Flexible Servo Controller

Highly configurable general-purpose brushed DC servo-motor controller.

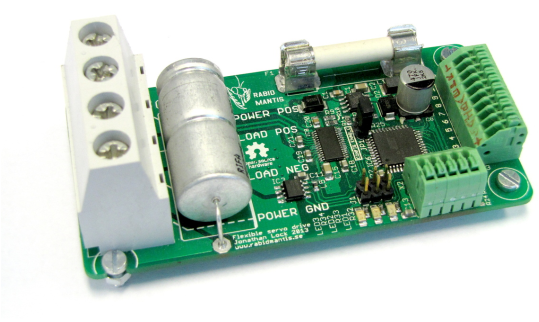
Features

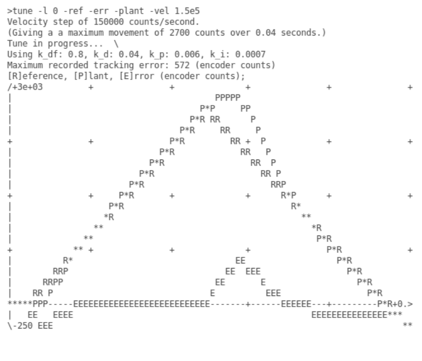
Up to 25 A continuous output (10 A continuous / 25 A peak without active cooling).
Up to 42 V supply voltage (50 V absolute maximum).
Protected from output short circuit, short to ground, short to supply.
Fully electrically isolated input/output (up to 560 V), 2.7 to 5.5 V logic compatible.
Configurable charge pump or active-low enable input.
Up to 1 MHz motor encoder and input setpoint rate.
Adjustable 1-64 input step multiplier.
5 V/3.3 V, 200 mA supply for encoder feedback.
Settings configured over a UART text-based interface.
ASCII-art-based controller tuning.
Major faults logged to nonvolatile memory.

Description

Flexible Servo Controller is, as its name suggests, a multi-use servo controller for brushed DC motors with incremental encoder feedback capable of driving motor currents of up to 25 A continuous (10 A continuous without active cooling) at supply voltages of up to 42 V (limited by an absolute maximum of 50 V). The input position setpoint uses an encoder-like quadrature (A/B) interface and supports speeds up to 1 MHz. A configurable charge pump or active-low input activates the controller, bringing the output out of a configurable off state (brake or coast). Logic outputs are a controller active output, a fault output, and an encoder index output. A serial UART interface (RX/TX) allows for configuring and reading the system state with a command line interface which offers easy access to read and modify the system configuration, tune the controller parameters, read any logged system faults, and save settings to nonvolatile memory.
A programmer capable of programming devices over the PDI interface (such as an AVRISP MKII) and a serial terminal, such as a USB to UART converter (3.3/5 V compatible) are required to program and configure Flexible Servo Controller.

Overview





```
>tune -l 0 -ref -err -plant -vel 1.5e5
Velocity step of 150000 counts/second.
(Giving a a maximum movement of 2700 counts over 0.04 seconds.)
Tune in progress...  \
Using k_df: 0.8, k_d: 0.04, k_p: 0.006, k_i: 0.0007
Maximum recorded tracking error: 572 (encoder counts)
[R]eference, [P]lant, [E]rror (encoder counts);
/+3e+03         +               +              +               +               +
|                                        PPPPP
|                                     P*P     PP
|                                   P*R RR      P
|                                 P*R     RR     P
+               +               P*R         RR +  P            +               +
|                             P*R             RR   P
|                           P*R                 RR  P
|                         P*R                     RR P
|                       P*R                         RRP
+               +     P*R       +              +      R*P      +               +
|                   P*R                                 R*
|                  *R                                     **
|                **                                         *R
|              **                                            P*R
+            ** +               +              +               P*R             +
|          R*                                E                   P*R
|        RRP                               EE  EEE                 P*R
|      RRPP                              EE       E                  P*R
|    RR P                               E          EEE                 P*R
*****PPP-----EEEEEEEEEEEEEEEEEEEEEEEEEEE-------+------EEEEEE---+---------P*R+0.>
|   EE   EEEE                                               EEEEEEEEEEEEEEE***
\-250 EEE                                                                     **
```

Command line interface and ASCII-art graphical velocity step response plot.

