of manipulator robot.

Based on this need, we develop SIRUPP (Simulator Robot Manipulator untuk Pembelajaran dan Perancangan/ Manipulator Robot Simulator for Learning and Designing). One of the major parts of SIRUPP is a simulator module that will simulate the behaviour of real robots in the virtual world. Simulator module will be built based on physics engine, a computer program that simulate model of Newtonian physics, using variables such as mass, velocity, and friction. Physics engine is usually a software library that can be combined with other components to form a complex application. Currently, there are quite a lot of physics engines, whether they are commercial or open source, such as PhysX, Havok, Newton, Open Dynamics Engine (ODE), and Bullet.

The use of physics engine in robot simulations has been done in some researches, like research by Pepper, Balakirsky, and Scrapper (2007) [4]. That research using Karma Physics Engine, a physics engine used in Unreal Tournament game. A research conducted in NASA by Lorenzo Flückiger, et al [3] using Mission Simulation Dynamics Engine, which was developed from Open Dynamics Engine (ODE).

This paper will describe the design and implementation of the manipulator robot simulator based on kinematics and dynamics model of robots, such that movement of the virtual robot in simulation has similar trajectory and response time compare with the real robot movement, in this paper is a PUMA 260 robot

B. SIMULATOR MODULE SPECIFICATIONS

SIRUPP subsystem (Figure 1) consists of 7 modules that have their respective duties. Focus of this paper is the Modelling and Simulation subsystem. In this paper, Modelling and Simulation subsystem, 3D Visualization subsystem and part of User Interface subsystem that related to Modelling and Simulation subsystem will be called simulator module.

Physics simulator module will have abilities as follow.

1) Capable of modelling kinematics of a manipulator robot in general, specifically the PUMA 260 robot, including dealing with Denavit-Hartenberg notation (DH) which is usually used to represent the geometrical form of a manipulator robot.
2) Capable of modelling parameters of the manipulator robot dynamics, including mass link and friction on the joints.
3) Capable of translating the virtual robot model robot to a virtual robot object in virtual world's physics engine.
4) Capable of controlling the virtual robot manually, input target angle is determined directly by users, or automatically, input target angle is taken from a movement plan. This movement plan can be obtained by loading the contents of a text file, or as results of motion planning module.
5) Capable of simulating kinematics of the robot, both forward kinematics and reverse kinematics, in the virtual world's physics engine.
6) Capable of simulating movement of the robot when a certain method of control (robot dynamics), namely trapezoidal velocity profiling Proportional-Integral-derivative (PID) filter is applied.
7) Capable of storing data to the simulation results in a text file for further analysis. Stored data consist of robot joint angles during the simulation.

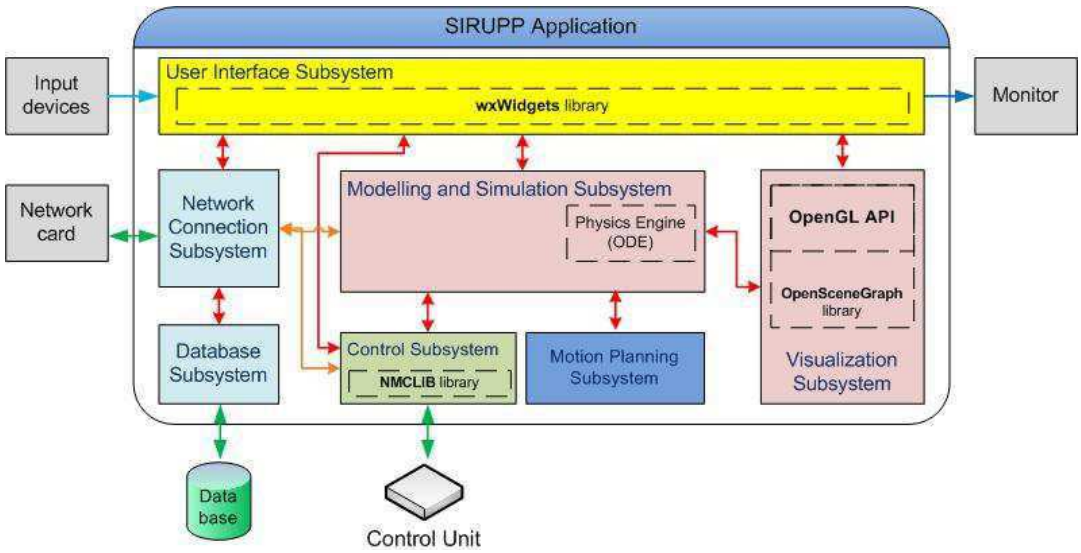

Fig. 1 Block Diagram of SIRUPP.

