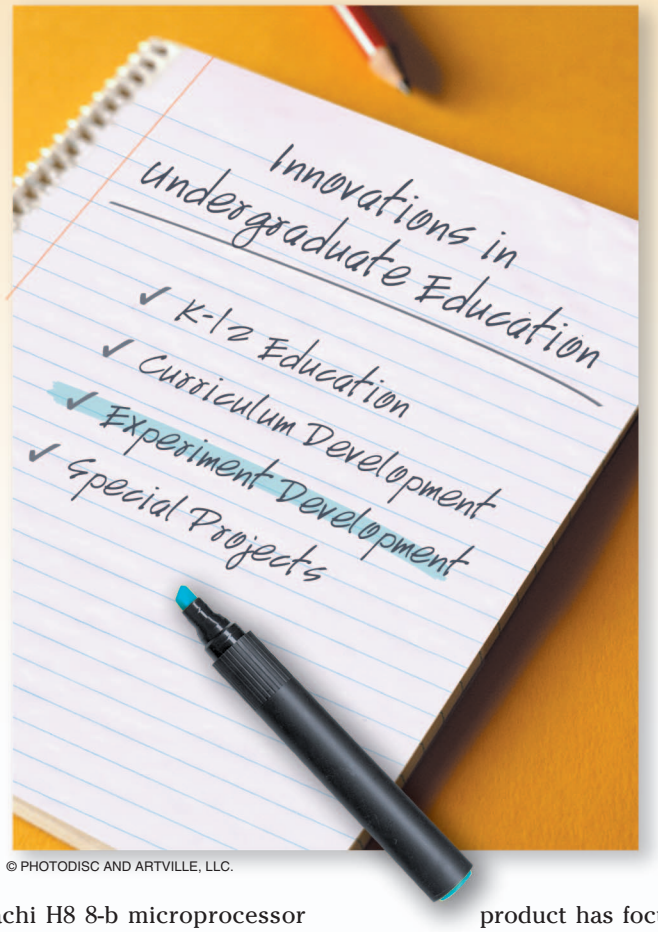
FEATURE

# A LEGO-Based Control Experiment

## Bench-top demonstrations and laboratory experiments in the field of continuous-time control systems






By Peter J. Gawthrop and Euan McGookin

The LEGO Mindstorms Robotic Invention System is marketed as an educational toy for children aged 12 years and older [1], [2]. Apart from a plethora of structural components, the kit contains dc motor actuators, a range of sensors, and, most importantly, the so-called RCX component. The RCX component is a self-contained programmable system based on the Hitachi H8 8-b microprocessor with built-in interface to three actuators and three sensors, as well as an infrared (IR) communications interface.

The RCX component was initially developed as an educational tool through the collaboration of LEGO and MIT [3], [4]. One of the first versions of the RCX allowed six input and six output blocks to be connected to the H8 microprocessor. When LEGO developed the commercial version of the RCX, the number of inputs and outputs was reduced to three of each. Although this change reduced the flexibility of the RCX utilization, it reduced the drain on the batteries that power the system.

The initial intended use of the RCX system was for research and educational activities. Combining the versatile LEGO Technics construction blocks with the easy-to-use programming and I/O interfacing of the RCX provided a fast prototyping system to support these activities. Although the commercialization of this product has focused on the recreational and K–12 educational markets, the flexible and expanding world of LEGO Mindstorms is widely accepted as a tool for research and higher education.

Enthusiasts have extended the hardware and software in various ways [1], [2]. A recent issue of *IEEE Robotics and Automation Magazine* [5]–[7] argues that these extensions show that the LEGO Mindstorms kit can be used to good effect in an educational context. In particular, the kit is relatively cheap, robust, reconfigurable, reprogrammable, and induces enthusiasm and innovation

0272-1708/04/$20.00©2004IEEE

**Figure 1.** *The cart and pendulum system. This view is generated using the LEGO design program Ldraw. The corresponding parts lists are given in Tables 1 and 2. The large yellow brick contains the microprocessor, and has three sensor inputs and three actuator outputs. The pendulum is suspended from a triangulated gantry attached to the cart. More detail appears in Figure 4.*

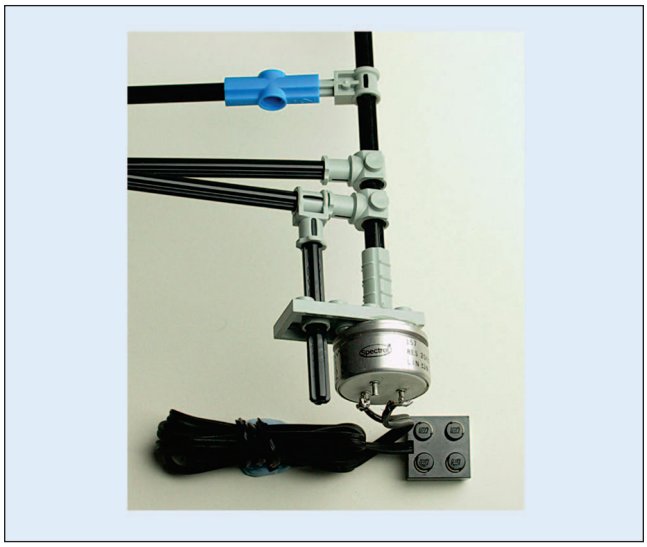


**Figure 2.** *Angle sensor detail. A* 10 kΩ *low-friction conductive-plastic potentiometer (Radiospares #173–580) is glued to a standard grey LEGO plate. The potentiometer shaft is firmly connected by means of a grey standard axle joiner to the black axle, which is firmly connected to the black pendulum by means of the grey* 90° *axle connecter and the blue connector; thus, the potentiometer shaft rotates with the pendulum axis. The potentiometer body is constrained by the short black axle protruding though the grey plate and firmly fixed to the black upright. One end of the conductive track and the wiper are soldered to a standard LEGO cable and connector; when connected to the RCX, this arrangement measures the angular rotation of the pendulum. The corresponding code appears in Code fragment 14.*

in students. However, most work in robotics and automation such as [5]–[8] focuses on *discrete-event* control using on-off sensors and logic control.

In contrast, this article uses the LEGO Mindstorms kit for bench-top demonstrations and laboratory experiments in the field of *continuous-time* control systems. However, neither the sensors nor the actuators provided as standard are appropriate for this purpose; in fact, the sensors are heavily quantized and the actuators are nonlinear. This article shows how these deficiencies can be overcome by creating more accurate sensors and by linearizing the actuators using electro-mechanical feedback. Several control experiments are presented to investigate and illustrate the use of this kit; in particular, we consider the identification and state-space control of the cart and pendulum system shown in Figure 1.

Throughout the text, ideas for student investigation are presented in boxed form like this.

## LEGO Components

The components of the LEGO Mindstorms Robotic Invention System are described in detail in [1]; for this article, components are categorized under the headings of sensors,

### Online Resources

- http://brickos.sourceforge.net/contains the latest version of the legOS/brickOS operating system for the LEGO RCX brick.
- http://www.ldraw.org/ has programs for drawing and displaying LEGO models.
- www.octave.org has information about the Matlab-like program Octave.
- http://www.mech.gla.ac.uk/~peterg/Lego contains
  - —the LEGO model in ldraw form
  - —the legOS and LNP source used for the project
  - —the cart and pendulum control code
  - —the Octave code for controller design and code generation
  - —a movie of a cart and pendulum experiment.

