Abstract

This paper describes two basic control problems related to locomotive control that were tested in Open Dynamics Engine (ODE), a real-time physics SDK. An ankle controller and a free fall leg orientation controller are developed and tested in ODE. The control theory for both controllers was discussed, and the results from the ODE simulations were presented. It is shown that though ODE is a convenient simulation platform, it has inherent inaccuracy and instability issues that make control problems deviate from theory. Alternative approaches are also discussed.

Introduction

Fields such as biomechanics, robotics, and ergonomics, have produced a rich description of locomotion of both biological and robotic systems. These studies are normally concerned with "normal" gaits on flat ground. A common route to provide realistic locomotive animation is to use data from motion capture techniques. There still, however, exists the problem of obstacle navigation. Motion captured datasets would have to be enormous to cater for all possible varieties of obstacles. Synthesis of locomotion for use in computer animation is an alternative, but is a challenging exercise, and of the various techniques the most convincing obstacle navigating techniques require masses of computation. To provide real-time control of locomotion currently requires techniques that simplify the locomotive model as much as possible and often require "cheats" to achieve realistic looking locomotion.

From here on, let's talk of locomotive systems as characters. Characters could be any mono or multi-pedal virtual character.

The overall goals are:

- **Interactivity**. The control of the character must be responsive to the user's input and require a minimum amount of control parameters.
- **Environmental reaction**: The character must respond to external influences, and obstacles. For example, it must respond when a brick hits in a physically correct way, and climb over, crawl under or bump into any obstacle depending on the nature of the obstacle.
- **Believability**: The character's locomotion must look believable and realistic.

Physics software development kits (SDK) tend to simplify and speed up the task of simulating and analyzing real world systems. In the fields of engineering, physics simulation packages enable an engineer to test designs without the need for prototyping. Recently a market for real-time physics SDKs has arisen. This is due to computer power and speed increases in accordance to Moore's Law and a user demand for more realistic interaction with virtual environments.

This paper serves as an introduction to locomotive control using a specific physics SDK, Open Dynamics Engine (ODE), and to evaluate ODE as a real-time control simulation platform. As such, complete control solutions will not be presented here. Two example controllers – a simple ankle controller and a mid-air falling body controller will be examined in theory and simulated in ODE.

Theory

The Ankle Controller

The first model that will be examined is the stationary inverted pendulum controlled by an actuator. This model can be viewed as a simplified ankle. The 2D model is presented in *Fig1* and *Table1* shows the symbol definitions.

Table 1

m	The mass of the bob
l	The distance of the bob from pivot
g	Gravity
u	Input Torque
?	Angular displacement

Looking at *Fig1*, we can imagine *u* to be a muscular torque produced by the ankles on some centre of mass, *m*. In a real world example, *l*, the distance to this centre of mass would be dynamic, dependant on the posture a character would be in. In this simplified model, we assume that the feet are 'cemented' to the ground, incapable of slipping.

The equation of motion for the system, by summing the torques clockwise is:

$$t_{TOTAL} = ml^2\ddot{q} = u - mgl\sin(q)$$

A state variable representation of a system is written as the following differential equation [1]:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

where x is a vector of state variables, u a scalar/vector of inputs, and y a scalar/vector of outputs. A is called the system matrix, B the input vector/matrix, C the output vector and D the feedthrough coefficient/vector.

To develop a state variable representation of the system, we take *u* to be the input torque of the system and the output, *y*, to be *?*. The chosen state variables are $\dot{q}$ and *?*.

The set of state variable equations can then be written as:

$$\frac{dq}{dt} = \dot{q}$$
$$\frac{d\dot{q}}{dt} = \ddot{q} = -\frac{g\sin(q)}{l} + \frac{u}{ml^2}$$
$$y = q$$

Linearise these equations about $q \approx 0$, so we can approximate $\sin(q) = -q$. The normalized (which means setting all constants to 1) state space representation of the system can then be written in matrix notation as[2]:

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x$$

Where *x* is a vector $[q, \dot{q}]$, representing the system's state variables.