A. SIMULATOR MODULE DESIGN

1. Kinematics Model based on D-H Notation Translation

On this experiment, the assignment of coordinate frames of each joint of PUMA 260 is shown in Fig. 2. The corresponding DH parameters are described in Table I. Once the position, orientation and link's length parameters as well as the position and orientation of joints parameters are obtained by translating the DH parameters, a virtual robot in the physics engine's virtual world will be built based on that parameters. Basically, each link will be translated into an object in the virtual world, while every joint will be translated into a virtual hinge. DH parameters will be used to form transformation matrix that will determine the initial positions and orientations of virtual robot links and joints.

Table I D-H Parameters for PUMA 260.

i	α_i	a_i (inch)	d_i (inch)	θ_i
1	-90	0	13	θ_1
2	0	8	-3,2	θ_2
3	90	0	-2,4	θ_3
4	-90	0	8	θ_4
5	90	0	0	θ_5
6	0	0	2,2	θ_6

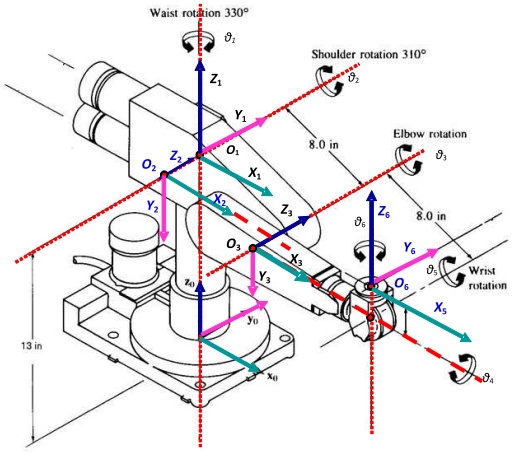

Fig. 2 Coordinate frame assignment of PUMA 260.

2. Dynamics Model Translation

Parameters that affect the dynamics of the robot are J_{eff} (effective inertia moment) and f_{eff} (effective friction coefficient). In ODE physics engine, the link's inertia moment will be calculated by physics engine based on object's mass, the

object's geometrical form, and the object's centre of mass. The friction coefficients will be represented by joint's friction parameter (`dParamFMax` parameter in ODE).

PUMA's specification sheets stated that arm's weight is 6.8 kg, but did not state the weight of each link [9]. According to Riyad Ahmad Firdaus [1], the link 1's mass (including link 0) = 3 kg, the link 2's mass = 1.5 kg, so the mass of link 3 and wrist part = 2.3 kg. These values will be used in the virtual robot model of PUMA 260.

3. Simulator Module Design

Overall block diagram of the simulator are shown in Fig. 3. Programming paradigm that will be used in SIRUPP application development is object-orientated programming, so these subsystems will be implemented in classes.

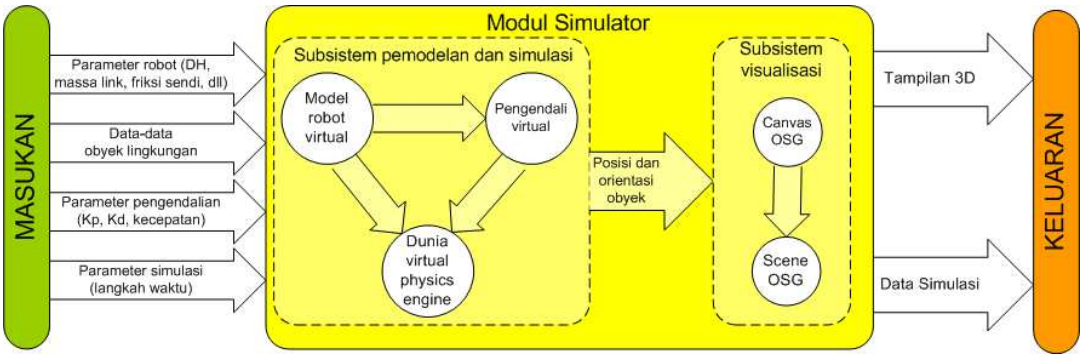

Fig. 3 Block diagram of simulator module.

Modelling and Simulation Subsystem has two main functions. First, this subsystem is responsible for modelling simulated robots and objects in its environment. This subsystem is also responsible for managing the simulation process and processing the simulation data.