Because brickOS and LNP are open source, the student can gain an understanding of real-time operating systems and sensor/actuator interfacing. Students are encouraged to download and use the latest versions to understand the importance of software upgrades and the corresponding bug-fixing cycle.

actuators, and information technology. As discussed in [2], it is possible to build alternative sensors and interface them to the RCX. Equally important, an alternative operating system legOS (now known as brickOS [9]) is available for implementing real-time controllers programmed in C++, and a TCP/IP based communications protocol LNP (legOS network protocol) is available for downloading code and communicating data by means of the IR link.

## Sensors

For control-system operation, the standard LEGO sensor of interest is the rotation sensor; however, this sensor can measure rotation only in increments of $360^\circ/16 = 22.5^\circ$. As discussed in [1], this resolution can be increased by suitable gearing; unfortunately, this approach introduces excessive friction for the purpose of suspending a pendulum. However, as shown in [2], an instrumentation-quality potentiometer can be glued to a standard LEGO flange and electrically interfaced to the RCX. Figure 2 illustrates how this modification is accomplished.

There is no standard velocity sensor provided. However, as every engineering student should know, an open-circuit dc motor produces a voltage proportional to angular velocity. As discussed in [2], the RCX can measure a voltage as long as the voltage source has large internal resistance. Figure 3(a) gives details of how the measurement can be accomplished.

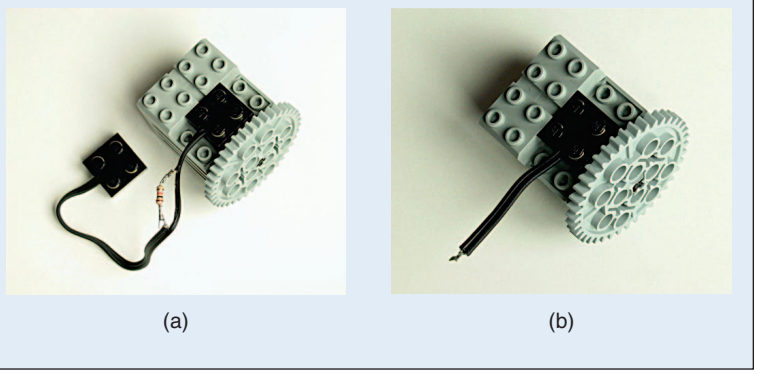



**Figure 3.** *Two uses for a dc motor: (a) velocity sensor and (b) brake. (a) One of the two wires in the standard LEGO connector is cut, and a 10 kΩ resistor is soldered between the two bare ends. Because the RCX draws little current, the measured voltage is effectively the armature voltage, which is proportional to angular velocity. The small armature current also means that the braking effect is small. The resistor is necessary to protect the RCX. The corresponding code appears in Code fragment 15. (b) Both wires of the standard LEGO connector are cut, and the two bare ends soldered together. This technique effectively short-circuits the motor terminals, and thus all generated power is dissipated in the armature resistance. This* electrodynamic brake *makes the cog wheel act as a linear mechanical resistance with torque proportional to angular velocity.*

As discussed later, the output from the angular velocity sensor is noisy. Students can design and build a suitable low-pass filter.

## Automatic Code Generation

Automatic code generation is widely used in industry and academia. As a simple example, consider the state-space controller of (7) with numerical values given in Table 3. Standard state-space design within the matrix manipulation package Octave (similar to MATLAB) is used to generate the state-feedback gain $K$ and observer gain $L$ given by

$$K = [1.2688 \quad 4.4721], \quad L^T = [1.6857 \; 10.1672].$$

The m-files controller2c.m of Code fragment 16 and matrix2c.m of Code fragment 17 generate the following code ready for compilation:

```
/* Error equations */
e = -y + x[1];
/* State estimator */
xdot = -2.30968*x[0] + 1.91551*u_out - 1.68575*e;
x[0] = x[0] + xdot*DT;
xdot = +x[0] - 10.1672*e;
x[1] = x[1] + xdot*DT;
/* Control signal */
u = 4.47214*w - 1.26875*x[0] - 4.47214*x[1];
```

Note that matrix entries with value zero do not appear, and matrix entries with value unity do not explicitly appear. Compared to the alternative approach of using loops to implement matrix multiplication, this approach yields fast-executing code, which is an important consideration when using the relatively slow Hitachi H8 processor. This code is included in the function Cart_con.c listed in Code fragment 18, where further explanation is given.

### *Actuators*

The standard LEGO actuator component is a good quality permanent-magnet dc motor with high inertia and low friction.

**Although the commercialization of this product has focused on the recreational and K–12 educational markets, the flexible and expanding world of LEGO Mindstorms is widely accepted as a tool for research and higher education.**

The high quality of the LEGO motors allows a good demonstration of electro-mechanical energy conversion by electrically connecting two motors and observing that manually rotating one shaft rotates the other shaft through the same angle.

Students can devise methods to measure the armature resistance $r_a$, the motor gain $k_m$, and the motor mechanical inertia $j_m$ assuming that both friction and armature inductance can be neglected.

Unfortunately for control purposes, the linear actuator is powered using pulse-width modulation. During the on phase, the armature resistance contributes to the system dynamics, whereas during the off phase the armature is an open circuit, and thus the armature resistance plays no part in the overall system dynamics. Thus, the effective armature resistance depends on the control signal to the dc motor. In the context of the cart and pendulum system, the armature resistance forms the major part of the resistive component of the dynamics, and so this nonlinear effect is significant.

To the control engineer, a natural way to overcome the nonlinearity arising from the pulse-width modulation is to design a velocity feedback loop. A simple design is to mechanically connect the actuating dc motor shaft to the shaft of a second dc motor with short-circuited armature. Figure 3(b) gives relevant details.

Students can show why this connection gives velocity feedback and why the short-circuited dc motor is equivalent to a rotational mechanical damper. They can also investigate the alternative approach of implementing the velocity feedback in software using the angular velocity sensor discussed above.

## Cart and Pendulum System

The cart and pendulum system shown in Figures 1 and 4 is constructed using the LEGO components of Tables 1 and 2, together with the special sensors discussed above and corresponding software. In particular, the hardware comprises the following items:

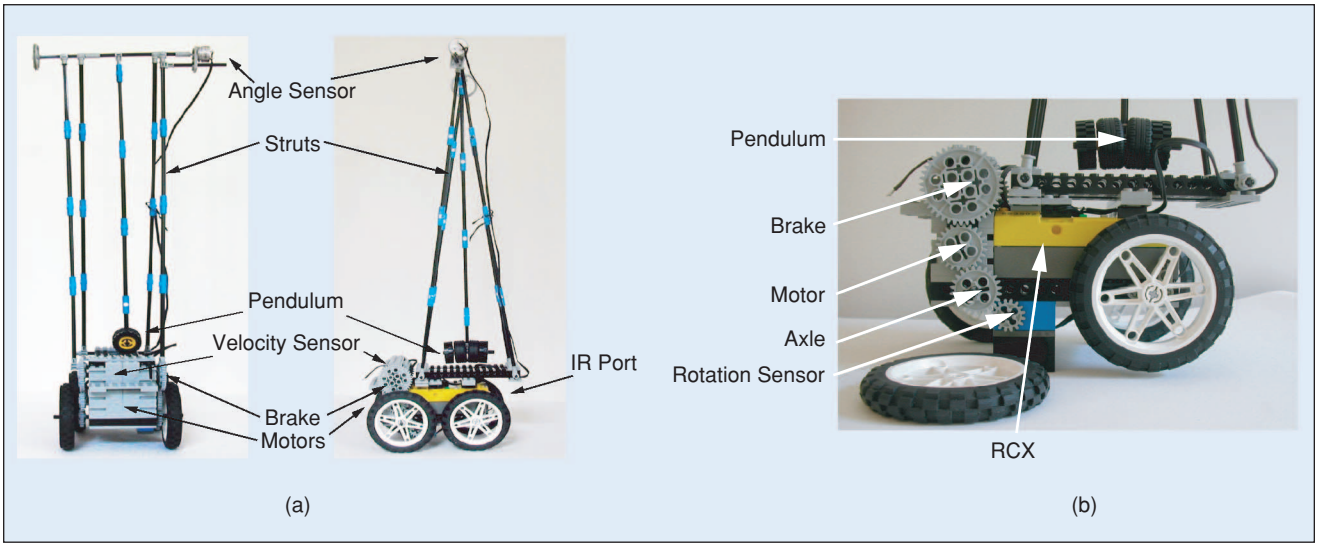



**Figure 4.** *The cart and pendulum system is constructed out of LEGO. The yellow brick is the RCX component containing a Hitachi H8 8-bit microprocessor, sensor and actuator interfaces, and IR communication port. (a) Overall system and (b) detail.*

