THREE TERM (PID) CONTROLLER IMPLEMENTATION FOR THE H8 MICROPROCESSOR

Status: Final

Version: 1.0

Date: 27 September 1994

Authors: M W Rand

Signed:

Approved:

Highfield Software Ltd
315 Portswood Road
SOUTHAMPTON
SO2 1LD

1. INTRODUCTION

The source code file PID.FTH contains an implementation of a classic three-term (PID) controller algorithm in Forth. The code has been developed for the Hitachi H8 microprocessor using the MPE Cross-compiler Version 5, and tested with the TDS 2020 single-board computer.

All words are written in high-level Forth, and should be capable of porting to any Harvard architecture processor with little or no adaptation. To port to other processor architectures, it should only be necessary to modify the defining word **Controller** and the set of data access words associated with it.

The implementation is for integer arithmetic, and is designed to operate with control variables in ranges up to -10000..10000 (giving a maximum of 4 digits precision).

2. APPLICATION NOTES

The three-term control algorithm is implemented via a defining word **Controller** which creates a three term controller object, and a set of words for accessing and manipulating such a controller. Most of these words validate that the address given is a reference to a controller object and return a flag to the client word to indicate that the word could be executed on the object.

2.1 SETTING CONTROLLER PARAMETERS

Words are provided for setting the constants of the proportional, integral and derivative terms of the control algorithm (**SetKp**, **SetKi**, **SetKd**). Each of these takes two 16-bit integer parameters representing numerator and denominator of a rational value. The words check that the fraction represented is in the range 0..1.

When a controller is created, the coefficients are set at (K_p = 1, K_i = 0, K_d = 0).

SetDeadband sets a deadband for the computed error (i.e. setpoint - process variable) which controls whether the output value is refreshed and can therefore suppress output 'jitter' on a process which is effectively in control. The output is not recomputed if the average of the last two error values is in the range: (-deadband value) .. (deadband value).

When a controller is created, its deadband value is set at 0.

SetSP sets the setpoint for the controller. The value given must be less than 10000. The initial setpoint assigned when a controller is created is 0.

SetSlidingWindow sets the size of the 'sliding window' on past error readings. In this implementation, the integral term is computed by multiplying the average of the last n error values by the integral term coefficient, where n is the sliding window size. By default, a sliding window size of 64 is set up when a controller object is created. The user may reduce the size but in this implementation cannot increase it beyond this size. Reducing the window size does not affect the storage allocated to a controller object or the performance of the algorithm in computing the integral term. The main purpose of changing the size is to avoid an excessive bias effect at changeover where there is a short term 'square wave' profile to the setpoint.

When a controller is first run, and subsequently on each occasion when **SetSlidingWindow** is invoked, the buffer of past error readings is cleared.

2.2 READING THE CONTROLLER STATE

A set of words (**ReadOutput**, **ReadPV** and **ReadError**) provides access to the current settings and readings of a controller, returned as 16-bit signed values.

ReadLockedStatus returns a flag which, when **True**, inidcates that on the last iteration the output was not recomputed because the average of the last two errors was within the deadband.

2.3 USER HOOKS

A **Controller** object operates on output, measured (PV) and setpoint values that are uniformly scaled and dimensionless. By default, a controller feeds its output back to its process value via a simulation (see the next section) and, given an opportunity, will 'free run' as often as it is invoked to carry out one iteration of the control algorithm. Three hooks are provided to allow the user to specify the handling of the output, the supply of the process value, and the timing of its next iteration.

OutputMethod allows the user to specify a word to call whenever the controller has completed a single PID step. The word can be used to scale or otherwise condition the output signal, send it to a real-world output device, or whatever else is needed to make the control useful. It should take a 16-bit signed value on the stack (the internal output value) and return nothing. It is called whether or not the controller is 'locked' through deadbanding, so that it can correctly handle output methods that are time-dependent such as pulse train generation.

InputMethod allows the user to specify a word to call whenever the controller is about to execute a single PID step. The word can be used to acquire data from the real world and to scale or otherwise condition the input signal. It should take no parameters on the stack and return a 16-bit signed value (the process value) which can be directly compared with the setpoint value.

SetSamplingTrigger allows the user to specify a word called at each pass when the controller is run continuously with the word **Run**. It should take no parameters and return a logical on the stack, set **True** if the control algorithm should be single-stepped on this pass. This allows the actual control to be synchronized with external events such as timers. A controller object is created with a default trigger routine installed which always returns **True**, and thus will free run a controller.

2.4 SIMULATION

When a new Controller object is created, the **InputMethod** and **OutputMethod** vectored user routines are plugged with default routines which generate a process variable value by simulating a load on the output. The recurrence relation used in this simulation is as follows:

Where P_i = output value at time i,
V_i = process value at time i,
i = 1,2,3...