3D visualization will use OpenGL API. In the development of this Visualization Subsystem, OpenGL API will not be accessed directly, but via the OpenSceneGraph library that wrap up the OpenGL API. The use of OpenSceneGraph will make visualization easier because it support various advanced features such as lighting and texturing of 3D objects, a relatively complex process when performed using the OpenGL API directly. This subsystem will retrieve the visualization data from Modelling and Simulation subsystem. The

3D rendering results of this subsystem is shown in the visualization panel managed by User Interface subsystem.

User Interface Subsystem is the basis of application framework. When SIRUPP is executed, first the application will initialize a main window which is the parent of application's panels and dialogues, that will be user interfaces for others subsystems. The use of wxWidgets library as a basis for user interfaces development make the development is quicker, because lots of ready to use widgets have been included in this library.

B. SIMULATOR MODULE IMPLEMENTATION

After the Modelling and Simulation Subsystem is implemented, initial testing is conducted to determine the appropriate time step for the simulation. ODE provides two functions for stepping the time in the virtual world, i.e. `dWorldStep` and `dWorldQuickStep` [6]. For a large system, `dWorldQuickStep` is far faster than `dWorldStep`, but less accurate. In this simulator module, the `dWorldStep` function is used. Function `dWorldStep` is selected because `dWorldQuickStep` have a problem when dealing with friction in the joints, one thing that is very important to simulate manipulator robot.

In this Visualization Subsystem, class `srOSGScene` will retrieve data from `srRobotModel`, `srODEWorld`, and `srSimEnv`, and translate the objects defined in those classes to visual objects. The data retrieving will be done in accordance to the simulation steps. To display the visualization, srOSGCanvas class will render of visual objects in srOSGScene to the wxWidgets panel. The process of rendering will be done when the user interface idle. Thus, although the visual object srOSGScene always in sync with the represented object, the rendering process to the user interface does not sync with the simulation steps. This mechanism is used so that the simulation process does not burdened by the rendering process to user interface.

To allow users to access Modelling and Simulation Subsystem that has been developed, the interfaces is needed that allows users to enter and change the robot modelling parameters and manage a simulation run. Fig. 4 displays the main window that implemented User Interface Subsystem. In a simulation run, in the Joint Angle panel, users can enter the target of robot joints, and press "Apply"

button to command the robot move to that target. In the End Effector panel, users can enter the target end effector position of a robot. When users press “Apply”, modelling subsystem will calculate target angles of robot (inverse-kinematics) and apply these angles to virtual robot. Boxes with light blue background colour on the right side of panel shows the actual values of joint angles and end-effector position of virtual robot at the time.

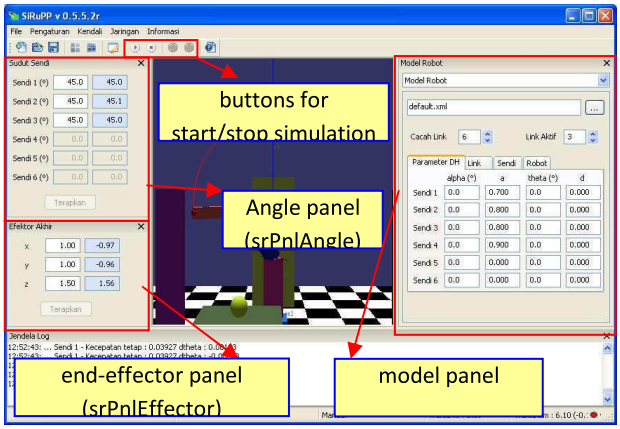

Fig. 4 SIRUPP’s main window.

Robot Model panel can be displayed by selecting menu Settings → Robot Model Setting. Given many parameters involved, a notebook widget which has multiple pages is used in this panel. Fig. 4 shows the D-H parameters page is active. On this page users can enter the number of link from the simulated robot and the values of D-H parameters. Changes that have been made can be saved to a file by selecting menu File → Save File.

In Model Robot panel, other than the DH Parameters page, there are also Links, Joints, and Robot page. In Link page, users can enter geometry form, mass, and size of each link. In Joints page, users can enter the angle limits and friction of virtual robot joints. In this page users can also change the parameters related to virtual robot control, like speed, acceleration, and maximum speed of virtual robot joint’s movement. In Robot page, users can enter position and orientation of the virtual robot’s base.

In addition to above panels, there are also Environment Objects panel and Simulation Settings panel. Environment Object panel displays models of objects in the virtual robot‘s environment. In this panel, users can load an environment

file with XML format that has been created with a text editor. In Simulation Settings panel user can set the control parameters, such as the value of Kp, Kd, and Ki.

When a simulation is running, either manually or automatically, it is possible to link virtual robot's movement to real robot's movement. When linked with the virtual robot, the real robot will do similar movement with movement commanded to virtual robot. To enable the linking, an interface to initialize and calibrate joint position of real robot, before link the real robot to the virtual one, is developed.

