S1 laboratory: PID control

Ivan Markovsky

November 25, 2010

Overview

This laboratory is related and complements the material taught in ELEC 2019 "Control and Systems Engineering" and ELEC 1011 "Communications and Control". It goes through a complete control system design cycle, starting with modeling of the plant, proceeding to design of a controller based on the model, implementation of the controller on a microprocessor, and finishing by experimental verification of the designed control system. Significant part of the work is on a computer and involves Matlab and C programming. Matlab is used for the parameter estimation and controller design and C is used for implementation of the control algorithm on the microcontroller.

Up-to-date version of this document and related files are available at:

`http://www.ecs.soton.ac.uk/~im/lego.html`

Preparation: Read Sections 1 and 2 and do the indicated assignments. The optional sections can be skipped.

1 Introduction and preparation

1.1 Proportional-integral-derivative (PID) control

A PID controller is a linear time-invariant dynamical system, defined by a transfer function

$$C(s) = k_p + k_d s + k_i s^{-1}$$

or equivalently by a differential equation

$$u(t) = k_p e(t) + k_d \frac{\mathrm{d}}{\mathrm{d}t} e(t) + k_i \int_0^t e(\tau)\mathrm{d}\tau. \tag{1}$$

The controller's input e is the error signal, *i.e.*, the difference between the reference r and the plant's output y, and the controller's output u is the plant's input, see Figure 1. The parameters k_p, k_d, and k_i specify the PID controller. They are called proportional, derivative, and integration gains, respectively.

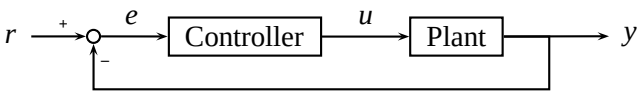

Figure 1: Feedback control scheme.

Designing a PID controller involves choosing the k_p, k_d, and k_i parameter values, so that the closed-loop system achieves given design specifications. Classical control design specifications impose upper bounds on the:

- rise time — the time required for the output to rise from 10% to 90% of its steady-state value,
- overshoot — the maximum value minus the steady-state value, and

- settling time — the time needed to reach and remain within 5% of the steady-state value.

A basic assumption in the above specifications is that a steady-state value exists. This implies that the controller achieves stability of the closed-loop system.

Choosing the parameter values (also called parameter tuning) is usually done by trial and error, following the heuristic rules summarised in Table 1. These rules specify the qualitative effects of increasing the PID controller parameter independently.

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
k_p	Decrease	Increase	Small change	Decrease	Degrade
k_i	Decrease	Increase	Increase	Decrease significantly	Degrade
k_d	Minor decrease	Minor decrease	Minor decrease	No effect in theory	Improve if k_d small

Table 1: Qualitative effects of increasing the PID controller parameter independently (source Wikipedia).

Preparation: Read more about PID control and the Ziegler-Nichols tuning method.

`en.wikipedia.org/wiki/PID_control`
`www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html`

1.2 Test platform: Lego Mindstorms NXT

In the laboratory exercise you are designing a PID controller for a cart, build from an Lego Mindstorms NXT constructor, see Figure 2. The cart includes a micro controller that is connected to an ultrasonic distance sensor and two electrical motors — one for the left wheels and one for the right wheels.