To represent the system as a discrete system, the system needs to be presented as difference equations, analogous to the differential equations of the continuous time system[1]:

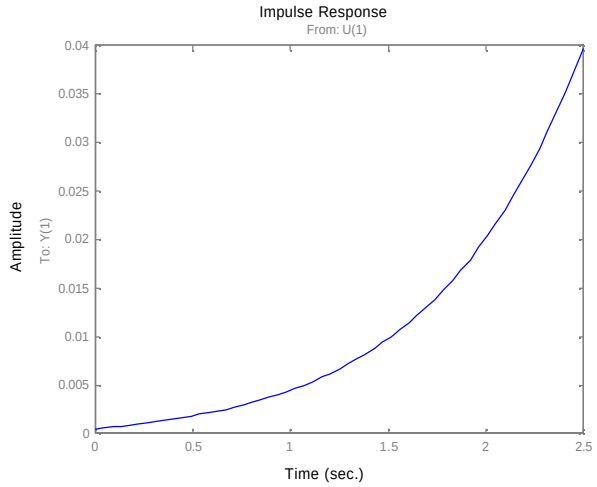
$$x[n+1] = Ax[n] + Bu[n]$$
$$y[n] = Cx[n] + Du[n]$$

Where *n* is the time represented as an integer.

Sampling the analogue system at 30 frames per second, using sample and hold method, gives the following discrete system:

$$\dot{x} = \begin{bmatrix} 1 & 0.033 \\ 0.033 & 1 \end{bmatrix} x + \begin{bmatrix} 0.005 \\ 0.033 \end{bmatrix} u$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x$$

The impulse response of the system is how the ouput changes over time if a unit input is produced for one time step at time 0. The impulse response of the open-loop system is shown in *Fig2*. The system is clearly unstable, with theta shooting off to infinity as time increases.



To stabilise the system, the torque needs to be controlled by some means. The state variable feedback technique adjusts the input of the system (the ankle torque in this case) by subtracting from the input an amount relative to the state variables, determined by a feedback gain matrix K (see *Fig3*).

The feedback gain K was chosen using the `dlqr` function in MATLAB. This function computes the optimal feedback gain matrix K, such that the feedback law [4]

$$u[n] = -Kx[n]$$

minimises the cost function

$$J(u) = \sum_{n=1}^{\infty} (x[n]^T Qx[n] + u[n]^T Ru[n])$$

where the simplest case is to assume $Q = C'C$ and $R = 1$[5].

By adding a state feedback controller with gain $K = \begin{bmatrix} 320 & 25 \end{bmatrix}$, shown in *Fig3*, gives the impulse response shown in *Fig4*. The real input to the system in this case is:
$u - K_1 * ? - K_2 * \dot{q}$.

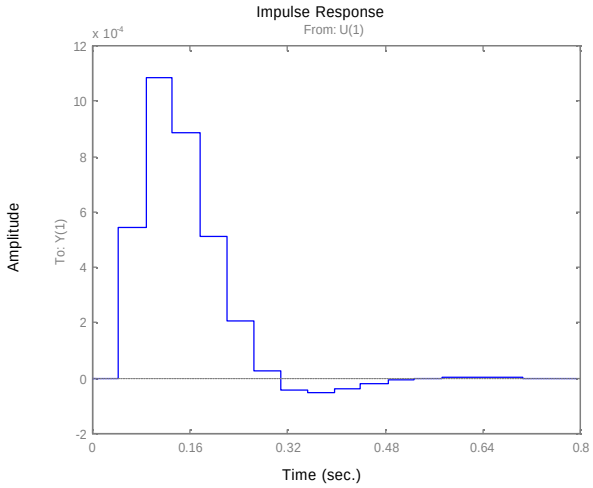


Fig3 shows that the normalized system can be made stable with the introduction of a state variable feedback controller. Since we should be able to determine the state variables directly in ODE, we don't bother with designing an observer for the system.

The falling body angular controller

This controller attempts to control a body in mid flight. It can be seen as a simplified leg in

mid stride attached to a simplified body. *Fig5* shows a schematic of such a system, and *Table2* shows the symbol descriptions.

Table 2

m_b	The mass of the flywheel
ml	The centre of mass of the leg
l	The distance of the leg centre of mass from flywheel pivot
t	Input Torque
?	Angular displacement
$\ddot{q}$	Angular acceleration
I_l	Moment of inertia of the leg

Because the system is in a state of free-fall, no gravity is attributed for. The equation of motion can be found by summing the torques of the system:

$$I_l \ddot{q} = t = u == -mr^2$$

where u is the input for the system. Again, taking *?* to be the output of the system, the system equation is:

$$\frac{dq}{dt} = \dot{q}$$
$$\frac{d\dot{q}}{dt} = \ddot{q} = -ml^2 u$$
$$y = q$$