C. SIMULATOR MODULE TESTING

1. Testing Procedure

Simulator module testing is done through two-stage testing, the functional testing and simulation testing. Functional testing is done to check whether the built module has met planned specifications. This testing will be done through the seven test cases, each to test whether the simulator module is able to fulfil the specifications, which has been described above. Conducted testing indicates that all planned specifications have been met.

Second-stage testing is simulation performance testing, which is used to compare the virtual robot simulation results with the behaviour of real robots when an equivalent movement set and methods of control are applied. Simulation testing will be done by move manipulator robot arm according to defined movement sets, which consists of 7 sets. Movements can be divided into 3 groups: 1) movement of one joint, 2) movement of two joints simultaneously, and 3) movement of three joints simultaneously. Movement testing is limited only to the first three joints that make up the arm of PUMA 260 robot.

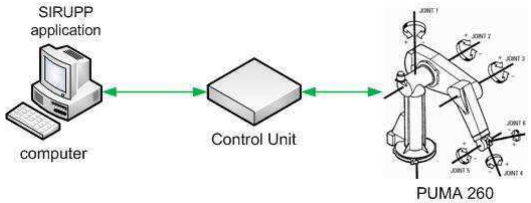

Fig. 5 Simulation tests configuration.

Real robot's control use a control unit based on PIC-Servo chip. PIC-Servo supports three control modes, i.e. speed mode, position mode, and PWM mode [8]. Given the control mode usually used to control PUMA robot is position mode, then this experiment use the position mode with trapezoidal profiling. Parameters for `ServoSetGain` command will use the default values.

Virtual robot's control will simulate the real robot's control. Virtual robot will be controlled by VRCtrlTrapez class which is the implementation of the trapezoidal velocity profiling algorithm. Because parameter's unit used in physics simulation is different with parameter's unit used in real robot control, it is needed to converse from values of the real robot parameter to the virtual robot parameters.

In the trapezoidal profiling control with PIC-Servo, PID control filter used on each servo tick after target position for that servo tick is calculated with the trapezoidal profiling. For real robot's control, the values of the parameters Kp set to 100, Kd = 1000, and Ki = 0. These values will be converted to the value of the virtual robot's control parameters, i.e. Kpv, Kdv, and Kiv.

For each set of defined movements, the following steps will be done.

1) Run SIRUPP application.
2) Load a PUMA 260 virtual robot model.
3) Initialize connection to real robot via Real Robot Control dialog.
4) Change simulation mode to Automatic, if current simulation model is Manual.
5) Load the movement plan file which contains movement set that will be tested.
6) Run the movement plan by pressing the button (Start movement plan) on the toolbar. Once movement plan is completed, a log data file will be created.

Simulation data files will contain set point theta, virtual robot's theta and real robot's theta for joints 1, 2, and 3. Testing with linked real robot are only needed to get the real robot‘s movement data. For testing that only need to obtain virtual robot's movement data, real robots do not need to be linked.

2. Simulation Tests Results

From the one joint movement tests results, it is shown that that the real robot did not move instantaneously when a new set point is given, but there is

delay before the joint moved. This delay is 0.35-0.4 second for joint 1. It's happened because trapezoidal velocity profiling would provide a position set that relatively small at the beginning of the movement, which in turn caused the small input error for the PID filter, small value of PWM sent to motor driver, and eventually the generated torque was not enough to overcome static friction in the joint.

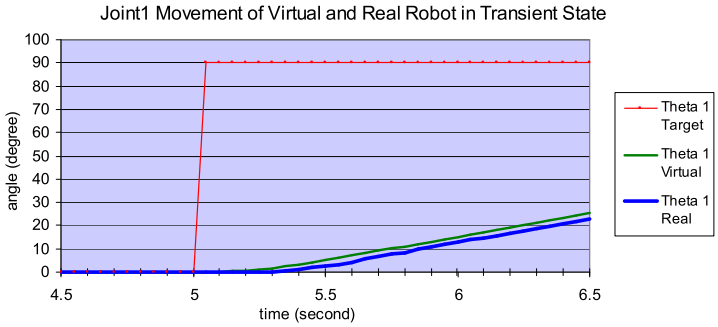

Fig. 6 Joint 1 in transient state.

Table II Simulation Tests Results.