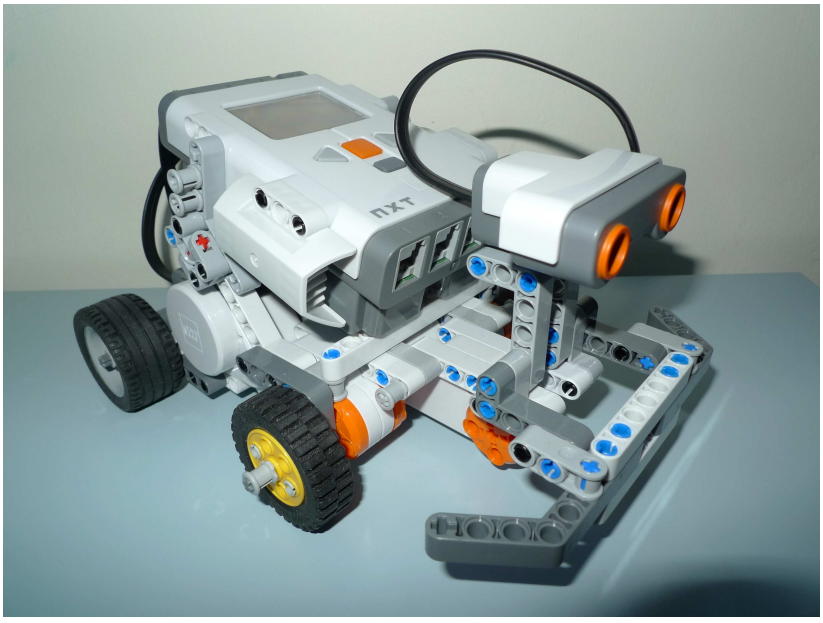

Figure 2: Test platform: a cart build from Lego Mindstorms NXT constructor.

Preparation: Read about Lego Mindstorms NXT.

`en.wikipedia.org/wiki/Lego_Mindstorms_NXT`
`www.philohome.com/nxtmotor/nxtmotor.htm`

The control objective is to keep the cart at a constant distance from a moving object. For simplicity the cart and the object is assumed to move along a line, so that the problem has one degree of freedom (back-forth motion). Therefore, the same power is applied to both motors (resulting in the cart moving back-forth).

The first task in the exercise is to identify a model for the relation between the power level applied to the motors (the control signal u) and the distance of the cart to the object (the controlled signal y). A parameterized model structure is postulated and the parameters are estimated from a measured response of the cart to a step input. The objective of the parameter estimation is to obtain a model that accurately reproduces the actual dynamical relation.

The PID controller design is initially carried out in Matlab (second task), where the controller is tested on the plant's model by simulation. Then, the designed controller is tested on the physical plant (third task) and the behaviour of the actual closed-loop system is compared to this observed in simulation.

1.3 Data preprocessing by moving average filtering (optional)

Data collected in a real-life experiment is often not suitable to use directly due to noise introduced by the measurement device and disturbances of the environment. The noise, however, often has higher frequency content than the desired signal, which allow us to separate them by low-pass filtering.

In Matlab the general filtering function, *i.e.*, simulation of a discrete-time linear time-invariant (LTI) dynamical system is `filter`. In case of a causal system, the filtering operation can also be done by the function `lsim` from the control toolbox.

Preparation: Find out how `filter` and `lsim` work by checking Matlab's documentation (type `doc filter` and `doc lsim` in Matlab).

There are different filter design methods (in the same way as there are different control design methods). In this laboratory, you will use a simple low-pass filter: the n-taps moving average filter

$$y_f(t) = \frac{1}{n}\big(y(t-1) + \cdots + y(t-n)\big). \tag{2}$$

Here y is the filter's input and y_f is the filter's output. The moving average filter is an LTI discrete-time dynamical system with a transfer function

$$F(z) = \frac{1}{n}\big(z^{-1} + \cdots + z^{-n}\big)$$

so it can be simulated by the `filter` function.

By default, however, `filter` assumes zero initial conditions of the filter, *i.e.*,

$$y(0) = \cdots = y(1-n) = 0$$

while in the laboratory setup, the cart generally starts from a nonzero initial condition, so that

$$y(0) = \cdots = y(1-n) = y_{\text{ini}},$$

where y_{ini} is the initial distance to the object. Ignoring the measurement noise, we can take $y_{\text{ini}} = y(0)$. Then, in order to take into account the initial condition in the subsequent filtering operation, we extent the measurement vector by $n-1$ samples equal to y_{ini}, filter the extended vector, and discard the first $n-1$ samples of the filtered signal.

Preparation: Justify the above procedure for filtering under nonzero initial conditions and write a Matlab function that implements it.

1.4 Least squares estimation (optional)

In the model identification task, you will fit the model response to the measured (and preprocessed) data by varying the model parameters. Although this task can be done by trial and error, in more complicated situations a systematic approach is needed. A widely used tool for parameter estimation is the linear least squares method, which is a method for finding an approximate solution of an incompatible linear system of equations. Using matrix-vector notation, the system is $A\theta \approx d$, where the matrix $A \in \mathbb{R}^{m\times n}$ and the vector $d \in \mathbb{R}^m$ are the given data and $\theta \in \mathbb{R}^n$ is the unknown vector of parameters. Since the system is overdetermined, there is more equations than unknown, *i.e.*, $m > n$.