**Table 1. Cart parts list. The cart is constructed from the LEGO components listed here with the modifications of Figure 3.**

| Quantity | Color | Description |
|---|---|---|
| 1 | Yellow | LEGO Mindstorms RCX |
| 1 | Black | LEGO Mindstorms IR Tower |
| 2 | Black | Brick 2 × 2 |
| 4 | Black | Electric Bricks 2 × 2 & connecting wire |
| 1 | Light Blue | Electric Rotation Sensor |
| 4 | Light Grey | Electric Mini-Motor 9 V |
| 10 | Light Grey | Plate 1 × 2 |
| 1 | Black | Technic Axle 6 |
| 2 | Black | Technic Axle 8 |
| 1 | Light Grey | Technic Axle Joiner |
| 6 | Light Grey | Technic Axle Pin |
| 4 | Black | Technic Brick 1 × 16 with holes |
| 1 | Light Grey | Technic Bush |
| 1 | Light Grey | Technic Gear 16 Tooth |
| 4 | Light Grey | Technic Gear 24 Tooth |
| 2 | Light Grey | Technic Gear 40 Tooth |
| 2 | Light Grey | Technic Liftarm 1 × 3 |
| 2 | Light Grey | Technic Liftarm 3 × 3 L-Shape |
| 4 | Light Grey | Technic Pin with Friction |
| 2 | Light Grey | Technic Plate 2 × 6 with holes |
| 7 | Light Grey | Technic Plate 2 × 8 with holes |
| 4 | Black | Tire 81.6 × 15 Motorcycle |
| 4 | White | Wheel 81.6 × 15 Motorcycle |

- Two motors fixed on a common axle to drive the rear wheels by means of a $24:24=1$ ratio gear (see Figure 4). The software converts a normalized control signal $-1 < u < 1$ into the full torque range of the motor drive.
- The *open-circuit* dc motor of Figure 3(a), coupled to the drive motor by means of a $24:40=0.6$ ratio gear, acts as a velocity sensor. The software converts this signal into the forward velocity of the cart $v_c$ m/s.
- The *short-circuited* dc motor of Figure 3(b), coupled to the drive motor by means of a $24:40=0.6$ ratio gear, acts as an electrodynamic brake. As discussed above, this motor provides velocity feedback to make the drive more linear.
- A standard rotation sensor, which is coupled to the motors by means of a $24:16=1.5$ ratio gear. The software converts this signal into the linear position of the cart $y_c$ m.
- The homemade rotation sensor of Figure 2, which is used as the suspension point of the pendulum. The software converts this signal into the horizontal movement of the pendulum $y_p$ m with respect to the cart.
- The gearing is shown in detail in Figure 4(b), where a driving wheel has been removed for visibility.

The software comprises the following programs:

- The real-time kernel legOS [9] version 0.2.4 (see "Online Resources"). Unlike later versions, this version has full support for the legOS Network Protocol (LNP) communications software and support for the serial IR communications port.

Students can upgrade this version to support the more recent USB IR communications port.

- Automatically generated controller code (see "Automatic Code Generation") implementing one of four controllers:
  - open-loop control
  - proportional control of cart position $y_c$
  - observer/state-feedback control of cart position $y = y_c$
  - observer/state-feedback control of load position $y = y_c + y_p$

**Table 2. Pendulum parts list. The pendulum and gantry are constructed from the LEGO components listed here with the modifications of Figure 2.**

| Quantity | Color | Description |
|---|---|---|
| 1 | – | Potentiometer position sensor; see Figure 2 |
| 2 | Light Grey | Plate 2 × 2 |
| 5 | Light Grey | Plate 2 × 10 |
| 16 | Light Blue | Technic Angle Connector #2 |
| 1 | Black | Technic Axle 5 |
| 3 | Black | Technic Axle 8 |
| 22 | Black | Technic Axle 12 |
| 2 | Light Grey | Technic Axle Joiner Offset |
| 4 | Black | Technic Brick 1 × 16 with holes |
| 8 | Light Grey | Technic Bush |
| 4 | Light Grey | Technic Connector |
| 6 | Light Grey | Technic Connector with Axle hole |
| 1 | Light Grey | Technic Plate 2 × 4 with holes |
| 2 | Light Grey | Technic Plate 2 × 6 with holes |
| 1 | Light Grey | Technic Pole Reverser Handle |
| 1 | Light Grey | Technic Pulley Large |
| 2 | Black | Tire 30.4 × 14 VR |
| 2 | Black | Tire Medium |
| 2 | White | Wheel 30.4 × 14 VR |
| 2 | Yellow | Wheel Center Large |

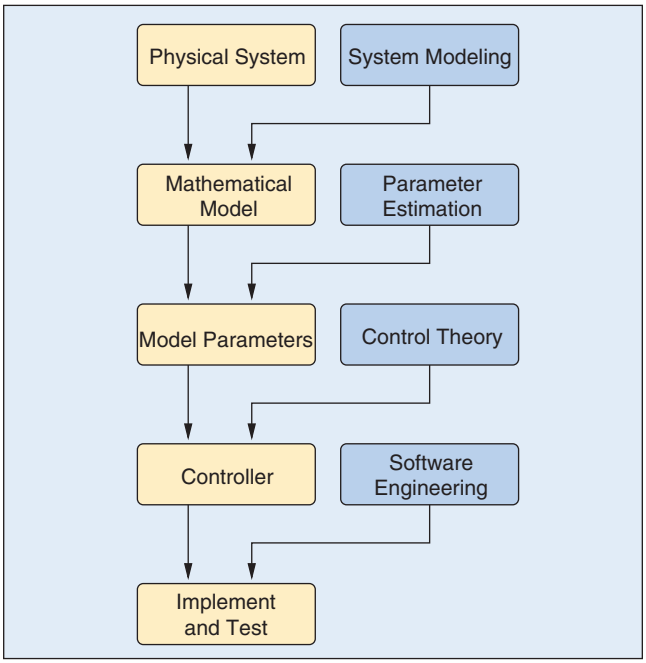



**Figure 5.** *From physical system to controller implementation. The implementation can be divided into several steps (left-hand boxes). Each step of the implementation requires a specific engineering skill (right-hand boxes). The LEGO cart and pendulum provides an effective testbed for this process.*

Students can implement their own controllers in C/C++. They can thus complete an entire design cycle from modeling and control design to discrete-time implementation.

