

# Real-Time Physics Simulation Packages: An Evaluation Study

J.Zouhair, and D.Ellison

**Abstract**—This paper includes a review of three physics simulation packages that can be used to provide researchers with a virtual ground for modeling, implementing and simulating complex models, as well as testing their control methods with less cost and time of development. The inverted pendulum model was used as a test bed for comparing ODE, DANCE and Webots, while Linear State Feedback was used to control its behavior. The packages were compared with respect to model creation, solving systems of differential equation, data storage, setting system variables, control the experiment and ease of use. The purpose of this paper is to give an overview about our experience with these environments and to demonstrate some of the benefits and drawbacks involved in practice for each package.

**Keywords**—DANCE, Inverted Pendulum, ODE, Simulation Packages, Webots.

## I. INTRODUCTION

**S**IMULATION is considered one of the important tools in different areas of research and development especially in the field of robotics because it can provide researchers with synthetic environments where they can investigate, design, visualize different types of objects, such as characters in a 3D games environment, and test their control methods in less time and at less cost.

Beside been cheaper and faster, Simulation is usually more flexible and easier to use than a real robot. It provides users with practical feedback when designing real world systems, which helps them to determine the correctness and efficiency of a system yet to be built. Nevertheless, simulation has its own drawbacks; virtual worlds are fundamentally hard to model, and simulating a robot also tends to be difficult because sensors in the real world can often exhibit different or unexpected characteristics [1].

Currently, they are many fields in which rigid body simulations are required such as in engineering applications, robotics and computer games. Due to this growing demand, various simulation software packages have been developed for the modeling and simulation of different kind of systems and intensively for the simulation of robotics systems, some of

them are open source others are commercial. Consequently, it becomes really hard for the users to decide on the suitable tool for their applications.

For this reason three simulation software packages have been evaluated, namely Open Dynamics Engine (ODE) [2], Dynamic Animation and Control Environment (DANCE) [3] and Webots [4]. ODE is the library that is used by DANCE and Webots but it is hidden to the user. All these packages allow us to create rigid bodies with physics properties such as mass, linear and angular velocity, joints, and a collision detection function that is integrated with these packages to identify bodies that are in contact.

First, a brief description of the inverted pendulum, which is used as a test bed, is discussed in section II, followed by the structure and features of ODE as well as the implementations of the inverted pendulum system in section III. We present after in section IV the DANCE package along with simulation results of the system, Webots structure and the implementation of the inverted pendulum in this environment are described in section V. We compare these simulation packages in more details in section VI. Finally, we give a conclusion in section VII.

## II. INVERTED PENDULUM

### A. Problem Definition

The inverted pendulum (IP) system consists of a cart on top of which a pole is pivoted. The cart is constrained to move only in the horizontal direction, while the pole can only rotate to its right, or left, depending on the controller's output. The goal of control is to balance the pole by supplying an appropriate force to the cart. The cart and pole friction coefficients are neglected in this study based on the work by [5], which indicates that they are too small to have interesting effects or are cosmetic.

One important issue to emphasize is that the cart wheels dynamics is ignored, in other words, their mass and moment of inertia are considered null.

### B. System Parameters

The following tables (Table1, Table2) describe the variables of the system, including both the inputs and outputs for a controller, and variables that govern the simulation of the system.

J.Zouhair is with the University of Abertay Dundee, School of Computing & Engineering, Kydd Building, Bell Street, Dundee DD1 1HG UK (e-mail: j.zouhair@abertay.ac.uk).

D. Ellison, was with the University of Abertay Dundee, School of Computing & Engineering, Kydd Building, Bell Street, Dundee DD1 1HG UK. He is now with (e-mail: dsellison@tiscali.co.uk).

