Governor's Suite

Table of contents

1. Overview

1.1. Overview

This document describes the installation, usage, and functionality of a TOPDOC Loadable Module **(TLM)** for version 4.x SoftPLC that is used in process control applications, with particular focus on hydro electric turbine governors. The TLM provides working code for a number of common control algorithms in ladder block form. Under license agreement the C source code is available and can be used as a starting point for extensions by integrators and end users.

SoftPLC can load one or more TLMs. TLMs may contain application specific extensions in the form of custom ladder instructions. The custom ladder instructions are called TLI's (TOPDOC Loadable Instructions). They appear as block type functions within the ladder program.

This TLM and build kit includes the following instructions. If you need one not listed, please ask.

- **Lead Lag** - and doubles as a 1st order filter if the Lead time constant is set to zero.
- **Ramp** - used to ramp up or down a float value or to integrate or accumulate a signal over time. It can be used to ramp up or ramp down a setpoint, or to integrate a signal over time. It can also be used as part of a feedback loop or to simulate a process component.
- **PID Super** - is an example of a supervisory/adaptive/auto-tuning agent that sits on top of a standard PID block and manages its operation. The standard PID block is the built-in ladder instruction that is standard with SoftPLC. This TLI is intended mostly as an example to show how to wrap your own logic around the standard PID block to create a governor type state driven controller. It is fully useful only when you have access to its C source code, which give structure definitions for the standard PID block and define new ones for the PID Super parameter data. It is a framework to allow you to extend these PID Super structures to your liking.
- **PID Super Load** - will read the parameters for the **PID Super** instruction in from a text based configuration disk file. This technique can be used to parameterize the **PID Super** instruction after you have copied the configuration text to disk. It is complementary to the **PID Super Save** instruction, but you can also use either a text editor or an FTP download to create the configuration parameter file.
- **PID Super Save** - will write the parameters for the **PID Super** instruction to a disk file. It can be used to save known working parameters.
- **Any Others** - that you write with the source code, or ask us to add.

2. Warranty

2.1. Terms of Use

Because of the variety of uses of the information described in this manual, the users of, and those responsible for applying this information must satisfy themselves as to the acceptability of each application and use of the information. In no event will SoftPLC Corporation be responsible or liable for its use, nor for any infringements of patents or other rights of third parties which may result from its use.

SOFTPLC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

SoftPLC Corporation reserves the right to change product specifications at any time without notice. No part of this document may be reproduced by any means, nor translated, nor transmitted to any magnetic medium without the written consent of SoftPLC Corporation.

SoftPLC, and TOPDOC are registered trademarks of SoftPLC Corporation.

Latest Printing: August, 2005

SoftPLC Corporation
25603 Red Brangus Drive
Spicewood, Texas 78669
USA Telephone: 1-800-SoftPLC
Fax: 512/264-8399
URL: http://softplc.com
Email: support@softplc.com

3. Installation

3.1. Installation

Login to your SoftPLC and check the `/SoftPLC/tlm` directory for the existence of `governor.tlm.so`. If it is not present, contact tech support and they can email you a copy. Then then ftp it to `/SoftPLC/tlm/governor.tlm.so`.

Then start TOPDOC NexGen and travers to "PLC | Modules". Click on the ***Use*** column in the row showing `GOVERNOR.TLM`. Then click ***Save***, then ***Send***.

The next step is to cycle power on your SoftPLC and the Governor's Suite TLM should be loaded. Check the syslog on the SoftPLC to verify that. Do this by logging in to SoftPLC and at the command prompt type:

```
# logread
```

4. Driver Initialization

4.1. Driver Initialization

TLM initialization is a special phase of operation that happens only when the TLM is first loaded. At time of TLM initialization, the SoftPLC runtime engine calls a TLM at a special entry point. This is the TLM's opportunity to do any system resource setup like allocating memory or reading disk configuration files. After these tasks are performed, the TLM returns control back to the SoftPLC runtime engine by simply returning from the C initialization function. More details are given in the **C/C++/Java Programmer's Toolkit** for SoftPLC 4.x, and the source code to this TLM.

In this TLM we do not use the initialization capability. The parameters for each instruction are assumed to be in the datatable. The PID Super Save and PID Super Load instructions read and write parameter data to and from disk respectively. So they are a more flexible approach which gives you programmatic control over when the configuration is read in.