- The infrared sensor is programmed using the LNP software to send five numbers back to the external computer at every sample instant: time $t$, pendulum absolute position $y = y_p + y_c$, control signal $u$, measured cart velocity $v$, and controller setpoint $w$. To save time, each floating-point number is scaled and converted into a two-byte integer.

Students should understand that the big-endian, little-endian controversy means that bytes have to be swapped to be meaningful to the computers at each end of the link. Students should also be aware of the accuracy/time tradeoff in choosing how to represent numbers over this slow (7 ms/byte) communication link.

## Experiments

This section presents one set of control experiments that can be performed with the cart and pendulum system. Our aim is to show that introductory experiments using proportional control, as well as more advanced state-space-based control, can be used to illustrate and teach basic control engineering skills. In particular, as indicated in Figure 5, the implementation of the real-time controller requires several steps, each of which exercises a particular engineering skill. Each of the following experiments requires the student to systematically follow the flowchart of Figure 5.

The outlines given below can form the basis for either prepackaged experiments, demonstrations for a first course in control, or open-ended projects for the more advanced student.

As illustrated in Figure 6, both classical and state-space methods can be used at the controller design stage.

### *System Modeling*

There are several ways to obtain a model of the cart and pendulum system in a form suitable for control. While this section considers system identification, other approaches are suggested as well.

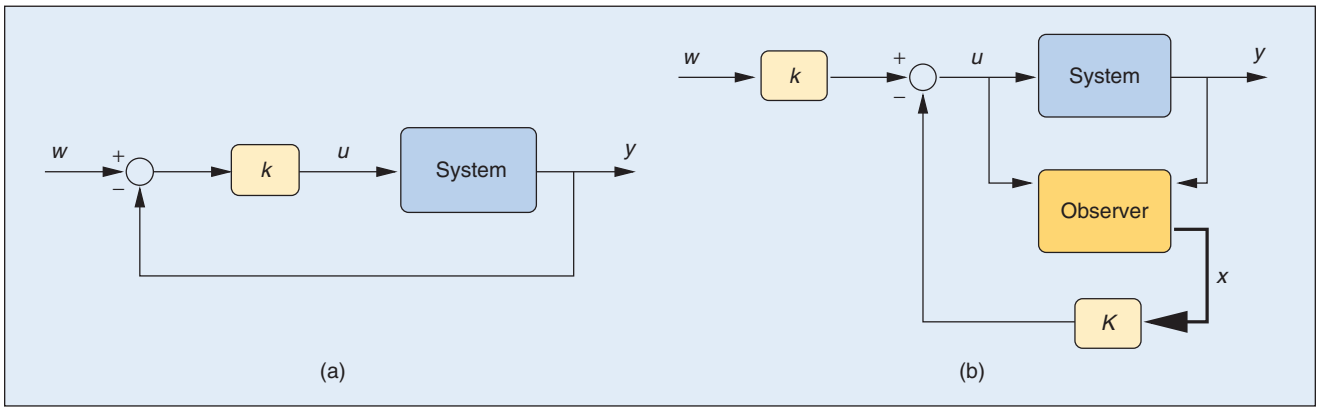



**Figure 6.** *Feedback control. (a) Proportional control and (b) state-space control. The System block can have either a transfer function or a state space representation. The system can be the cart with y = cart position, or the cart and pendulum with y = pendulum bob position. w is the desired value of y, and u is the control signal. (a) The control signal is proportional to the difference between w and y. k is chosen using classical control theory. (b) The observer generates the state estimate x in terms of the system model and the measured system input and output. In the case of the cart only, the vector x has two components, whereas, in the case of the cart and pendulum, x has four components. The state-feedback gain vector K multiplies the state estimate x to give a scalar signal. The gain k gives the correct steady-state value for y. The gains K and k and the observer are chosen using state-space control theory.*

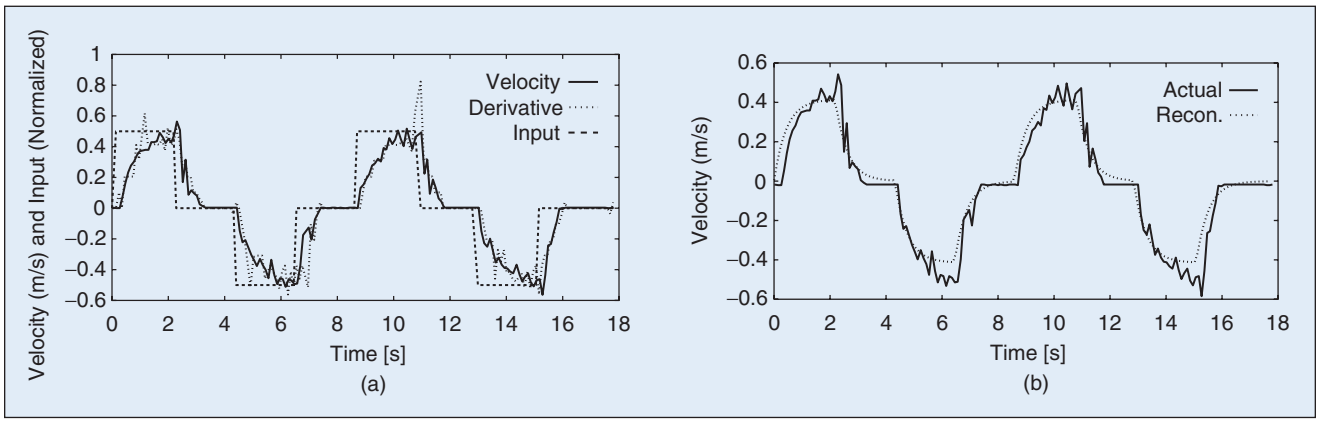



**Figure 7.** *System identification: (a) measured data and (b) reconstructed data. (a) The solid line shows the cart velocity v as measured by the velocity sensor of Figure 3(a), and the dashed line shows the system input u to the motor. As a check on the velocity sensor, the computed derivative ẏ of the cart position y is shown as a dotted line; as expected, the computed derivative is a noisy version of the measured velocity. (b) The solid line is the measured velocity of (a), and the dotted line is obtained by passing the system input u of (a) through the transfer function (1) using the identified parameters a and b of Table 3. Comparison of the two plots shows that the identified transfer function is a reasonable approximation of the true system dynamics.*

With the pendulum removed, the cart consists of several inertial elements (the inertias of the cart, motors, brake, and gears) and several resistance-like components (the friction of the motors, brake gears and cart, and the armature resistances). All of the inertias are rigidly coupled by the gear train of Figure 4(b), and thus there is only one independent inertia. Ignoring the effect of armature inductance, it follows that the cart subsystem relating input $u$ to linear velocity $v$ is first order. Assuming further that the system is linear, its transfer function can be written as

$$\frac{v}{u} = G_v(s) = \frac{b}{s+a}. \tag{1}$$

The corresponding transfer function relating to the "System" block of Figure 6(a) and (b) is $G(s)$ given by

$$\frac{y}{u} = \frac{1}{s} G_v(s) = \frac{b}{s(s+a)}. \tag{2}$$

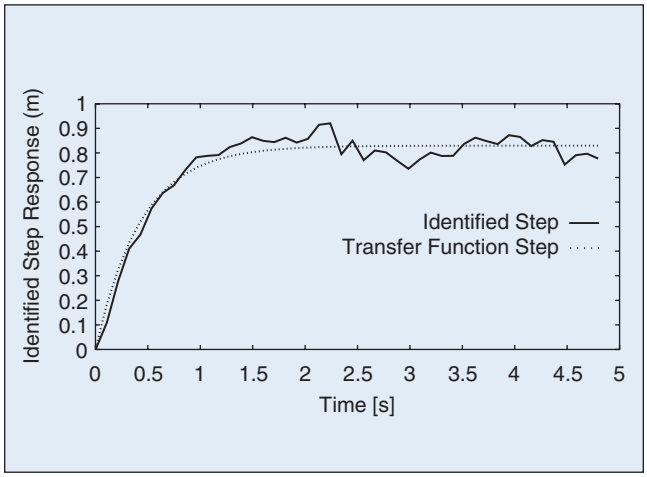



**Figure 8.** *Identified step response. The step response given by the solid line is identified using the measured velocity v and the input signal u of Figure 7(a) using the method of [11]. This identified step response is used to identify the unknown parameters of the transfer function (1); the step response of the transfer function (1) with the identified parameters a and b of Table 3 is given as the dotted line. As expected, the dotted line is a smoother version of the solid line indicating that the identified transfer function is a reasonable representation of the identified step response.*