TABLE I  
SYSTEM VARIABLES

Symbol	Name	Unit
$\theta$	the angle the pole makes with the vertical	radians
$\dot{\theta}$	pole angular velocity	radians/s
$X$	cart position	m
$\dot{X}$	the current velocity of the cart	m/sec
$F$	force applied to cart	N

TABLE II  
SYSTEM CONSTANTS

Symbol	Name	Unit
$M$	Cart mass	1Kg
$m$	Pole mass	0.1 kg
$l$	Pole length	1 m
$g$	Gravity acceleration	9.8m/s <sup>2</sup>
$t$	Time step	0.02 sec

### III. ODE

#### A. ODE Environments

ODE has been used in both research and commercial applications such as computer games. It is platform independent with a C/C++ API, written by Russell Smith [2]. It is an open source physics engine for simulating the physics of real-world objects. ODE is integrated in a number of commercial games and virtual reality simulations because it has the GNU Lesser General Public License (LGPL), which makes its source freely to be used by those commercial products.

They are many features supported by this library. ODE has a rigid body which is a structure that consists of many shapes connected together by different types of joints, like the ball and socket joint, the hinge joint (which was used for implementing our inverted pendulum model to allow a rotation around one degree of freedom), and the slider joint. It also supports integrated collision detection for real-time simulation, with friction.

#### B. Application of SFC to the IP in ODE

Building the model of the system was straightforward. First the bodies were created, and then connected with proper joints as shown in Fig. 1.

The State Feedback controller (SFC) was incorporated in equation (1); it takes the measurable state variables to calculate a force to balance the pendulum. The inverted pendulum was controlled using a controlling force that is based on the work of [6].

$$F = f_1 X + f_2 \dot{X} + f_3 \theta + f_4 \dot{\theta} \quad (1)$$

where  $f_1$ ,  $f_2$ ,  $f_3$  and  $f_4$  are coefficients that depend on the masses of the cart and pole as well as the friction of the system.

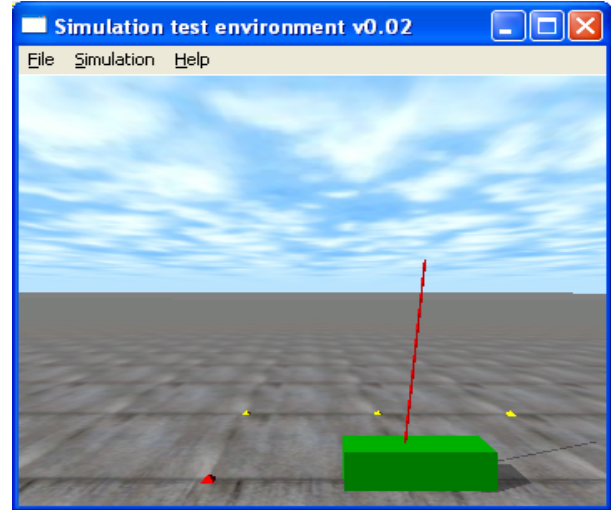


Fig. 1 The inverted pendulum in ODE

ODE has several functions for returning state variable values and which function to use depends on the type of joint, therefore; the values of state variables to calculate a force to balance the pendulum above were obtained from ODE by simply supplying details of the joints, for example to get the current pole angle and its rate of change  $dJointGetHingeAngle()$  and  $dJointGetHingeAngleRate()$  could be used, and to get the cart position, its velocity  $dBodyGetPosition()$  and  $dBodyGetLinearVel()$  may well be used for such a task.

```
const dReal poleAngle = dJointGetHingeAngle(joint1);
const dReal poleAngleRate = dJointGetHingeAngleRate(joint1);
const dReal *cartPos = dBodyGetPosition(body[0]);
const dReal *cartVel = dBodyGetLinearVel(body[0]);
```

So the controlling force that was applied into the cart was the following:

$$F = (f_1 * cartPos[0] + f_2 * cartVel[0] + f_3 * poleAngle + f_4 * poleAngleRate);$$

The result was a robust solution to the inverted pendulum problem; Fig. 2 shows the simulation plots for  $\theta$ ,  $X$  and  $F$ . At first the pendulum was in 0.1 radian position. The SFC balances the pendulum in the upright position at about 3 sec. The cart returned to its initial position at about 6 sec and no forces were applied to the system after 6 sec since it was balanced.