The linear least squares method finds an approximate solution that minimizes the 2-norm

$$\|e\| := \sqrt{e_1^2 + \cdots + e_m^2}$$

of the approximation error $e := A\theta - d$. A least squares approximate solution always exists and is unique when A has rank equal to n. In this case, the best in the least squares sense parameter estimate is

$$\widehat{\theta} := (A^\top A)^{-1} A^\top d.$$

In Section 2.2 you will apply the linear least squares method for estimation of the parameters of the cart's model.

Preparation: Read more about the least squares method.

`http://en.wikipedia.org/wiki/Least_squares`

1.5 Simulink

The identified plant's model is a LTI system. The PID controller (1) is also LTI, so that the closed-looped system plant's model-controller is LTI. In Matlab, a general causal LTI system can be simulated by the function `lsim`. Although, in the exercise, it is possible to do all the simulation using `lsim`, in Sections 2.4 and 2.5 of the laboratory, you will use Simulink for the simulating of the closed-loop system.

Preparation: Read how to use Simulink (`doc simulink`) and study some of the demos. Implement and simulate the control system, shown in Figure 3 (set the plant parameters to $\widehat{a} = 1$ and $\widehat{b} = 1$ and tune the PID controller, as explained in Section 1.1.

1.6 Controller implementation

The PID controller is a continuous time dynamical system. In order to implement it on a digital microprocessor, it has to be converted to a discrete-time system; a process called discretization. In order to discretize the controller, we replace the derivative by a finite difference approximation and the definite integral by a sum:

$$\frac{\mathrm{d}}{\mathrm{d}t}e(t) \approx \frac{e(t) - e(t - t_s)}{t_s} \qquad \text{and} \qquad \int_0^t e(\tau)\mathrm{d}\tau \approx \sum_{k=1}^{\lfloor t/t_s \rfloor} t_s e(t_s k), \tag{3}$$

where t_s is the sampling time (in the range of 0.015 sec. to 0.02 sec. for the NXT microprocessor running the PID control and data collection tasks) and $\lfloor t/t_s \rfloor$ is the largest integer smaller than t/t_s. The approximation improves when the sampling time decreases, however, the hardware imposes limitation on the minimum sampling time.

An improvement of the PID controller is achieved by what is called "integral windup". The idea is to prevent the integral or the sum of the errors to grow too large and stay large even after the error gets small eventually. One possible modification to avoid this behaviour is to add exponential forgetting, *i.e.*, replace the sum in (3) by

$$\sum_{k=1}^{\lfloor t/t_s \rfloor} \lambda^k t_s e(t_s k),$$

where $0 < \lambda \leq 1$ is a design parameter, called forgetting factor.

Different programming languages can be used for implementing the discrete-time PID controller on the LEGO Mindstorms NXT microcontroller. In this exercise, you are given a program written in a C-like language, called Not eXactly C (NXC). NXC is specially created for development of Lego Mindstorms NXT applications and is supported by a powerful integrated programming environment, called the Bricx Command Center.

Preparation: Read about NXC and the Bricx Command Center.

`bricxcc.sourceforge.net/` and `bricxcc.sourceforge.net/nbc/`

1.7 Controller implementation

2 Work plan

2.1 Modeling

For model based control design, we need to model the dynamical relation between the motor power u and the distance to the object y. This relation is approximated by a linear-time invariant dynamical system with the following structure

$$H(s) = \frac{1}{s}\frac{b}{s+a}, \tag{4}$$

i.e., a first order system followed by an integrator. The first order system approximates the response from the motor power to the cart's speed. Then, the distance is obtained as an integral of the speed.

The model is specified by the parameters a and b. These parameter are determined so that the response of the model to a given input matches closely the measured output in a real-life experiment. Thus, for estimation of the parameters, we need measured data.