**Table 3. Numerical values used to design the state-space controller. The parameter λ is the linear-quadratic control weight, σ is the measurement noise variance, and *a* and *b* are the identified cart parameters of (1) and (6).**

| λ | σ | *a* | *b* |
|---|---|---|---|
| 0.05 | 0.01 | 2.31 | 1.92 |

One approach to finding the parameters $a$ and $b$ is given in the next section.

## System Identification

The open-loop data of Figure 7(a) were generated, where the solid line shows the measured value of the cart velocity $v$. Following the methods of [10] and [11], the identification is accomplished in two stages. First, the frequency-sampling filter (FSF) approach of [10] is used to estimate the system step response given as the solid line in Figure 8 from the input-output data of Figure 7(a). Second, following [11] , a first-order model of the form of (1) is fitted to the estimated step response giving the numerical values of $a$ and $b$ appearing in Table 3. The corresponding step response is given as the dotted line in Figure 8. As a further comparison, the input data of Figure 7(a) are

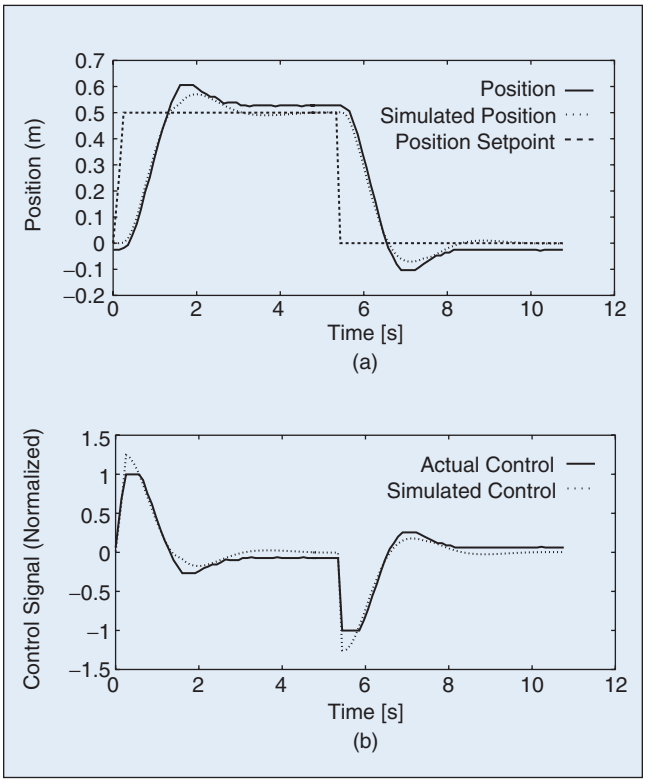



**Figure 9.** *Proportional control of the cart: (a) cart position and (b) control signal. This experiment enables the student to compare simulation with reality. Repeating the experiment with a different gain illustrates the tradeoff between control effort and performance. (a) This plot shows measured (solid) and simulated (dotted) cart positions superimposed on the setpoint (dashed). Although the match is not exact, the measured and simulated cart positions are similar. Discrepancies might be due to the sensor quantization and unmodeled nonlinear friction as well as the unmodeled control limits shown in (b). (b) The actual (solid) and simulated (dotted) control signals are different since the actual control signal is limited to ±1 and is less smooth due to feedback from the imperfect [(a)] actual system.*

passed through the transfer function of (1) with the parameters $a$ and $b$ of Table 3 to generate the dotted line of Figure 7(b), which is superimposed on the actual data $v$ appearing as a solid line.

Figure 10 shows a two-cart system that approximates the cart and pendulum system. The leftmost cart, labeled $m = 1$, represents the cart that we have already identified, while the spring and the second cart represent the pendulum. The length of the pendulum is $l_p = 0.38\,\text{m}$, and thus the natural frequency of the cart and pendulum system with the cart fixed is

$$\omega = \sqrt{\frac{g}{l_p}} = \sqrt{\frac{9.81}{0.38}} = 5.08\,\text{rad s}^{-1} = 0.81\,\text{Hz}. \tag{3}$$

The two-cart system of Figure 10 with the first cart fixed has the same natural frequency when the spring compliance is given by $c_p = (l_p/gm_p)$.

It turns out that the system transfer function is largely independent of $m_p$ as long as $m_p \ll 1$. Here, we somewhat arbitrarily choose $m_p = 0.1$. With this choice, together with (1), (3), and Table 3, the model of the system of Figure 10 is fully identified.

## Proportional Control of the Cart

With the pendulum removed, the system is described by (1), or, in terms of $y$, as

$$\frac{y}{u} = \frac{b}{s(s+a)}. \tag{4}$$

Root-locus analysis shows that the proportional controller

$$u = 2.5(w - y) \tag{5}$$

gives a closed-loop system with poles at $s = -1.1548 \pm 1.8588j$. Figure 9(a) shows the setpoint $w$ together with the actual and simulated system output $y$, and Figure 9(b) shows the corresponding control signals.

## State-Space Control of the Cart

From (4), a state-space model giving $y$ in terms of $u$ is given by

$$A = \begin{bmatrix} -a & 0 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} b \\ 0 \end{bmatrix}, \quad C = [\,0 \quad 1\,], \quad D = 0. \tag{6}$$

A standard observer/state-feedback controller [12], [13] is designed in the form

$$\begin{cases} \hat{y} = C\hat{x} \\ \frac{d\hat{x}}{dt} = A\hat{x} + Bu - L(\hat{y} - y) \\ u = kw - K\hat{x}. \end{cases} \tag{7}$$

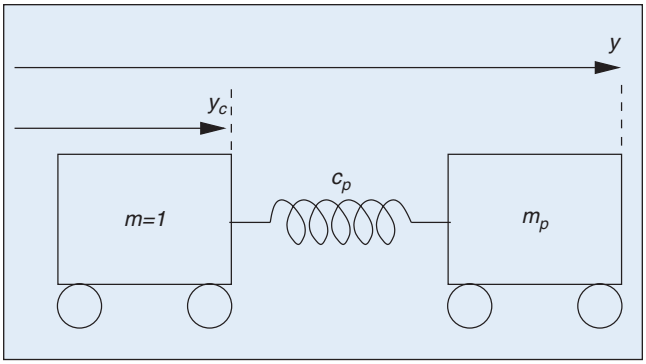



**Figure 10.** *Cart and pendulum system represented by a mass-spring system. The pendulum friction is small enough to ignore the resultant damping term.*