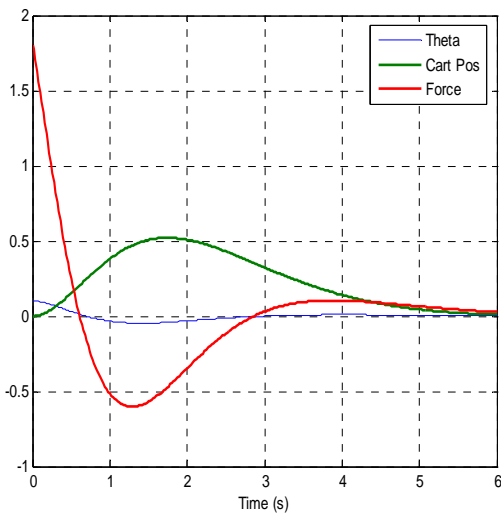


Fig. 2 Time Responses of the IP with SFC in ODE

IV. DANCE

A. DANCE Environment

DANCE is an open framework for computer animation that is based on an object oriented, plug-in architecture. It provides a rapid prototyping environment for implementing animation and control techniques with minimal design and implementation overhead [7].

DANCE supports four base classes [8], which are Systems, Simulators, Actuators and Geometries that are loadable as plug-ins in accordance with its APIs. Actuators and simulators are implemented as DANCE plug-ins to allow the user to dynamically load objects of any type, controllers and simulators at runtime. Controllers are implemented as plugin subclasses of the Actuator class.

B. Application of SFC to the IP in DANCE

Creating the inverted pendulum model in DANCE was achieved more easily than using ODE with the help of a graphical user interface (GUI) as illustrated in Fig. 3, which allowed us to manipulate the system, create objects, and add collision detection and gravity and so forth. Once the implementation of the inverted pendulum was completed, DANCE allowed us to save our environment in a single python file with a .py/ .dpy extension and all accompanying information gets saved automatically to a subdirectory that defines the linkage and joint structure, the system description file for cart and pole objects, the configuration file for cart and pole object. The .dpy/.py file contains all python scripting commands necessary to reconstruct the DANCE environment to its current state.

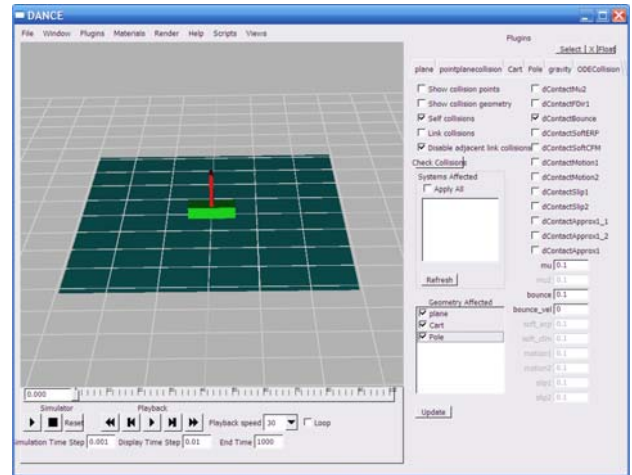


Fig. 3 Graphical output of the IP system in DANCE

The structure of the State Feedback controller was the same described in section III (B). Fig. 4 illustrates the result of the simulation in DANCE environment, which are similar to the results achieved in Fig. 2.