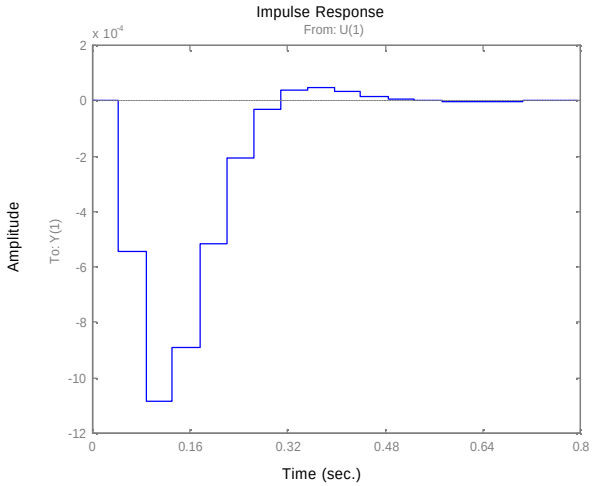
These equations describe a simple double integrator system, and the normalized state space representation can be written as[2]:

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ -1 \end{bmatrix} u$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x$$

The equivalent digital representation, sampled again at 30 frames per second, as follows [2]:

$$\dot{x} = \begin{bmatrix} 1 & 0.033 \\ 0 & 1 \end{bmatrix} x + \begin{bmatrix} 0.005 \\ -0.033 \end{bmatrix} u$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x$$

The impulse response is linear, and with the same state feedback as before, the closed loop impulse response is shown in *Fig5*.



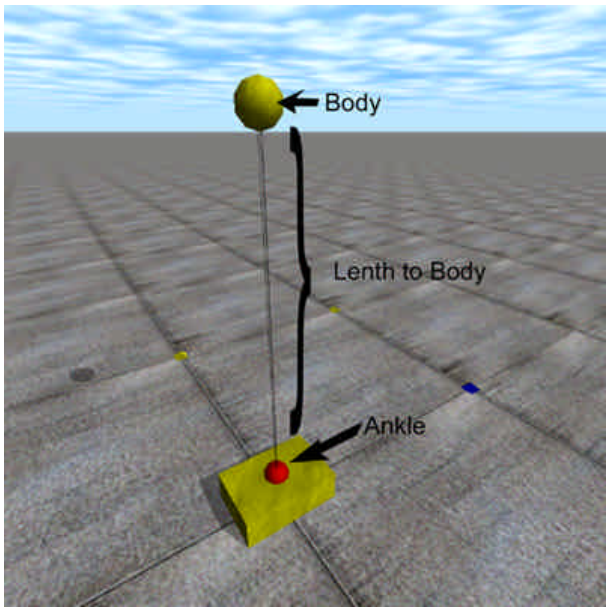
This is almost exactly like *Fig4*, only inverted, since the input torque is in the opposite direction to $\ddot{q}$.

The major difference between these two systems is that the angular controller is inherently linear. The ankle controller only approximates the system for small angle displacements from the vertical equilibrium position. Also, assuming we have a flat plane and if *?* > p or *?* < -p, then the ankle control system does not sufficiently describe a real system, since at these angles collision with the floor would have occurred.

Experiment

ODE is a C/C++ library that provides functions to create a virtual environment, and simulate the physics in that environment in real-time [3]. ODE allows a user to describe simple rigid body properties, like mass, position, speed, angular motion and the inertia tensor. Rigid bodies can then be connected with joints of various types. Some joints have motors, but only the velocity of the motor can be set. Geometries for the bodies are then specified for collision detection. Collision detection is handled by ODE, and collisions create temporary contact joints.

The Ankle Controller simulation



Sphere masses describe point masses most accurately. A sphere was created, attached to a ball joint close to the ground. The ball joint was anchored to the world, thus making it a static joint. Joints in ODE have no mass, therefore making it correlate with the theory. The joint and sphere were placed in a precarious equilibrium, with the sphere one meter above the ball joint. At the starting time of the simulation, a torque was introduced to destabilize the system.

The following code snippet shows how to do the above in ODE and C++:

```
// create a body
ANKLE = dBodyCreate (world);
// setup the body positions
dBodySetPosition (ANKLE,0.0,0.0,1.1);
// set the body mass distributions
dMassSetSphereTotal   (&m,M2,0.05);
dBodySetMass (ANKLE,&m);
// create the collision geometries
ankle = dCreateSphere (space,0.05);
dGeomSetBody (ankle,ANKLE);
// create the joint
anklejoint = dJointCreateBall ( world,
0);
// attaching a joint to 0, is attaching
// it to a static environment
dJointAttach (anklejoint, ANKLE, 0);
// create the impulse torque
dBodyAddTorque (ANKLE,1,0,0);
```

The angular displacement, *?*, and velocity, $\dot{q}$, were calculated for each time step and recorded to a file. A state space controller was introduced by simply adding a torque during each time step equal to 320* *?* + 25*$\dot{q}$. The results were then recorded to file.