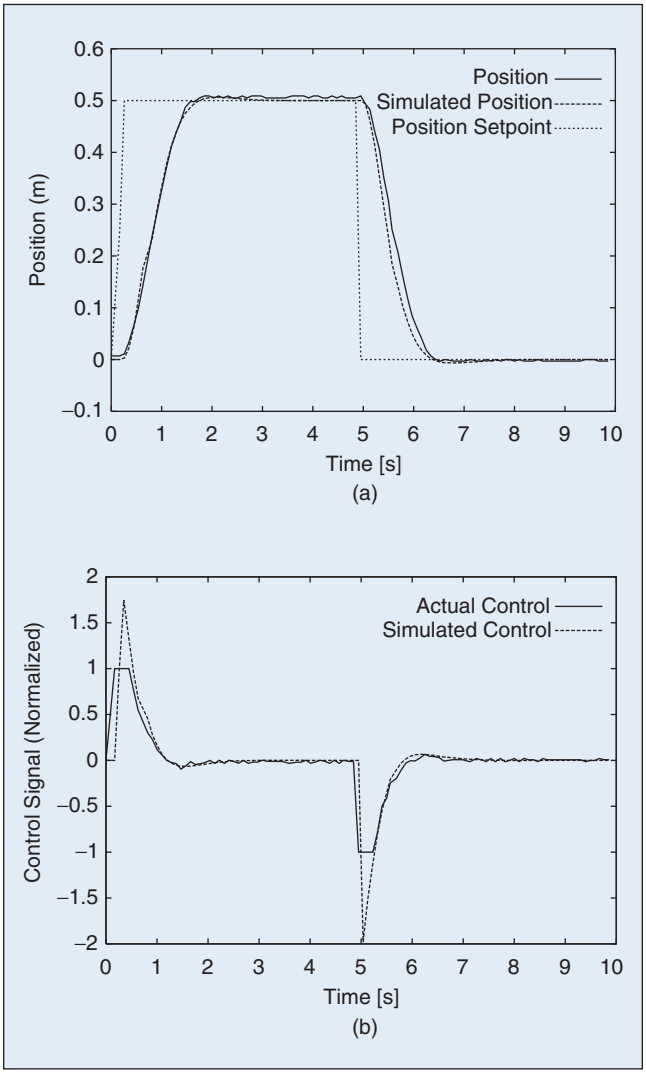



**Figure 11.** *State-space control of the cart: (a) cart position and (b) control signal. The state-space controller has better performance than the proportional controller of Figure 9. (a) This plot shows measured (solid) and simulated (dotted) cart positions superimposed on the setpoint (dashed). (b) This plot shows actual (solid) and simulated (dotted) control signals.*

The state-feedback gain $K$ is the solution to the steady-state linear quadratic optimization problem [12], [13]

$$J = \int_0^\infty y(t)^2 + \lambda u(t)^2 dt. \tag{8}$$

The observer gain $L$ is the solution to the steady-state optimal observer problem [12], [13], where the system states are corrupted by uncorrelated white noise with unit standard deviation and the measurement is corrupted by uncorrelated white noise with standard deviation $\sigma$. Numerical values for $\lambda$, $\sigma$, $a$, and $b$ appear in Table 3.

The numerical computation is performed using the-MATLAB-like program Octave (see "Online Resources"). A simple octave script is written to convert the equations implied by (7) and Table 3 into C code, together with a

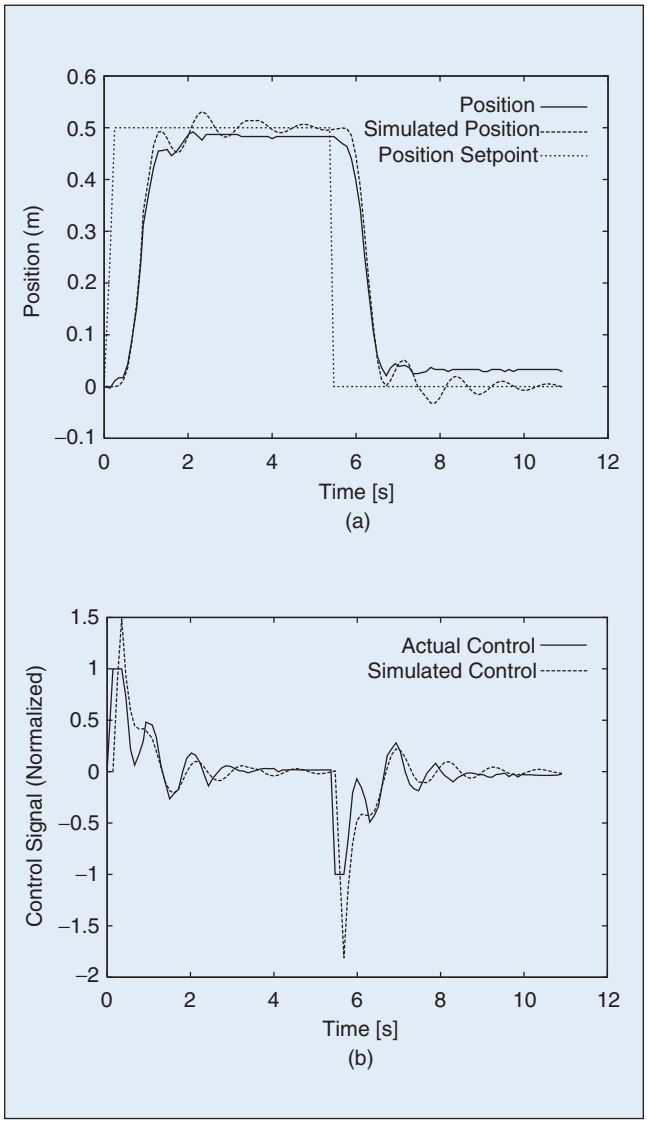



**Figure 12.** *State-space control of the cart and pendulum system: (a) load position and (b) control signal. (a) This plot shows measured (solid) and simulated (dotted) positions superimposed on the setpoint (dashed). Despite the low damping of the* open-loop *system, the* closed-loop *load position behaves in a well-damped fashion. (b) This plot shows the input (solid) and simulated input (dotted). The oscillations in the control signal show how the cart has to be moved to counteract the pendulum movement.*