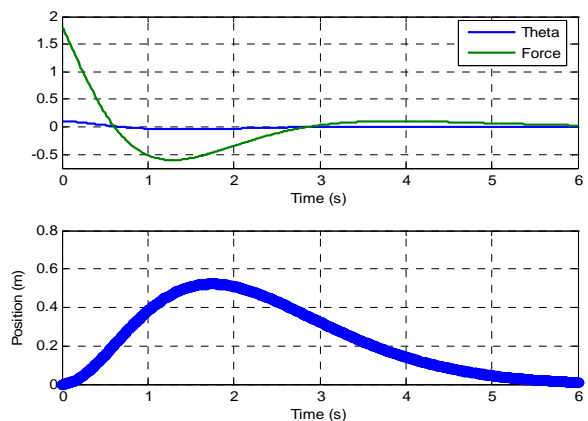


Fig. 4 Time Responses of the IP with SFC in DANCE

V. WEBOTS

A. Webots Environment

Webots is a commercial three-dimensional mobile robotic simulation software developed by Cyberbotics [9] providing the users with rapid technique of modeling complex virtual worlds, programming and simulating mobile robots within these environments. These robots can be equipped with a number of sensor and actuator devices, such as distance sensors, drive wheels, cameras, servos, etc, and can have different locomotion schemes for instance wheeled robots, legged robots, or flying robots. Added to this, Webots has many interfaces to real mobile robots, so that once the simulated robot behaves as expected, its control program can be transferred to many commercially available real mobile robots. Webots also relies on ODE to perform accurate dynamic physics simulation.

*B. Application of SFC to the IP in Webots*

Before modeling the cart and pole, a project directory was required to be created with two subdirectories *World* and *Controllers*, which were necessary to create a simulation in Webots.

- a. *The world file* defines the virtual environment such as shape, position and orientation, and physical properties of every object.
- b. *The controller file* contains the program for controlling the simulated cart and pole model and defines its behavior.

Building the inverted pendulum model was attained by simply adding the required elements from Webots'GUI as shown in Fig. 5:

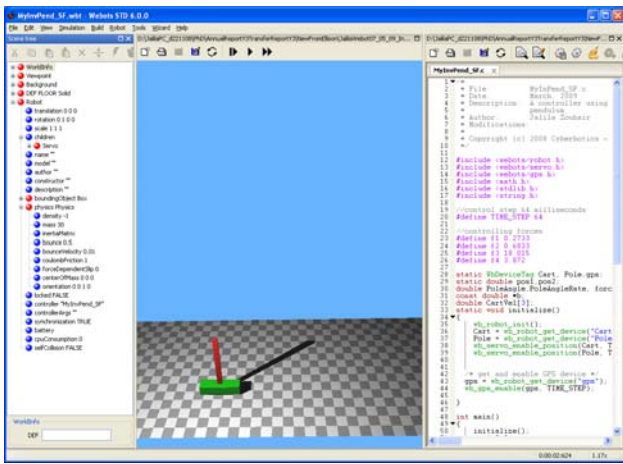


Fig. 5 Graphical output of the IP system in Webots

Once the cart and pole model were created, the controller program was coded to exhibit the desired behavior, which is to keep the pole balanced on the cart. Similarly, as it was discussed in section III (B), calculating a controlling force to balance the pendulum was required. Therefore, the values of state variables to calculate the controlling force were needed and obtained from Webots as follow:

The *wb\_servo\_enable\_position()* was used first to activates position measurements for the servo and *wb\_servo\_get\_position()* functions was used to get the value of the pole angle as well as the pole angle rate (deduced from position values returned by this function), *wb\_gps\_get\_values()* was used to get the cart position, and the linear velocity by differentiating the vector returned by this function.

The results of the simulation show that the cart maintains a good balance throughout the simulation period and this is noticeable in Fig. 6 as there is only a little variation between the forces applied to the cart in order to balance the pendulum. These results are in line with the results obtained using DANCE (see Fig.4) and ODE (see Fig.2).