$z1_0 = 0$
$z2_0 = 0$
$z1_i = 5P_i + z1_{i-1}$
$V_i = z2_i = (0.52z1_i + z2_{i-1})/2$

Note: this simulation as supplied works on a single set of internal variables and therefore cannot be used as the default behaviour for more than one Controller object at a time. It could be modified to use variables specific to each Controller, but at the cost of increasing the RAM requirement for each Controller, whether or not it is of interest.

2.5 RUNNING THE CONTROLLER

Words are provided to single-step the controller, run it continuously, and control continuous running in a multitasking environment.

Kick will single-step the given Controller object as follows:

```
Invoke the user-supplied or default vectored routine to return the current process value;
Recalculate the error from (setpoint - process value);
Update the error 'history' for integral and derivative term calculations;
IF two latest (absolute) error terms show deviation from deadband:
        Recalculate the output;
END;
Invoke the user-supplied or default vectored routine to process the output.
```

Run will continuously run the Controller. As supplied it will run in a single-tasking environment and take over the single Forth thread, but can be stopped by invoking **Stop** for the Controller (see below) within an interrupt service routine. By removing the commenting from PAUSE in a multitasking environment, it can run in a multitasking environment and any other thread can then invoke **Stop** on the Controller to terminate its run.

The action of **Run** is as follows:

```
WHILE not signalled to stop:
        ( Yield to other processes);
        Execute vectored routine to check whether to trigger next iteration;
        IF next iteration to be triggered:
                Single step Controller;
        END;
END;
```

2.6 DEBUG AND TESTING

The control algorithm has a simple debug mode which can be switched on globally for all defined Controller objects by setting the variable **Debug** to **True**. In debug mode, each time a controller is single-stepped through its control algorithm it will report on the Forth console: the number of iterations to date; the current setpoint, process variable and error values; newly calculated values for the porportional, integral and derivative terms; and, if deadbanding does not come into play, the new output value.

A couple of testing routines are provided. **InitController** will install a set of Controller parameters that are ‘stable’ for the simulation provided. **Test** will switch the system into ‘debug’ mode and repeatedly single-step the specified Controller each time the user presses a key, until he or she presses ‘Esc’.

3. BUILD

The build control file is supplied on disk as PIDAPP.CTL, which is a direct adaptation of the supplied file TDS532.CTL. The controller words are supplied in PID.FTH, which is expected to reside in the same directory as TDS352.CTL. All other files assume the same directory structure as TDS352.CTL.

As supplied, the multitasker is not built as part of the H8 target. Although most of the words intended to be internal in PID.FTH are declared as INTERNAL, there is insufficient ROM space to generate an open Forth system with both the multitasker and PID.

4. TESTING

The results of two test runs using the supplied **Test** word are documented in Annexe A. The tables show a complete listing of values through the run, together with points at which controller parameters were changed. The graphs give a good visual indication of setpoint, process variable and output.

Run 2 shows the results with the simulation as supplied. Run 1 was generated with a bug in the simulation which caused a large erratic deviation when a large change was made in the setpoint. It is included for interest in showing the controller recovering from a 'pathological' condition, which is similar to that sometimes seen in flow control systems when the flow measuring device is mounted in an area of turbulence near the control valve.

The run 1 graph shows:

- Limited effect of integral term on startup;
- One cycle lag in process variable matched by overswing in output(12, 32 etc.);
- 'Bias' in stable conditions from prominence of integral term (40-50) subsequently reduced by change in integral coefficient;
- Increase in proportional term (70) leading to osciallation, but reducing bias yet further in stable conditions.

The test can be repeated by executing:

```
Controller C1
C1 InitController
C1 Test
```

and subsequently interrupting the run and injecting new parameter values at the appropriate ponts.

5. GLOSSARY

A complete source listing is attached at Annexe B.

Controller

Usage: Controller <Name> (---)

<Name> (--- addr)

Creates a controller object with a default set of properties and behaviour as described below under individual words.

InputMethod

Usage: (addr1 addr2 --- t/f)

Allows the user to define a routine to process the input signal each time the controller's control algorithm is invoked and return it as the process variable. This could involve reading the signal from a physical device and/or conditioning it. The routine should take no parameters and should return a single 16-bit integer on the stack. By default a controller installs a routine which reads a value for the process variable from that internally stored by the default (simulation) output method.

If 'addr2' appears to reference a valid controller object, returns TRUE and 'addr1' is stored as the execution address of the input routine. Otherwise returns FALSE.