5. TLI Ladder Instructions

5.1. TLIs (Ladder Instructions)

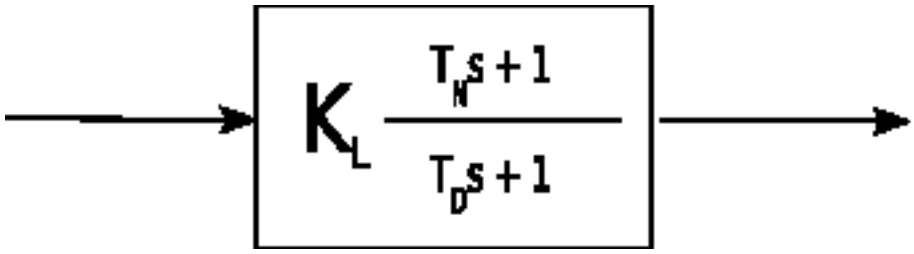
This section documents in more detail the operation of the included ladder function blocks, the **TLI**s. For TOPDOC online, these instructions are automatically added to the instruction menu the moment you go online to a SoftPLC that has the GOVERNOR TLM installed and loaded.

> **Note:**
> The rung logic preceding these instructions must be true for the instructions to operate.

Except for the **PID SUPER LOAD** and **PID SUPER SAVE** instructions, the remainder of these instructions are time based. Similar to our standard PID instruction, the time based instructions should be enabled only every so often. The interval between when they are enabled is called the **update time**. You can do this either by putting them all in a subroutine that is called only every update time or by conditionalizing them with a contact which is coming off of a free wheeling timer, whose preset is set to the update time.

5.1.1. LEAD LAG

This instruction takes an input signal in float form and transforms it according to the industry standard lead lag algorithm. The block diagram for it in Laplace transform representation is as follows:



where **KL** is the steady state proportional gain, **TN** is the lead time constant in seconds and **TD** is the lag constant in seconds.

> **Note:**
> You can also use this instruction as a first order lag simply by setting TN to zero.

It takes the following instruction parameters.

Parameter Name	Description
Input	is the input signal which is shown in the the above diagram coming in from the left side of the box.
Output	is the output signal which is shown in the the above diagram exiting to the right side of the box.
KL	is the steady state gain. It is often near 1.0.
TN	is lead time constant in seconds. As a memory aid, the "N" in TN stands for numerator.
TD	is the lag time constant in seconds. As a memory aid, the "D" in TD stands for denominator.
Delta T	is the update time in seconds. Set this to the interval period between calculations, which is every time the instruction is executed with a true rung condition.
Scratch	This is a block of float words that is used internally by the instruction to keep historical data between invocations.

5.1.2. RAMP

This instruction takes an input signal in float form and transforms it according to an integration or accumulation algorithm. The block diagram for it in Laplace transform representation is as follows:

It can be used to ramp up or ramp down a setpoint, or to integrate a signal over time. It can also be used as part of a feedback loop or to simulate a process component.

It takes the following parameters:

Parameter Name	Description
Input	is the input signal which is shown in the the above diagram coming in from the left side of the box.
Output	is the output signal which is shown in the the above diagram exiting to the right side of the box.
KI	is a gain or time constant. It establishes the rate of accumulation. If negative, the accumulated value will go down if the Input is positive.
Delta T	is the update time in seconds. Set this to the interval period between calculations, which is every time the instruction is executed with a true rung condition.
Scratch	This is a block of float words that is used internally by the instruction to keep historical data between invocations. The first word of this block is the Output on the last invocation.

5.1.3. PID SUPER

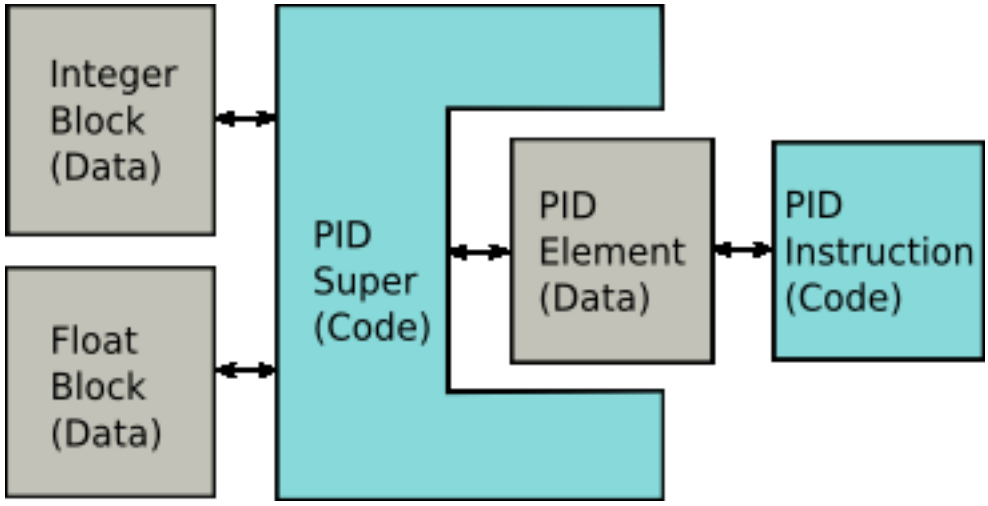
is a framework for a supervisory/adaptive/auto-tuning agent that sits on top of a standard PID controller and manages its operation. This instruction is a template for a C programmer to modify. It is probably only useful when the C source code is available for change. As structured, it takes an **integer block**, a **float block**, and **PID element** as parameters. A PID element is a standard *data structure* used by the standard PID instruction within SoftPLC. Together a PID element and a PID instruction make up a PID controller.