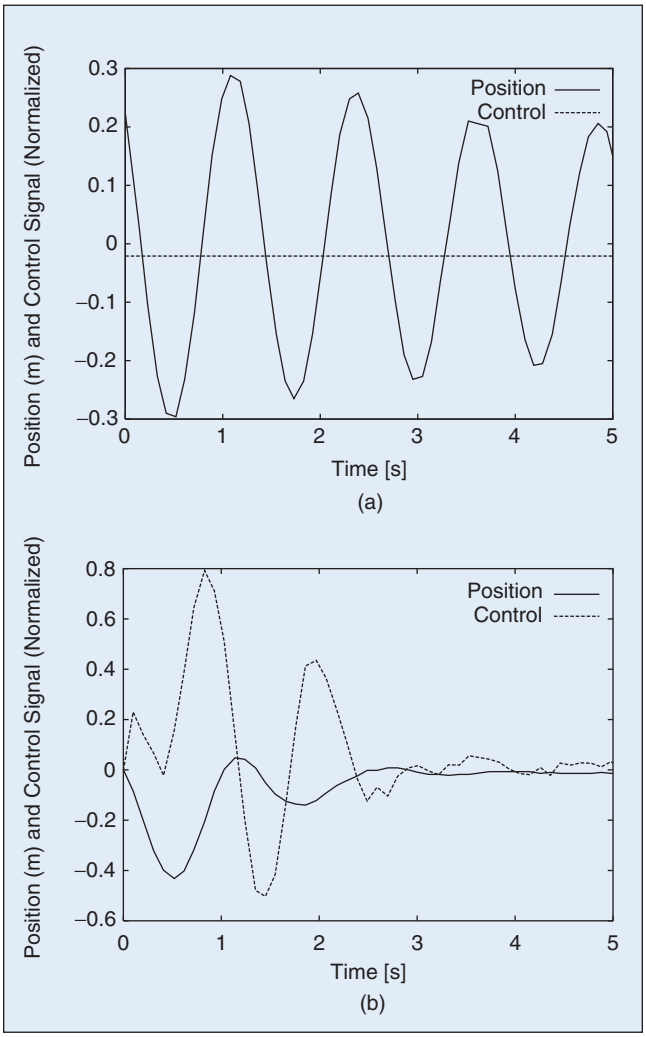



**Figure 13.** *Load position when (a) regulating cart position and load position when (b) regulating load position. In (a) the cart stays still but the load swings in an uncontrolled fashion. In (b) the cart moves so as to stop the swing of the load. It is a salutary experience for the student to try and stop the load swinging by* manually *moving the cart. (a) The cart position (dotted) is held by the controller at a constant value; this mode leaves the load position (solid line) free to swing. The amplitude of the load oscillation is reduced due to natural damping. (b) The feedback controller drives the load position to zero by moving the cart position (dotted line) to counteract the pendulum movement.*

simple Euler integration routine [14], which then becomes an include file for the overall compilation into RCX code (see "Automatic Code Generation").

Figure 11 shows the actual and simulated results. As can be seen, the performance of the state-space controller is better than that of the proportional controller of Figure 9.

**The LEGO Mindstorms kit is relatively cheap, robust, reconfigurable, reprogrammable, and induces enthusiasm and innovation in students.**

The state-space controller has a transfer function representation [12], [13]. Students can compare the relative performance of the two controllers and give an explanation using the transfer-function representation of each controller.

The student can observe the practical effects of LQ parameters as well as control-signal saturation. The student can devise an extended state-space controller to remove steady-state offset as discussed in [12] and [13].

## State-Space Control of the Cart and Pendulum

In a similar fashion, a fourth-order version of the observer/state-feedback controller of (7) is designed using the same optimization criterion to give the code shown in Code fragment 19. Once again, Figure 12 shows the set-point-following behavior of the actual and simulated control system. As in Figures 9 and 11, there are discrepancies between the simulated and actual result; in this case, there is a significant steady-state offset.

```
/* Angle sensor data */
#define MAX_INT_ANGLE 1023
#define MAX_ANGLE 3.142
#define INTERNAL_RESISTANCE 10000
#define POT_RESISTANCE 20000

/* Angle sensor code */
double pendulum_angle(double angle_0)
{
  double angle,resistance;
  unsigned int angle_raw;
  angle_raw = SENSOR_2/64;
  resistance = 1.0*angle_raw*INTERNAL_RESISTANCE/
               (MAX_INT_ANGLE-angle_raw); /* ohms */
  angle = 2*PI*(resistance/POT_RESISTANCE); /* Radians */
  angle = 0.85*angle;                  /* Fiddle factor!! */
  return angle-angle_0;
}
```

**Figure 14.** *Pendulum_angle.c. The BrickOS API provides the three short integer values SENSOR_1, SENSOR_2, and SENSOR_3, which contain the current raw sensor readings; we use the center sensor connection, and thus SENSOR_2 here. The remaining code converts the reading to the actual angle in radians. angle_0 is the value corresponding to the vertical pendulum.*

Perhaps the most dramatic desktop demonstration of the controller is as a regulator. With the controller switched off (using the RCX on/off button) the load is manually set swinging up to an angle of about 45°. The controller is then switched on, and the data recorded on the host machine.

When the cart position is regulated, the load swings freely with a slow decay [Figure 13(a)]. In contrast, with the load position regulated, the pendulum stops swinging within about 3 s [Figure 13(b)]. As can be deduced from the control signal u, the controller achieves this performance by moving the cart appropriately. In effect, the pendulum is shaken to a halt. This experiment makes an impressive benchtop demonstration for a lecture or class. An online movie is available (see "Online Resources").

## Summary

A nice feature of the LEGO system is that, although the mechanical, electrical, and structural construction is good

```
/* Velocity sensor data */
#define VELOCITY_FACTOR 342.0

/* Velocity sensor code */
double motor_velocity()
{
  int velocity_raw;
  double velocity;
  velocity_raw = (SENSOR_3/64) - 512;
  velocity = velocity_raw/VELOCITY_FACTOR;
  return velocity;
}
```

**Figure 15.** *Motor_velocity.c. As discussed in Code fragment 14, SENSOR_3 contains the raw sensor reading when the angle sensor is connected to the third sensor input. The remaining code converts the reading to the velocity of the cart in m/s.*

```
function eqns = controller2c (A,B,C,D,K,L,g_ss)
  ## usage:  eqns = controller2c (A,B,C,D,K,L,g_ss)
  ##
  ## Copyright (C) 2003,2004 by Peter J. Gawthrop
  ## $Id: controller2c.m,v 1.6 2004/04/06 08:56:04 peterg Exp peterg $
  [n_x,n_u,n_y] = abcddim(A,B,C,D); # Dimensions
  eqns = '';                              # Initialise string

  ## Error equations
  eqns = sprintf('#s  /* Error equations */\n',eqns);
  for i = 1:n_y
    eqn = sprintf('  e[#i] = -y[#i]', i-1, i-1);
    eqn = matrix2c(C,i,'x',eqn); # C matrix
  endfor
  eqns = sprintf('#s#s;\n', eqns, eqn);

  ## Observer equations
  eqns = sprintf('#s  /* State estimator */\n', eqns);
  for i = 1:n_x
    eqn = sprintf('  xdot =');
    eqn = matrix2c(A,i,'x',eqn); # A matrix
    eqn = matrix2c(B,i,'u_out',eqn); # B matric
    eqn = matrix2c(-L,i,'e',eqn); # L matrix
    eqn = sprintf('#s;\n  x[#i] = x[#i] + xdot*DT;\n', eqn,i-1,i-1);
    eqns = sprintf('#s#s', eqns, eqn);
  endfor

  ## Controller equations
  eqns = sprintf('#s  /* Control signal */\n', eqns);
  for i=1:n_u
    eqn = sprintf('  u[#i] = #g*w',i-1,1/g_ss);
    eqn = matrix2c(-K,i,'x',eqn); # K matrix
    eqns = sprintf('#s#s;\n', eqns, eqn);
  endfor

  ## Replace arrays by scalars
  if (n_u==1)
    eqns = strrep (eqns,'u[0]','u');
    eqns = strrep (eqns,'u_out[0]','u_out');
  endif
  if (n_y==1)
    eqns = strrep (eqns,'y[0]','y');
    eqns = strrep (eqns,'e[0]','e');
  endif
endfunction
```

**Figure 16.** *Controller2c.m. This Octave function converts the system matrices A, B, C, D, the feedback matrix K, and the observer matrix L to the corresponding C code.*

enough for successful results, there are enough discrepancies between theory and practice to introduce the student to real-world problems of controller implementation. This article concentrates on implementing the feedback controller within the RCX, with the host computer used for compiling, downloading, and data display. However, an interesting research project is to implement part of the controller (for example, optimization-based model-predictive control) on the host computer and communicate by means of the IR channel. The bandwidth restriction that this approach imposes can provide the basis for a research project on control through a restricted-bandwidth channel.

## Acknowledgments

This project would not have been possible without the help and support of LEGO.