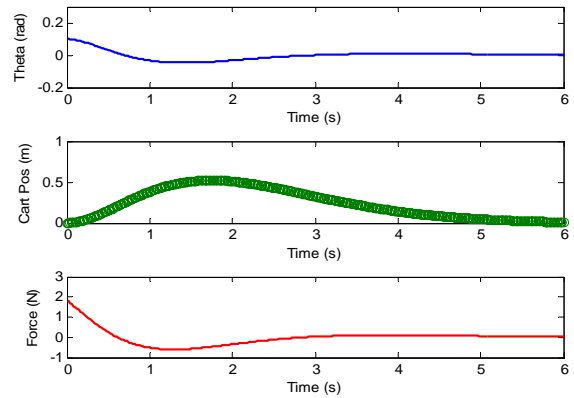


Fig. 6 Time Responses of the IP with SFC in Webots

VI. COMPARISON

*A. Model Creation*

For creating the inverted pendulum model using ODE, we had to code it completely by hand, as well as adding a simulation. This was an effortless task, but if we had to develop a larger model such as a humanoid it will be very time consuming. On the other hand, using DANCE and Webots to perform the same task was much simpler and easier with the help of the GUI; the user can simply add simulation and objects with joints and bodies one by one with the GUI and directly see the result on the screen. That makes the conception of the model smoother and faster.

*B. Solving Systems of Differential Equations*

ODE provides the dynamics and a collision analysis library and it was used in our project as a physics library, it also has its own differential equations solver. DANCE and Webots rely on ODE to perform accurate dynamic physics simulation but it is completely hidden to the users.

Several state variable values such as current pole angle and its rate of change, used for our inverted pendulum were effortlessly obtained from those packages by simply supplying details of the joints. As a result, it was straightforward to solve the controlling force applied to our cart.

*C. Data Storage*

DANCE stores some of the data of the model and its environment in the run directory. Once the session is saved, we can re-run the script or re-load the session at later stage directly from DANCE environment without having to rebuild the solution, which is not possible in ODE because any editing to the model or its behavior needs to be hard coded directly in the C++ program. As a result we have to build the entire solution every time we want to view any modification to it. On the other hand, Webots stores all the files related to our project in separate files, i.e. world files, controller files, data files, plugins, etc. This allows the models created to be easily modified and reused.

#### D. Setting System Variables

The GUI interface of DANCE helps us to interactively manipulate various parameters of the system, i.e. set its position, set its velocities on the joints, etc., while Webots' GUI has a Scene Tree Window which is a hierarchical representation (see Fig. 5) of the current world that helps us to define or modify certain properties and system variables and directly see the result in the simulation window, i.e. translate or rotate any object in the scene, changing their physics parameters such as mass, joints type, friction, density, etc. In ODE, all the variables need to be set in the main program before running the model, which can be time consuming if we want simply to rotate the pole angle for example.

#### E. Experiment Control

In order to control our inverted pendulum, both DANCE and Webots have a built in simulator GUI control available in their environments which provide us with a good control for simulating our model such as lowering or increasing the simulation, measure how often the screen is refreshed. However, this simulator control is not available in ODE.

#### F. Ease of Use

Although the ODE library is easy to use as mentioned in this paper, it takes a while to learn to use it well plus the example programs that come with it were not clear. On the other hand, DANCE uses a number of third party libraries, such as Python, ImageMagick and OpenGL, which makes it not user friendly. Added to this, a new OO scripting language (Python) needs to be learnt.

As for Webots, its GUI makes it very simple environment because many of its facilities are integrated in the Webots' user interface, such as a source code editor and motion editor. Moreover, it allows us to interact with our simulated IP during its run time using its GUI, which contains step by step mode, rotating objects in the scene or changing view point positions.

#### G. User Guide

The documentation for ODE is more of a reference manual because it has a good description of its features but it does not include any tutorials or examples to assist the average users.