PID Super therefore has full access to all the data in the PID element including the update time, process output, PID gains, limits, etc. It may modify or examine any of these PID parameters depending on state data. You can have multiple sets of PID gains sitting in the float block and depending on which state you are entering, you can stuff the proper PID gains into the PID element upon transition in state. It can either rely on the standard PID instruction to execute after it does, or it can implement its own alternative PID algorithm. The demo code relies on the standard PID algorithm, and merely modifies the PID gains based on state.

In the sample code, the first integer in the integer block is expected to be set to a state

variable elsewhere in the control logic to a value between 0 and 4, inclusive. Depending on which state is transitioned to, one of 5 separate gain sets are stuffed into the PID block.

Conceptually it looks like this:



In the C source code the integer block and the float block are actually two separate *C structures* with each field having a convenient name. You are free to modify these C structure declarations to match your needs. The **PID SUPER SAVE** instruction is used to save these 2 structures to disk, and the **PID SUPER LOAD** instruction is used to load these 2 structures in from disk. If you modify the C source code for either struct mapped to the integer block or float block in **PID SUPER**, then you may also have to modify the two PID SUPER LOAD/SAVE instructions as well, or comment them out if you do not need them. Because the two blocks reside in the SoftPLC datatable, they can also be set via an HMI, TOPDOC NexGen, or a Servlet running under the Webserver.

This instruction takes only 3 parameters, but all 3 parameters are structures and therefore this instruction can operate on nearly any amount of data that you choose. Notice that there is no Delta T required, that is because you can reference the one in the PID element if you need one.

Parameter Name	Description
PID Elem	is the PID Element that is managed.
Integers	is a block of up to 10,000 integers, with the actual size depending on the C source code.
Floats	is a block of up to 10,000 floats, with the actual

	size depending on the C source code.

5.1.4. PID SUPER SAVE

This instruction will write the block parameters for the **PID SUPER** instruction to a disk file. It can be used to save known working parameters. See the description in PID SUPER for more information. The save takes place when the instruction is energized.

Parameter Name	Description
Integers	is a block of up to 10,000 integers, with the actual size depending on the C source code.
Floats	is a block of up to 10,000 floats, with the actual size depending on the C source code.
Filename	a STRING element which holds the name of the file to write to.
ErrorTxt	a STRING element which gets an error message if failure writing to disk, say because the requested directory does not exist. This string will be set to length zero if the write succeeds.

5.1.5. PID SUPER LOAD

This instruction will read the block parameters for the **PID SUPER** instruction from a disk file. It can be used to load known working parameters. See the description in PID SUPER for more information. The load takes place when the instruction is energized. The format of the file is all ASCII with `name=value` pairs on each line. So it may be edited with a text editor to parameterize a governor controller.

Parameter Name	Description
Integers	is a block of up to 10,000 integers, with the actual size depending on the C source code.
Floats	is a block of up to 10,000 floats, with the actual size depending on the C source code.
Filename	a STRING element which holds the name of the file to read from.
ErrorTxt	a STRING element which gets an error message if failure either reading the disk file or parsing its contents. This string will be set to length zero if

	the write succeeds.

6. How to Build

6.1. How to Build the TLM

This is quite easy because the provided tar file has a Makefile in it which drives the C compiler. However a longer write up is still in progress. You will need a copy of Linux for x86 to build the TLM. If you are not already a Linux x86 user, then the easiest thing is to download the LIVE CD version of UBUNTU and then just run the development system off of the CD. You do not even have to install it.

More to come on this later.

7. All