In order to measure a response of the cart experimentally, the NXT microcontroller should apply power to the motors and store the distance measurements of the ultrasonic sensor in a file. An NXC program that achieves this begins with definition of parameters:

5a ⟨s1.nxc 5a⟩≡ 5b▷

```
#define RUN_TIME 2000       // data collection time (in msec)
#define SAMPLING_PERIOD 50  // data collection sampling time (in msec)
#define STEP_LEVEL -50      // step motor power (integer between -100 and 100)

⟨PID controller coefficients ??⟩

#define MOTORS OUT_AC       // motors' connection
#define SENSOR IN_4         // sensor's connection
#define FILE_NAME "s.dat"

int dist;                   // target distance to the object

⟨Auxiliary functions ??⟩
```

The experiment involves two independent tasks: control and data acquisition. The NXC language allows concurrence, *i.e.*, simultaneous execution of two or more chunks of code, by defining functions with the keyword `task`.

5b ⟨s1.nxc 5a⟩+≡ ◁5a 6a▷

```
task save_data() {
  int  y, t; long IniTick;
  byte yb, handle;
  unsigned int result;
  const int file_size = RUN_TIME / SAMPLING_PERIOD;
```

```
  DeleteFile(FILE_NAME);
  result = CreateFile(FILE_NAME, file_size, handle);
  if (result == LDR_SUCCESS) {
    IniTick = CurrentTick(); t = 0;
    while (t < RUN_TIME) {
      t = IniTick - CurrentTick();
      if (t % SAMPLING_PERIOD) {
        y = SensorUS(SENSOR); yb = y;
        result = Write(handle, yb);
        if (result != LDR_SUCCESS) {
          TextOut(0, LCD_LINE7, "Data collection");
          TextOut(0, LCD_LINE8, "stopped."); beep();
          break;
        }
      }
    }
    CloseFile(handle);
  } else {
    TextOut(0, LCD_LINE2, "error"); NumOut(50, LCD_LINE2, result);
  }
}
```

In the step input measurement experiment, the control task is simple:

6a ⟨s1.nxc 5a⟩+≡ ◁5b 6b▷

```
task step_control() {
  OnFwd(MOTORS, STEP_LEVEL); Wait(RUN_TIME + 250); Coast(MOTORS);
}
```

⟨*PID control task* 9⟩

The main task is where the execution begins. The aim of this task is to initialize the sensor and start the control and data acquisition tasks. In the case of PID control, the target distance is measured before starting the control task.

6b ⟨s1.nxc 5a⟩+≡ ◁6a

```
task main() {
  SetSensorLowspeed(SENSOR);
  if (!false) { // PID control
    TextOut(0, LCD_LINE1, "To measure DIST"); WaitPressM();
    dist = SensorUS(SENSOR);
    ClearScreen();
    TextOut(0, LCD_LINE3, StrCat("DIST = ", NumToStr(dist)));
    TextOut(0, LCD_LINE1, "To start"); WaitPressM();
    TextOut(0, LCD_LINE1, "Tracking ...");
    start pid_control;
  } else       // step input
    start step_control;
  start save_data;
}
```

Task 1: Download from the lab's webpage the nxc program file `s1.nxc`. If necessary, modify the main function, so that open-loop step response is measured. Then, using the Brix Command Center, compile and transfer the program to the microcontroller. Before executing the program, prepare the cart by positioning it in front of an object to which distance will be measured and make sure that there is enough space for the cart to move unobstructed. Note that, by default negative power is applied, so that the cart is moving backwards. Finally, copy the created data file from the microcontroller to the PC, using the the Brix Explorer available in the tools menu of the Brix Command Center.

Task 2: Using the functions `load_data`, load in Matlab the data file created in Task 1. Plot and examine the data. Can you explain the measured response by the model (4)? Filter the data with the function `ma_filter` and plot the filtered data on top of the original data. In doing these and the following tasks in Matlab, you may find useful the provided template script file. If you write your own code, work in a file rather than on the command line (type `edit` in Matlab in order to evoke an editor).

2.2 Parameter fitting

Analytical expression for the response of the model defined by the transfer function (4) to a step input under initial conditions

$$s(0) = s_{\text{ini}} \qquad \text{and} \qquad \frac{\mathrm{d}}{\mathrm{d}t}s(0) = 0$$

is

$$s(t) = s_{\text{ini}} + b\tilde{s}(t), \qquad \text{where} \quad \tilde{s}(t) := \frac{1}{a}t - \frac{1}{a^2}(1 - e^{-at}), \qquad \text{for } t \geq 0. \tag{5}$$

For specific parameter values, the response s can be obtained numerically and plotted by evaluating the analytic expression or by simulation of the model (`lsim` function in Matlab), which does not require an analytic formula for the response.