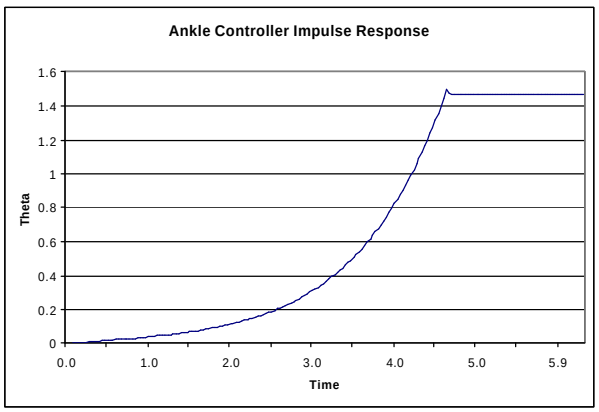
The falling body angular controller simulation

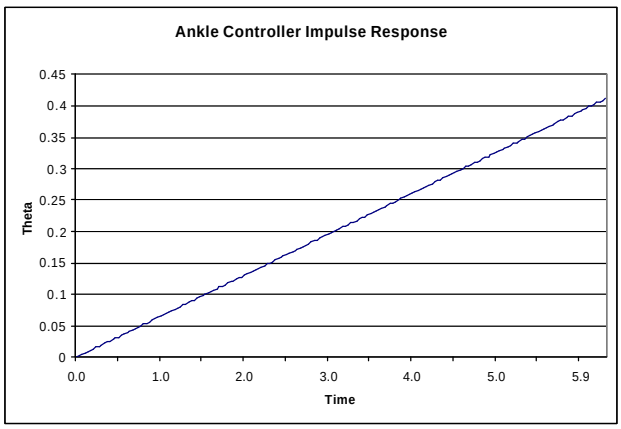
This was simulated in a similar fashion to the ankle simulation, but with no gravity to simulate free fall. The joint was kept fixed since the angle is measured relative to the main body. The main body does not feature in this control problem.

Results

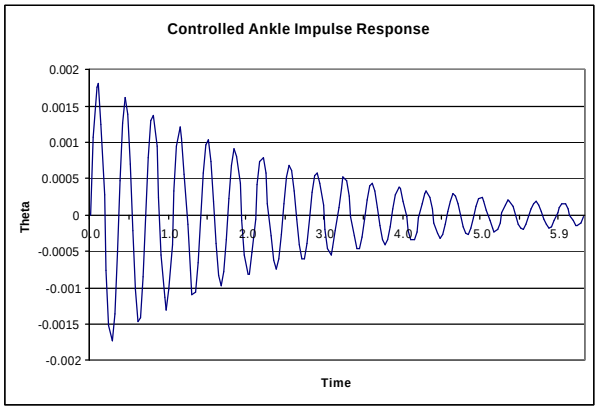
The Ankle Controller simulation

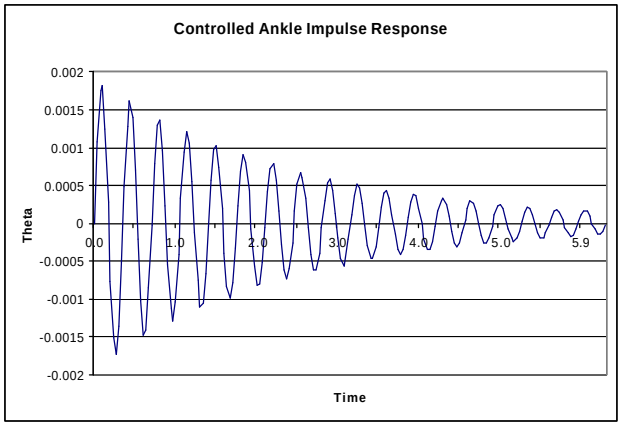
A normalized open loop system as described in the theory section was set up and found to have an impulse response shown in *Fig7*. The flat region represents the sphere hitting the ground. Notice how the sphere penetrates the ground momentarily. This is to ODE's ERP and CMF parameters, discussed later.





When the controller described in the theory section was added, the impulse response shown in *Fig8* was recorded.





Clearly, this deviates from the expected result just by comparing with *Fig4*.

The falling body angular controller simulation

The impulse response of the open loop system without feedback is linear, as expected from theory, and is shown in *Fig9*.