The authors thank the reviewers for many helpful suggestions that have substantially improved this article.

```
function eqn = matrix2c(M,i,name,eqn);
  ## usage:  eqn = matrix2c(M,i,name,eqn);
  ## Writes equations for row i of matrix M with name 'name'
  ## Used in controller2c.m
  ## Copyright (C) 2003,2004 by Peter J. Gawthrop
  ## $Id: matrix2c.m,v 1.7 2004/04/05 08:24:33 peterg Exp $

  ## Defaults
  if nargin<3
    name = 'x';                    # Default name
  endif
  if nargin<4
    eqn = '';                      # Blank string
  endif

  ## Create string containing equation of row i
  [n,m] = size(M);                 # Dimensions
  for j = 1:m
    m_ij = M(i,j);
    if m_ij<0                      # sign symbol
      pm = '-';
    else
      pm = '+';
    endif
    a = abs(m_ij);                 # |m_i|
    if (a==1)
      eqn = sprintf('#s #s #s[#i]', eqn, pm, name, j-1);
    elseif (a~=0)
      eqn = sprintf('#s #s #g*#s[#i]', eqn, pm, a, name, j-1);
    endif
  endfor
endfunction
```

**Figure 17.** *Matrix2c.m. This Octave function converts the ith row of a matrix to the corresponding C code.*

## References


[1] D. Baum. *Definitive Guide to Lego Mindstorms*. Berkley, Ca: Apress, 2000.

[2] D. Baum, M. Gasperi, R. Hempel, and L. Villa. *Extreme Mindstorms*. Berkley, CA: Apress, 2000.

[3] M. Resnick, F. Martin, R. Sargent, and B. Silverman, "Programmable bricks: Toys to think with," *IBM Syst. J.*, vol. 35, no, 3&4, pp. 443–452, 1996.

[4] S. Papert, "What's the big idea? toward a pedagogy of idea power," *IBM Syst. J.* vol. 39, no. 3&4, pp. 720–729, 2000.

[5] J.B. Weinberg and Xudong Yu, "Robotics in education: Low-cost platforms for teaching integrated systems," *IEEE Robot. Automat. Mag.*, vol. 10, no. 2, pp. 4–6, 2003.

[6] F. Klassner and S.D. Anderson, "LEGO mindstorms: Not just for K-12 anymore," *IEEE Robot. Automat. Mag.*, vol. 10, no. 2, pp. 12–18, 2003.

[7] L. Greenwald and J. Kopena, "Mobile robot labs," *IEEE Robot. Automat. Mag.*, vol. 10, no. 2, pp. 25–32, 2003.

[8] R.D. Beer, H.J. Chiel, and R.F. Drushel, "Using autonomous robotics to teach science and engineering," *Commun. ACM*, vol. 42, no. 6, pp. 85–92, 1999.

[9] brickOS. An alternative operating system for the Lego Mindstorms RCX Controller [Online]. Available: http://brickos.sourceforge.net

[10] L. Wang and W.R. Cluett, "Frequency-sampling filters: An improved model structure for step-response identification," *Automatica*, vol. 33, no. 5, pp. 939–944, 1997.

[11] L. Wang, P.J. Gawthrop, C. Chessari, T. Podsiadly, and A. Giles, "Indirect approach to continuous time system identification of food extruder," *J. Process Contr.*, vol. 14, no. 6, pp. 603–615, 2004.

[12] H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems*. New York: Wiley, 1972.

[13] G.F. Franklin, J.D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems (3rd edition)*. Reading, MA: Addison-Wesley, 1994.

[14] G.F. Franklin and J.D. Powell. *Digital Control of Dynamic Systems*. Reading, MA: Addison-Wesley, 1980.

```
/* File Cart_con.c */
/* Created by MakeController.sh at Tue Apr  6 10:03:18 BST 2004 */
/* Using control weight lambda=0.05 and observer weight sigma=0.01 */
double controller(double y, double u_out, double w, double x[N_state])
{
  double u,e,xdot;
  /* Error equations */
  e = -y + x[1];
  /* State estimator */
  xdot = - 2.30968*x[0] + 1.91551*u_out - 1.68575*e;
  x[0] = x[0] + xdot*DT;
  xdot = + x[0] - 10.1672*e;
  x[1] = x[1] + xdot*DT;
  /* Control signal */
  u = 4.47214*w - 1.26875*x[0] - 4.47214*x[1];
  return u;
}
```

**Figure 18.** *Cart_con.c. Automatically generated state-feedback controller for the cart control of Figure 11. The sample interval DT is defined in the main program. u_out is the* constrained *control signal actually applied to the cart in the previous time instant. It is important to use u_out in the observer equations rather than the* computed *value u.*

```
/* File PendulumOnCart_con.c */
/* Created by MakeController.sh at Tue Apr  6 10:03:25 BST 2004 */
/* Using control weight lambda=0.05 and observer weight sigma=0.05 */
double controller(double y, double u_out, double w, double x[N_state])
{
  double u,e,xdot;
  /* Error equations */
  e = -y + x[3];
  /* State estimator */
  xdot = - 2.30968*x[0] - 2.58158*x[1] + 1.91551*u_out + 0.909516*e;
  x[0] = x[0] + xdot*DT;
  xdot = + x[0] - 10*x[2] + 4.68795*e;
  x[1] = x[1] + xdot*DT;
  xdot = + 2.58158*x[1] - 4.16972*e;
  x[2] = x[2] + xdot*DT;
  xdot = + 10*x[2] - 10.1683*e;
  x[3] = x[3] + xdot*DT;
  /* Control signal */
  u = 4.47214*w - 1.89038*x[0] - 7.78873*x[1] + 0.48409*x[2] - 4.47214*x[3];
  return u;
}
```

**Figure 19.** *PendulumOnCart_con.c. Automatically generated state-feedback controller for the cart and pendulum controller of Figures 12 and 13. Compared to Code fragment 18, this controller has four observer states to update, namely, the cart position and momentum, and the pendulum position and momentum.*

***Peter Gawthrop (P.Gawthrop@eng.gla.ac.uk)*** obtained his B.A., D.Phil., and M.A. degrees in engineering science from Oxford University in 1973, 1977, and 1979, respectively. Following a period as a research assistant with the Department of Engineering Science at Oxford University, he became W.W. Spooner Research Fellow at New College, Oxford. He then moved to the University of Sussex as a lecturer and later a reader in control engineering. In 1987, he took up the Wylie Chair of Control Engineering in the Department of Mechanical Engineering at Glasgow University, where he was a founding member of the Centre for Systems and Control. Since 2002, he has held an honorary research position at Glasgow. He has also held visiting appointments in Australia at the universities of New South Wales, Newcastle, Sydney, and the Royal Melbourne Institute of Technology. In the UK, he has a visiting position at the University of Bristol. His research interests include continuous-time model-based predictive control, system identification, and system modeling using the bond graph method. He can be contacted at the Centre for Systems and Control and Department of Mechanical Engineering, University of Glasgow, Glascow, G12 8QQ, Scotland.

***Euan McGookin*** graduated from the University of Glasgow in 1993 with a master of engineering degree in Avionics. After attaining his Ph.D. in marine vehicle control from Glasgow in 1997, he worked for the Defence Evaluation and Research Agency (DERA), UK. In 1998 he was appointed as a lecturer at the Department of Electronics and Electrical Engineering, University of Glasgow. His research interests are robotics, marine vehicle navigation (particularly submersibles), unmanned aircraft, gas turbine engine systems, and evolutionary optimization. He is a consultant for LEGO System A/S, working on robotics projects using LEGO Mindstorms.