Task 3: Compare the analytical and simulated responses of a model with parameters $s_{\text{ini}} = 1$, $a = 1$ and $b = 1$.

The parameter fitting problem is to find the parameters s_{ini}, a, and b that "best" fit the response

$$s_{\text{d}} = \big(s_{\text{d}}(0), \ldots, s_{\text{d}}(T-1)\big)$$

measured in Task 1. The notion of best fit can be quantified for example by the root mean squares fitting error

$$f(s_{\text{ini}}, a, b) := \sqrt{\sum_{k=0}^{T-1} \big(s_{\text{d}}(k) - s(t_s k)\big)^2},$$

where s is given by (5). A few approaches for solving the parameter fitting problem (which are common in solving other problems as well) are:

Trial and errors

Guess values for s_{ini}, b, a, and plot the resulting response s on top of the measured data s_{d}. Modify the initial guess trying to make s fit s_{d} better. Repeat until no improvement is possible.

Trial and error without some way of making educated guesses is not very effective.

Heuristic

Note from (5) that $s_{\text{ini}} = s_{\text{d}}(0)$. This equation determines one of the parameters. For large t, b/a determines the slope of s, *i.e.*, $b/a = \tan(\alpha)$, where α is the asymptotic slope of s. This gives one equation for the parameters b and a. In order to determine a and b uniquely another equation is needed. Consider the line defined by the equation

$$s'(t) = s_{\text{ini}} + \frac{b}{a}t.$$

It has the same slope as the asymptote of s but does not have the transient due to the first order term $1/(s+a)$ of the transfer function. The difference $s(t) - s'(t)$ can be calculated and for large t is approximately equal to b/a^2. This give us a second relation for a and b.

The heuristic approach gives an initial guess for the trial and errors approach, which can be used for further improving the estimated parameters. In addition, intuition of how the change of parameters influences the response (s_{ini} is the initial distance, b/a is the gain of the system, and $1/a$ is the time constant) makes the combination of heuristics and trial and errors a powerful approach.

Task 4: Find the parameters s_{ini}, b, and a, using the heuristic and trial and errors methods.

For complex problems (many parameters that interact in a nonintuitive way), however, there is a need for systematic methods. Such methods are derived by analysis and numerical computations.

Analytical (optional)

Assuming for a moment that a is known, the problem that we want to solve is

$$\text{minimize} \quad \text{over } s_{\text{ini}} \text{ and } b \quad f(s_{\text{ini}}, b, a). \tag{6}$$

Define the vector d, containing the measured data, and the matrix A, determined by the function $\tilde{s}$, defined in (5)

$$d := \begin{bmatrix} s_{\text{d}}(1) \\ \vdots \\ s_{\text{d}}(T) \end{bmatrix}, \qquad A := \begin{bmatrix} 1 & \tilde{s}(1) \\ \vdots & \vdots \\ 1 & \tilde{s}(T) \end{bmatrix}.$$

Using the above vector-matrix notation and the vector 2-norm $\|\cdot\|$, problem (6) becomes

$$\text{minimize} \quad \text{over } s_{\text{ini}} \text{ and } b \quad \left\| d - A \begin{bmatrix} s_{\text{ini}} \\ b \end{bmatrix} \right\|, \tag{7}$$

which is a standard linear least squares problem. Therefore, the optimal parameter estimates are

$$\begin{bmatrix} \widehat{s}_{\text{ini}} \\ \widehat{b} \end{bmatrix} = (A^\top A)^{-1} A^\top d,$$

and the minimum value of f over s_{ini} and b is given by

$$\tilde{f}(a) = \sqrt{d^\top A (A^\top A)^{-1} A^\top d}.$$

Task 5: Write a Matlab function, called `est`, that accepts as an input a parameter a, a vector of step response samples d, and a vector of corresponding time instances t and returns as an output the minimum value of (7).