Contents

Assembly 3

Pin Description 7

Initial Startup 8

Software Interface 10

Usage 19

Bill of materials 28

Mechanical Description 31

Electrical Characteristics 33

Typical Performance 36

Operation 39

Copyright Notice

Hardware Revision History

1.0.0 Development version (shown in photograph on cover page).

1.1.0 Initial public version. Input fuse changed, encoder output supply fuse added.

1.2.0 Improved ESD resiliance; ESD diodes replaced and series input resistors added. **NOTE! Component designators reset.**

Software Revision History

1.0.0 Initial version.

1.1.0 Applies to hardware revision 1.2.0, only relevant sections in the sections Bill of materials, Electrical Characteristics, and Operation modified.

Documentation Revision History

1.0.0 Initial version, applies to hardware revision 1.1.0.

Absolute Maximum Ratings

PARAMETER	SYMBOL	RATING
Power positive input voltage, relative to power ground	$V_{sup,pwr}$	-1.5 V to +50 V[a]
Isolated logic input/output supply voltage	V_{IO}	-0.5 V to +7 V
Isolated logic input voltage[b]	$V_{I,IO}$	-0.5 V to V_{IO}+0.5 V
Average output current per logic output	$I_{logic,out}$	-11 mA to +11 mA
Common-Mode Transients[c]		-100 kV/µs to +100 kV/µs
Ambient operating temperature		See section Usage.
Encoder supply output short circuit		Continuous
Encoder logic input voltage range		-0.5 V to +7.0 V

All logic inputs are protected with ESD-suppression diodes and are expected to tolerate ESD up to 10 kV with a human body model (MIL-STD-883) and 30 kV with a contact discharge model (IEC 61000-4-2), however ***this remains untested!***

[a]Will often survive a reverse-polarity state by destroying the input protection fuse. Survivability rates depend on the current capability of the source and the fuse selected, but are generally very favorable for fast-acting fuses below 30 A.

[b]This specification is only applicable when the digital isolators are mounted. See section Assembly for limits when operating in an unisolated mode.

[c]Refers to common-mode transients across the isolation barrier between the isolated logic inputs and power stage.

Assembly

Flexible Servo Controller consists primarily of surface mount components with a few through-hole components — primarily contacts and large components. The most difficult component to mount is the A3941 H-bridge driver, which packaged in a TSSOP-28 carrier with exposed thermal pad, requiring a reflow oven, hotplate, or hot air reflow workstation to correctly solder[1]. All remaining components can be soldered with a fine-point soldering iron, and consist of various TSSOP, TQFP, SO8, 0603 and larger packages. Components are mounted on both sides of the PCB, it is recommended to mount the components on the bottom first if soldering in a reflow oven, while if soldering on a hotplate the top side (containing the A3941 H-bridge driver) must be soldered first. No glue is required if using a reflow oven for soldering the components, the components on the bottom of the PCB are lightweight enough to remain attached solely due to the solder's surface tension.

The rest of this section will describe component mounting choices and how to upload the binary files to the microprocessor. See section Bill of materials for the components used, figure 25 for the placement of the components, section Pin Description for what to connect to the pin-headers, section Initial Startup for how to communicate with the device for the first time, sections Usage and Software Interface for how to to use and

[1]This can be ghetto-soldered with an ordinary skillet on a stove, information on which can easily be found by searching for *solder hot plate skillet* or the like.

configure Flexible Servo Controller for the final application, and section Operation for a more detailed description of how the entire device functions, both from an electrical and software perspective.

Assembly choices

Depending on the way Flexible Servo Controller is intended to be used some components may be left unmounted and/or mounted in different ways. A fully exhaustive list of all possible assembly choices follows;

Ring terminals can be used instead of mounting the relatively costly high-current connector (X1). Simply leave X1 unmounted and use ring terminals with a center bore diameter > 4 mm and an outer diameter < 9 mm for the input power and motor connections, and fasten them with M4 machine screws with a spring washer.

Different encoder types are supported; encoders with open-collector, open-drain, and push/pull digital outputs are supported with the default bias resistor configuration (R16, R23, R32 mounted, R19, R27, R38 not mounted), while encoders with open-emitter or open-source outputs should instead use the alternate bias resistor configuration of R19, R27, R38 mounted and R16, R23, R32 left unmounted. **Note that operation at high speeds with an encoder with open-collector/drain/emitter/source outputs requires short cabling!** Cable capacitance must be kept far below $C_{cable} \ll \frac{1}{2\pi 10 \cdot 10^{-3} f_{max}}$ where f_{max} is the maximum expected output frequency from the encoder. Encoders with a push/pull output stage do not exhibit this issue. Lowering the bias resistors (down to a minimum value of approximately 470Ω) will improve the performance at high speeds, allowing a cable capacitance of $C_{cable} \ll \frac{1}{2\pi R_{bias} f_{max}}$.