Despite the fact that DANCE has a number of example scripts that demonstrate its functionality and are easily accessed through its GUI, yet still a lot of time was spent in understanding its internals and taking advantages of its capabilities. Conversely, Webots provides a comprehensive documentation and tutorials to get us up and running quickly in both the modeling and writing up of the controller. It has also a separate reference manual that contains all the information on the world description language used in Webots, and how to program robot controllers.

## VII. CONCLUSION

In this study, three physics simulation tools ODE, DANCE and Webots were evaluated using SFC and the Inverted Pendulum as a test bed. Based on our comparative study, we

concluded that Webots has many advantages over other simulation packages discussed in this paper and could be summarized as follow:

- Good separation between the model and the controller, which does not exist in other packages.
- Once installed, webots allows us to instantly create an environment, while in both ODE and DANCE the setting up of project environment takes time just to get started and to build the main project solution.
- Webots provides us with the facilities to import external models from a 3D modeling software such as Maya but other packages lack this feature.
- Ease of system implementation was best in Webots, whereas DANCE offered better environment for modeling our system compared to ODE.
- The portability of the control system, we can develop our control system and validate it on the simulator, and then test our control system in a real mobile robot such as Khepera and the Lego Mindstorms[10]. However; this feature is not available in the other two packages.
- Webots facilitates creation of AVI or MPEG simulation movies and take PNG screen shots. Dance allows us also to render its objects to a file in one of the standard image formats such as png and gif files, but it does not automatically produce animations from these images. A third-party program (such as Quicktime) is needed to assemble the animations into a working video.

Although Webots is a commercial software compared to other packages, we concluded from these results that it provides the best suitable environment for testing ideas in intelligent robotics algorithms.

## REFERENCES

- [1] T. Jones, "Open source robotics toolkits," 05-Sep-2006. [Online]. Available: <https://www.ibm.com/developerworks/linux/library/l-robotools/>. [Accessed: 05-Aug-2010].
- [2] R. Smith, "Open Dynamics Engine - home," 2007. [Online]. Available: <http://ode.org/>. [Accessed: 19-Jan-2009].
- [3] A. Shapiro, "Dynamic Animation and Control and Environment," 2009. [Online]. Available: <http://www.arishapiro.com/dance/>. [Accessed: 19-Apr-2009].
- [4] O. Michel, "Cyberbotics," *Webots 6 fast prototyping and simulation of mobile robots*, 2009. [Online]. Available: <http://www.cyberbotics.com/>. [Accessed: 19-Jun-2009].
- [5] S. Geva and J. Sitte, "A cartpole experiment benchmark for trainable controllers," *Control Systems Magazine, IEEE*, vol. 13, no. 5, pp. 40-51, 1993.
- [6] J. White, "System Dynamics Introduction to the Design and Simulation of Controlled Systems Introduction," 1997. [Online]. Available: <http://gershwin.ens.fr/vdaniel/Doc-Locale/Cours-Mirrored/Methodes-Maths/white/sdyn/s7/s7intro/s7intro.html>. [Accessed: 10-Jul-2009].
- [7] A. Shapiro, P. Faloutsos, and V. Ng-Thow-Hing, "Dynamic animation and control environment," in *Proceedings of Graphics Interface 2005*, p. 70, 2005.
- [8] P. Faloutsos, M. V. D. Panne, and D. Terzopoulos, "Composable Controllers for Physics-Based Character Animation," *Proceedings of ACM SIGGRAPH 2001*, pp. 251-260, 2001.
- [9] O. Michel, "Cyberbotics Ltd. Webots TM: Professional Mobile Robot Simulation," *International Journal of Advanced Robotic Systems*, vol.

- 1, no. 1, pp. 39–42, 2004.
- [10] MINDSTORMS, “LEGO.com MINDSTORMS NXT Home,” 1999.  
[Online]. Available:  
[http://mindstorms.lego.com/eng/Egypt\\_dest/Default.aspx](http://mindstorms.lego.com/eng/Egypt_dest/Default.aspx). [Accessed:  
10-Jul-2009].