Kick

Usage: (addr -- t/f)

Single-shots the control algorithm for the controller specified by 'addr'. If 'addr' does not appear to reference a valid controller object, returns FALSE. Otherwise, calls the installed routine to input the process variable (see 'InputMethod' above), recalculates the output, calls the installed routine to process the output (see 'OutputMethod' above), and returns TRUE.

OutputMethod

Usage: (addr1 addr2 --- t/f)

Allows the user to define a routine to process the output signal each time it is refreshed by the controller. This could involve conditioning the signal and/or sending it to a physical device. The routine should take a single 16-bit integer from the stack and return nothing on the stack. By default a controller installs a routine which calculates a new simulated value for the process variable and stores it internally.

If 'addr2' appears to reference a valid controller object, returns TRUE and 'addr1' is stored as the execution address of the output routine. Otherwise returns FALSE.

ReadOutput

Usage: (addr --- n t/f)

Returns TRUE and the current value of the controller output if 'addr' appears to reference a valid controller object, otherwise returns FALSE and ‘n’ is indeterminate.

ReadPV

Usage: (addr --- n t/f)

Returns TRUE and the current value of the process variable if 'addr' appears to reference a valid controller object, otherwise returns FALSE and ‘n’ is indeterminate.

ReadError

Usage: (addr --- n t/f)

Returns TRUE and the current value of the error if 'addr' appears to reference a valid controller object, otherwise returns FALSE and ‘n’ is indeterminate.

ReadLockedStatus

Usage: (addr --- t/f1 t/f2)

Returns TRUE in 't/f2' and the current locked status in 't/f1' if 'addr' appears to reference a valid controller object, otherwise returns FALSE in 't/f2' and 't/f1' is indeterminate. The locked status is TRUE if the setpoint is latched because the error is within the deadband limit.

Run

Usage: (addr ---)

Primarily for use in multitasking environments. Puts the current thread into an indefinite loop which can only be terminated by executing 'Stop' on the controller from another thread or an interrupt (see below). In each iteration it invokes the installed routine to check the sampling trigger (see 'SetSamplingTrigger' above) and if it is set, calls 'Kick' to recompute the output. Yields control after each iteration.

SetKp

Usage: (n1 n2 addr --- t/f)

Sets the constant of the proportional term of the controller and returns TRUE if the constant is in range and 'addr' appears to reference a valid controller object, otherwise returns FALSE. The constant is represented by n1/n2. The default value of the constant is 1.

SetKi

Usage: (n1 n2 addr --- t/f)

Sets the constant of the integral term of the controller and returns TRUE if the constant is in range and 'addr' appears to reference a valid controller object, otherwise returns FALSE. The constant is represented by n1/n2. The default value of the constant is 0.

SetKd

Usage: (n1 n2 addr --- t/f)

Sets the constant of the derivative term of the controller and returns TRUE if the constant is in range and 'addr' appears to reference a valid controller object, otherwise returns FALSE. The constant is represented by n1/n2. The default value of the constant is 0.

SetSP

Usage: (n addr --- t/f)

Sets the controller setpoint and returns TRUE if the setpoint is in range and 'addr' appears to reference a valid controller object, otherwise returns FALSE. The default value of the setpoint is 0.

SetDeadband

Usage: (n addr --- t/f)

Sets the controller deadband (i.e. the absolute value for the error below which no change will be made to the output) and returns TRUE if the deadband is in range and 'addr' appears to reference a valid controller object, otherwise returns FALSE. The default value of the deadband is 0.

SetSlidingWindow

Usage: (n1 addr --- n2 t/f)

Sets the size of a 'sliding window' on past readings of the error from which the integral term will be calculated.

If 'addr' does not appear to reference a valid controller object or 'n1' is negative, returns FALSE and 'n2' is indeterminate. Otherwise returns TRUE.

An 'n1' of 0 implies 'let the software decide on a suitable buffer size', a positive 'n1' implies 'allocate a buffer of this size'. 'n2' is the actual size of the sliding window allocated. The default window size is 64.

SetSamplingTrigger

Usage: (addr1 addr2 --- t/f)

Only required where 'Run' is used (see below), to specify a user routine which controls whether it is time to run the control algorithm again. The user routine should take no parameters and should return a boolean on the stack (TRUE implies time to single-shot the control algorithm again). By default a routine is installed which always returns TRUE.

If 'addr2' appears to point to a valid controller object, sets 'addr1' as the execution address of the sampling trigger routine for the 'Run' routine, and returns TRUE; otherwise returns FALSE.

Stop

Usage: (addr --- t/f)

For use in multitasking environments. If 'addr' appears to reference a valid controller object and is currently running from a call to 'Run', signals the controller to terminate its run on the next scanning loop and returns TRUE. Otherwise returns FALSE.

ANNEXE A

Test Data

ANNEXE B

Source Listing