Low output current applications can use less expensive components in the high-power circuitry. C4, Q1, Q2, Q3, and Q4 can be replaced with low-cost components when reduced output current is acceptable. Choose a capacitor and N-channel MOSFET transistors that fulfill the application-specific requirements, and ensure that the `i_max` parameter (as described in section Software Interface) and the selected fuse F1 is set to a safe value for the components chosen . Should the nominal drain-source voltage ($V_{DS,ON}$) over the selected MOSFET exceed 0.1 V at peak output current then the values chosen for R22 and/or R24 must be modified, see the Allegro A3941 datasheet for how to determine the correct value(s). Though Flexible Servo Controller will function with nearly any input capacitor (C3) and N-channel MOSFET (Q1, Q2, Q3, Q4) combination, it is generally a good idea to choose an input capacitor $\geq 100\,\mu$F with a reasonable ripple current specification, and N-channel MOSFET's with a total gate charge of $\leq 250\,$nC.

SJ1 and SJ2 allow for supplying 3.3 V to the encoder output (X2) instead of the default 5 V output. To enable the 3.3 V output, cut the trace through SJ1 with a scalpel or other small knife, short SJ2 with solder, and replace F2 with the alternate fuse listed

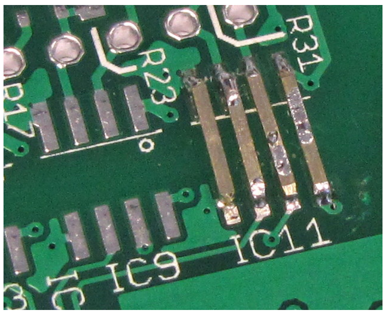


Figure 1: For unisolated mode, connect the opposite pads of for IC4, IC5, IC9, and IC11 to another. In this image IC11 has been replaced, while the other 3 IC's remain.

in section Bill of materials. **Note; If F2 is not replaced with the alternate fuse Flexible Servo Controller may be damaged if excessive current is drawn when configured for 3.3 V output mode**!

Unisolated I/O allows for reducing the total component cost for Flexible Servo Controller, as the digital isolators (IC4, IC5, IC9, and IC11) are not strictly required for operation. For unisolated operation leave the isolators unmounted and connect pads 1 to 8, 2 to 7, 3 to 6, and 4 to 5 using a conductor as shown in figure 1. Additionally, C14, C27, C28, D3, D5, and D6 may be left unmounted when in unisolated mode. **Note that when assembled in unisolated mode the logic inputs are directly connected to the microprocessor I/O pins, which are *NOT* 5 V-tolerant and far more ESD susceptible than the ordinary inputs!** Additionally, all input voltages must lie in the more restrictive range of -0.5 V to $V_{sup,IO,3v3} + 0.5\,\mathrm{V}$. Note that this implies that when Flexible Servo Controller is unpowered the unisolated logic inputs may not be driven to a logic high level! When operating in unisolated mode connector X3 pin 1 can supply power to external logic (see section Electrical Characteristics). Additionally, when operating in unisolated mode keep in mind the I/O ground and power ground are connected, leading to the potential for ground loops. **Due to the numerous potential pitfalls with operation in unisolated mode this mode is only recommended for applications where a small daughterboard is attached to the plated-through holes for X3, such as a hypothetical step/direction to quadrature converter, powered by the 3.3 V supply generated from Flexible Servo Controller and with its own galvanically isolated and ESD-protected inputs.**

Programming the microprocessor

Programming the microprocessor is easily performed when mounted on the PCB. Only the input power terminals on X1 and the programming interface on J1 need to be connected in order to program the device; all other connectors can be left unconnected. Supply 9-40 V to POWER_POS and POWER_GND as described in Pin Description (this will draw a relatively low current; $I_{Q,inactive}$ — which is typically 15-40 mA with decreasing current at higher supply voltages). Ensure a programmer capable of programming over the PDI interface (such as the AVRISPMKII) is connected to J1, after which the firmware should be able to be uploaded. Depending on the platform and programmer used the actual commands to program the device vary. If using Atmel Studio the correct settings should be automatically loaded when opening the project, if not, or if another software package is used, the important actions to perform are uploading the .hex file containing the firmware and specifying the fuse bytes below as follows;

Fusebyte 1 0x06 *(WDWP = 8 CLK, WDP = 512CLK)*

Fusebyte 2 0xFE *(BOOTRST = APPLICATION, TOSCSEL = XTAL, BODPD = CONTINUOUS)*

Fusebyte 4 0xE1 *(RSTDISBL = 1, SUT = 64MS, WDLOCK = 1)*

Fusebyte 5 0xE9 *(BODACT = CONTINUOUS, EESAVE = 0, BODLVL = 2V8)*

When successfully programmed and powered up one of the status LED's should either blink or shine steadily.