Set	Joint No.	Movement Description (°)	Error Mean (transient state)	Error Mean (steady state)
I	1	from 0 to 90	3,53°	1,72°
II	2	from 0 to 90	4,32°	0,17°
III	3	from 0 to 90	4,94°	0,40°
IV	1 2	from 0 to 90 from 0 to 90	3,81° 5,16°	1,63° 0,23°
V	1 2	from 0 to 90 from 0 to 90	3,90° 1,80°	5,21° 0,47°
VI	1 2	from 0 to 90 from 0 to 90	4,33° 0,32°	5,33° 0,75°
VII	1 2 3	from 0 to 90 from 0 to 90 from 0 to 90	3,66° 4,81° 5,54°	1,89° 0,28° 0,65°

The summary of the simulation results data shown in Table II. Error in the table is referring to difference between virtual robot's theta and real robot's theta at same sampling time. The data revealed that, for the real robot, three joints movement simultaneously would make the movement of each joint is different from the movement when a joints is moving alone. The cause of this phenomenon

is that the movement of one joint would become interference to the movement of other joints.

In virtual robot, the movement of two or more joints simultaneously did not affect the movement of each joint. This is possible because the physics engine make simplification in the calculation of physics simulation. One simplification that has big effect is friction modelling in the physics engine ODE. To get better simulation results, then the better implementation of friction model must be done.

However, in general, for the purpose of learning, which is one of SIRUPP function, the virtual robot simulation result is able to show how the real robot moves when a particular control method is applied.

D. CONCLUSIONS

1. A manipulator robot simulator module has been built based on physics engine. This modules together with others modules form a complete application SIRUPP. The module capable of modelling kinematics and dynamics of manipulator robot, translating those parameters into a virtual robot, controlling the robot with a particular control method, and storing simulation data files for further analysis.
2. In this simulator, a real robot, PUMA 260, has been modelled and translated into an equivalent virtual robot. The trapezoidal velocity profiling control method with PID control filter used on real robots also has been modelled to a virtual controller which is used to control the virtual robot.
3. Initial test results showed that the simulator module based on physics engine capable of simulating the manipulator robot kinematics and dynamics, with a few limitations. The cause of differences between the virtual robot and the real robot responses is dynamics parameters that have been implemented in this experiment are limited, only link's mass and joints friction that have simple modelling in ODE.
4. While the physics engine ODE still have limitations to simulate manipulator robot accurately, but the ease of use and easy integration with the visualization library, like OpenSceneGraph, make the using of ODE physics engine, or

others physics engine, are feasible in robotics simulation used in robotics learning.

E. REFERENCES

[1] Ahmad Riyad Firdaus (2008), *Pengendali Modus Luncur (PML) Pada Robot Manipulator Dengan Optimisasi Algoritma Genetika*, Magister Thesis, ITB.

[2] Braunl, T., Pollak, R., Schutzner, J. (2006), *RoboSim – A Simple 6 – DOF Robot Manipuator Simulation System*, http://www.ee.uwa.edu.au/~braunl/robosim/, May 5, 2008, 10.00 WIB.

[3] Flückiger, L., Neukom, C., Pisanich, G., Buchanan, E., Wagner, M., and Plice, L. (2005), *Experiments with Autonomous Software for Planetary Robots: A Simulation Success Story*, http://robotics.estec.esa.int/i-SAIRAS/isairas2005/session_08a/ 4_flueckig_8a.pdf, Jan 8, 2009, 13:05 WIB.

[4] Pepper, C., Balakirsky, S., and Scrapper, C. (2007), *Robot Simulation Physics Validation*, http://www.isd.mel.nist.gov/PerMIS_2007/ proceedings/Papers/PerMIS07.Final_Pepper.pdf, Dec 30, 2008, 22:49 WIB.

[5] Rachmat Hariyanto (1999), *Perangkat Lunak Visualisasi Tiga Dimensi Kinematika Lengan Robot*, Magister Thesis, Institut Teknologi Bandung.

[6] Smith, Russel (2006), *Open Dynamics Engine User Guide*, http://www.ode.org/ode-latest-userguide.pdf , June 4, 2008, 14:21 WIB.

[7] Wahidin Wahab (1987), *Modelling dan Simulasi Suatu Sistem Lengan Robot*, http://digilib.itb.ac.id/, April 29, 2008, 11.00 WIB.

[8] _____ (___) , *Datasheet PIC-Servo SC V.10, Servo Motion Control IC*, Jeffrey Kerr LLC, http://www.jrkerr.com/picsrvsc.pdf, Feb 12, 2009, 15:42 WIB.

[9] _____ (1984), *Unimate PUMA Series 200 Specification Sheets*, http://www.antenen.com/htdocs/downloads/files/files_dl/puma260.pdf, Feb. 8, 2009, 11:13 WIB.