The impulse response of the closed loop controlled system again is unlike the response expected from theory. Because there is no added force from gravity, the oscillations die down more quickly than the ankle example. The controlled impulse response is shown in *Fig10*. Note how similar *Fig10* is to *Fig8*.

Why the difference between theory and experiment?

The control theory, discussed in the theory section, caters for linear systems, and through linearising, much of the behaviour of a real system is lost. This, however, does not account for all the discrepancies between theory and results. Digital control theory was developed to control real systems with digital equipment, not so much digital systems with digital equipment. ODE is not meant to be a scientific analysis tool. ODE has inherent inaccuracies due to the fact that it only approximates real physics. ODE uses a Lagrange multiplier integrator, which compromises accuracy for speed[6]. By decreasing the time step between integrations, accuracy and stability can be increased. Physics in ODE also has an inherent "sponginess" due to the error reduction

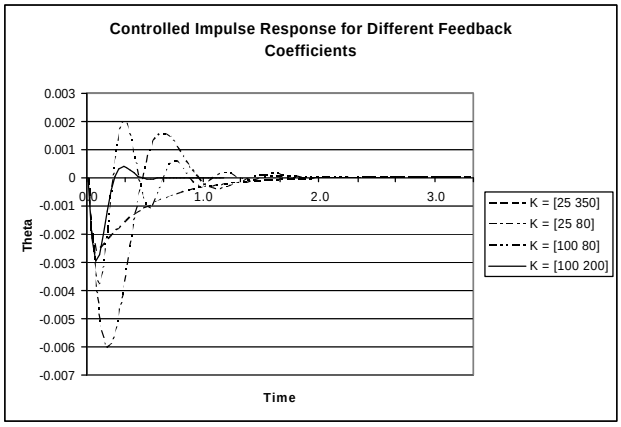
parameter ERP and constraint force mixing CFM parameters. The effect of these parameters can be seen in *Fig7* where the sphere briefly penetrates the ground before settling. This "sponginess" creates a control problem to designers because all systems have inherent natural frequencies different from their intended ones.

The relationship between these two parameters and the 'springyness' of the system, can be related to a spring constant, *k*, and a dampening constant, *d*, with the following two equations:

$$ERP = \frac{hk}{hk + d}$$

$$CMF = \frac{1}{hk + d}$$

Through trail and error, different values for the feedback gain matrix, *K*, were tested for the ankle experiment. The results of these are plotted in *Fig11*.



It appears that to get good results for the control problem, the feedback gain matrix deviates from theory substantially.

ODE does however provide a control mechanism of its own. Some joints have 'motors' and some can have angular motors attached to them by the user. The user can only adjust the velocity of the motor by specifying a desired velocity. The motor attempts to achieve this velocity dependent on bodies attached to it, collisions, and the maximum torque that can be produced by the motor (also specified by the user). A naïve servo control method to move an angular motor to some desired angle, q_d, would be to set the desired angular velocity, $\dot{q}$, at each time step to be:

$$\dot{q} = s(q_d - q_c)$$

where q_c is the current angle of the motor and *c* is a scaling factor.

Future work

The added dynamics inherent in ODE need to be compensated for. To realise the goal of creating a real-time controllable character locomotion control system, basic control design principles need to be developed specifically for non-real virtual systems. Other methods might include ways and means of "forcing" a virtual system to adhere to expected control behaviour, while normal system behaviour is still handled by ODE in its usual way. Eventually a generalised leg model must be developed that can operate in a virtual environment with obstacles by itself, or connected to other legs.

Conclusion

ODE was not meant to be used for quantitative engineering. It does, however, offer an easy-to-use development platform for real-time physics simulations that do not require too much accuracy. With a generalised control strategy, it would be possible to use it to simulate more complex controlled systems. A generalised control strategy should employ joint motors as much as possible to enhance stability and promote model consistency.

References

[1] Richard J. Vaccaro, *Digital Control, A State-Space Approach*, 1995, McGraw-Hill. Pages : 81,82

[2] Karl J. Åström, Björn Wittenmark, *Computer-Controlled Systems, Theory and Design*, 1997, Prentice Hall. Pages: 32 – 36, 528 – 531

[3] Russel Smith, *Open Dynamics Engine v0.039 User Guide*, 2003.

[4] *Control System Toolbox, for Use with MATLAB, User Guide Version 4.2*, 1999, The Math Works.

[5] *CTM Example: State-space design for the inverted pendulum*, Carnege Melon, http://www.engin.umich.edu/group/ctm/examples/pend/invSS.html

[6] *Open Dynamics Engine - ODE* , 2003, http://opende.sourceforge.net/ode.html - Home Page.