(Note that it is not necessary to write the EEPROM data, as the default values will be loaded from flash memory on the initial startup).

Pin Description

HEADER	PIN	DESCRIPTION
X1	POWER_GND	Primary high-current power ground connection.
	POWER_POS	Primary high-current power positive connection.
	LOAD_POS	Positive motor output connection.
	LOAD_NEG	Negative motor output connection.
X2	V+	Incremental encoder 5 V/3.3 V power output.
	GND	Incremental encoder ground connection.
	IN_A	Incremental encoder A-phase input. If using a differential encoder leave $\bar{A}$ disconnected.
	IN_B	Incremental encoder B-phase input. If using a differential encoder leave $\bar{B}$ disconnected.
	IN_Z	Incremental encoder Z (index) input. If using a differential encoder leave $\bar{Z}$ disconnected.
X3	V+	Isolated logic supply voltage input[2].
	GND	Isolated logic supply ground.
	IN_A	A-phase quadrature reference position input.
	OUT_Z	Encoder Z (index) output.
	IN_B	B-phase quadrature reference position input.
	OUT_ACT	Device active output (active high). Active when motor is actively controlled.
	CP/$\bar{E}$	Charge pump / enable (active low) input. Servo drive is enabled when this input is active.
	$\bar{F}$	Device fault output (active-low). Active when a major fault has occurred or a logged error has not been cleared.
	RX	UART serial interface input (receive).
	TX	UART serial interface output (transmit).
J1	N/A	PDI programming interface.
JP1	N/A	Short on startup to enable serial input, leave open to disable.

[2]Only applies when assembled for isolated mode. In unisolated mode this is an output with specifications as described in section Electrical Characteristics.

Initial Startup

In order to configure and verify that Flexible Servo Controller functions correctly, assemble and program the device as per section Assembly and, at the bare minimum, **attach a jumper to JP1 (shorting the pins)** and connect a UART serial terminal (such as a USB to UART converter) to X3, following the pin description in Pin Description. Keep in mind that **the serial adapter must supply the digital isolators with logic power and ground** (nominally 2.7 V to 5.5 V, typically <4 mA, see section Electrical Characteristics for more details)! Nearly any serial terminal application on the host PC will function, such as PuTTY, Minicom, or CuteCom; all of which are freely available and function on most platforms (Linux, Mac, Windows). Set up the terminal application for communication at 57600 bps, with software flow control, 8 data bits, no parity, and 1 stop bit (57600 8N1)[3]. If all goes well, a startup message similar to that shown in figure 2 should be displayed when Flexible Servo Controller is powered up.[4]

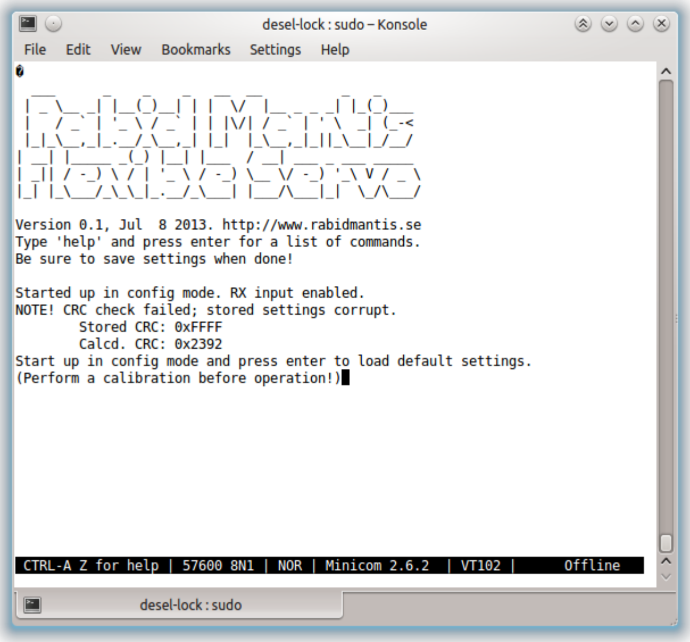
When serial communications have been established, press the enter key (as prompted) to load the default settings to nonvolatile memory, afterwards run the calibration command and save the results from the self-calibration to nonvolatile memory to finish the initial startup. These steps are performed by entering