Numerical (optimal)

We still have to determine the best value of the parameter a, *i.e.*, the remaining problem is

$$\text{minimize} \quad \text{over } a \in \mathbb{R}_+ \quad \tilde{f}(a).$$

Unfortunately this problem has no simple analytic solution. However, it can be solved numerically, *e.g.*, by evaluating $\tilde{f}$ for a range of candidate a's and choosing the a that corresponds to the minimum value of $\tilde{f}$. This procedure is similar to plotting the function $\tilde{f}$ (in Matlab, you can use `fplot`) and selecting the minimum value by visual inspection. A more sophisticated method for finding the minimum of $\tilde{f}$, however, is to use numerical optimization, *e.g.*, the function `fminbnd` from the optimization toolbox of Matlab.

Task 6: Find the optimal estimate of the parameter a visually by plotting the function $\tilde{f}$ and automatically by minimizing $\tilde{f}$. Then defined the identified model and compare the simulated response of the model to the measured data.

2.3 Validation

Task 7: Take new step measurements for different motor power levels and check how they are fitted by the identified model `sysh`.

2.4 Model based PID controller design

In order to simulate the closed-loop system, you can use the Simulink scheme, shown in Figure 3. The initial condition of the integrator is set to `sini`.

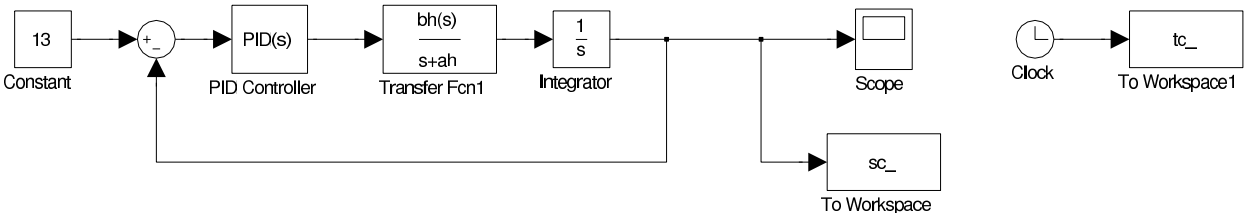

Figure 3: Simulink scheme of the closed-loop system.

Task 8: Tune the PID model parameters in Matlab, simulating the closed-loop system plant's model-controller.

2.5 Test the controller on the plant

The PID control task is implemented as follows chunk of the NXC program `s1.nxc`:

9 ⟨*PID control task* 9⟩≡ (6a)

```
inline int max(int a, int b) {return (a > b) ? a : b;}
inline int min(int a, int b) {return (a < b) ? a : b;}
inline int clip_to_100(int a) {return min(max(a, -100), 100);}

task pid_control()
{
  int y, u, e, e_old, diff; long sum = 0;

  while (true) {
    y = SensorUS(SENSOR);
    e_old = e;
    e     = y - dist;
    if (e > ERROR_TOL) {
      TextOut(0, LCD_LINE1, "I've lost it!"); break;
    }
    sum  = e + 0.95 * sum;
    diff = e - e_old;
    u    = clip_to_100(KP * e + KI * sum + KD * diff);
    OnFwd(MOTORS, u);
    ⟨Diagnostics ??⟩
  }
}
```

Task 9: Modify `s1.nxc` changing the input from step to the PID controller, designed in Task 8. Compile and execute `s1.nxc` and copy the created data file.

Task 10: Compare the measured closed-loop response to the response simulated in Matlab. (Make sure the input and the initial conditions in the simulation are the same as the one in the real-life experiment.) If necessary, revise the controller coefficients to improve the performance of the closed-loop system with the plant.

---

3 Optional

Task 11: Make the cart, running the designed PID control, track the cart of your neighbours, which tracks an moving object or yet another cart. Do the last cart in the chain track properly the cart in front?

Summary of things you did as part of S1

- PID controller tuning.
- Discretization of a continuous-time plant.
- Programming of a microcontroller in C.
- Use of multitasking and concurency in C.
- Use of Matlab and the Control Toolbox for data processing and data modeling.
- Data preprocessing by a moving average filter.
- Parameter estimation by solving a least squares optimization problem.
- Use of Simulink for simulation of an interconnected systems under nonzero initial conditions.

Further things, related to S1, you can try

- Robot design with Lego Mindstorms NXT.
- Object tracking in the plane (two degrees of freedom).
- Building realistic plant models with Simulink.
- Choice of sensor location, avoiding nonminimum phase dynamics.
- Application of advanced control methods (*e.g.*, the state space methods taught in the 3rd year control courses).