`<enter>`	*(press the enter key to acknowledge that the default settings will be loaded)*
`calibrate<enter>`	*(to call the self-calibration function)*
`<enter>`	*(press when the motor shaft is verified to be stationary if motor is attached)*
`save<enter>`	*(to save the calibration results to nonvolatile memory)*

on the serial interface. The final result of this should be similar to that shown in figure 3. Flexible Servo Controller is now ready to be tuned and tested with the output motor; see section Software Interface for a description of the terminal interface and the commands available and section Usage for how to tune and test Flexible Servo Controller.

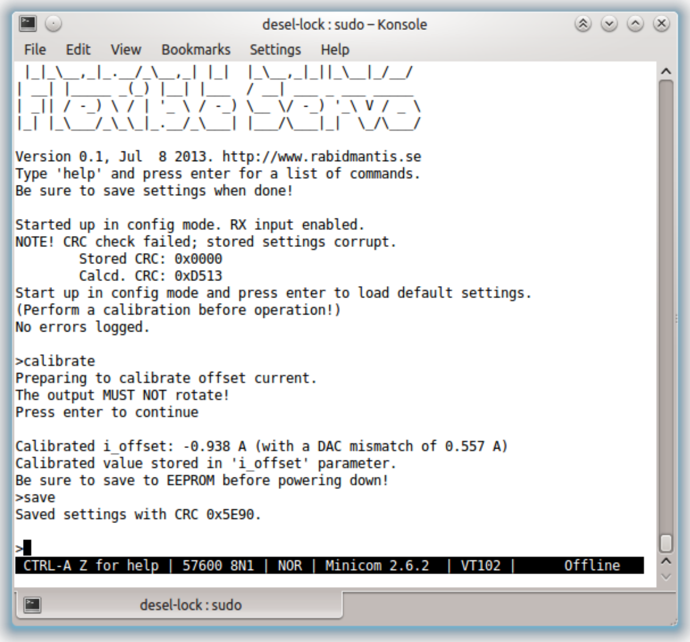
[3]Any of the LF, CR, and CR/LF line-feed modes are supported and will behave identically.

[4]Should only garbled text or invalid characters appear in the terminal application when the correct bit-rate is chosen then verify that Flexible Servo Controller is compiled for the specified bit rate, by checking the UART speed definitions in the `config.h` file in the source directory; for a bit-rate of 57600 bps `UART_BSEL` should be defined to 135 and `UART_BSCALE` should be defined to -2. In the event that nothing is output at all when the correct serial terminal is chosen in the PC application then verify that Flexible Servo Controller outputs data by measuring the TX output with an oscilloscope; on startup there should be a relatively large amount of data output followed by a steady logic high level.


desel-lock : sudo – Konsole
File Edit View Bookmarks Settings Help
desel-lock : sudo

```
 ___      _    _    _ __  __           _   _    
| _ \__ _| |__(_)__| |  \/  |__ _ _ _ | |_(_)___
|   / _` | '_ \ / _` | |\/| / _` | ' \|  _| (_-<
|_|_\__,_|_.__/_\__,_|_|  |_\__,_|_||_|\__|_/__/
 ___ _         _ _    _       ___                
| __| |_____ _(_) |__| |___  / __| ___ _ ___ _____
| _|| / -_) \ / | '_ \ / -_) \__ \/ -_) '_\ V / _ \
|_| |_\___/_\_\_|_.__/_\___| |___/\___|_|  \_/\___/

Version 0.1, Jul  8 2013. http://www.rabidmantis.se
Type 'help' and press enter for a list of commands.
Be sure to save settings when done!

Started up in config mode. RX input enabled.
NOTE! CRC check failed; stored settings corrupt.
        Stored CRC: 0xFFFF
        Calcd. CRC: 0x2392
Start up in config mode and press enter to load default settings.
(Perform a calibration before operation!)




CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

Figure 2: Output on initial startup.


desel-lock : sudo – Konsole
File Edit View Bookmarks Settings Help
desel-lock : sudo

```
|_|_\__,_|_.__/_\__,_|_|  |_\__,_|_||_|\__|_/__/
 ___ _         _ _    _       ___                
| __| |_____ _(_) |__| |___  / __| ___ _ ___ _____
| _|| / -_) \ / | '_ \ / -_) \__ \/ -_) '_\ V / _ \
|_| |_\___/_\_\_|_.__/_\___| |___/\___|_|  \_/\___/

Version 0.1, Jul  8 2013. http://www.rabidmantis.se
Type 'help' and press enter for a list of commands.
Be sure to save settings when done!

Started up in config mode. RX input enabled.
NOTE! CRC check failed; stored settings corrupt.
        Stored CRC: 0x0000
        Calcd. CRC: 0xD513
Start up in config mode and press enter to load default settings.
(Perform a calibration before operation!)
No errors logged.

>calibrate
Preparing to calibrate offset current.
The output MUST NOT rotate!
Press enter to continue

Calibrated i_offset: -0.938 A (with a DAC mismatch of 0.557 A)
Calibrated value stored in 'i_offset' parameter.
Be sure to save to EEPROM before powering down!
>save
Saved settings with CRC 0x5E90.

>
CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

Figure 3: Output after self-calibration and saving to nonvolatile memory.

Software Interface

Flexible Servo Controller offers a command-line interface (CLI) for configuring, tuning, and reading system parameters. In order to use the CLI the device must be assembled and programmed as described in section Assembly, after which the steps described in section Initial Startup must be performed. At this stage the serial communications with Flexible Servo Controller should be established and seen to be working.

Command-line interface

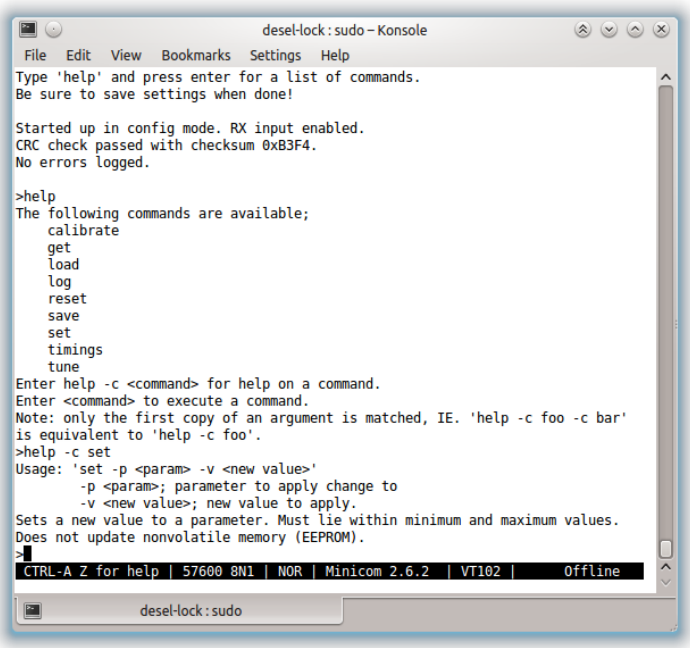
A brief introduction to the syntax used by the CLI follows, which should be very familiar to any user of traditional PC-based CLI's.

All commands sent to Flexible Servo Controller are line-based, and typically of the format[5] `command -argument1 value1 <newline>`, where <newline> is typically generated by pressing the enter key on the keyboard and is implied (not shown) in the following examples. Some commands may expect several arguments and values, while others need none. For example the command to save all settings to nonvolatile memory is simply `save`, while the command to set a parameter to a value is `set -p <param> -v <value>`. The order of arguments does not matter, meaning that the commands `set -p <param> -v <value>` and `set -v <value> -p <param>` are functionally identical. Additionally, only the first appearance of an argument is used while unused arguments are ignored, meaning that the commands `set -p <param> -v <value>` and `set -p <param> -v <value> -anotherarg 3 -p <anotherparam>` are also functionally identical.

A list of all valid commands can be generated by calling the `help` command, while details for each individual command can be displayed by calling `help -c <command>`, as is shown in figure 4. Note that parameters shown in `<...>` are required, parameters enclosed within square braces and separated by pipes require one of the parameters, IE. `[ -foo <...> | -bar <...>]` requires either `foo` or `bar` to be specified, while parameters enclosed within parenthesis are optional, IE. `(-foo)` means that `foo` is optional and not required.

Flexible Servo Controller can be easily used with both character-based terminal applications (like PuTTY and Minicom) and line-based terminal applications (like CuteCom), as all valid input is directly echoed to the terminal and a rudimentary input-buffer is implemented, **allowing the use of the backspace key to remove characters and the up/down arrow keys to recall previous input lines.**

[5]Flexible Servo Controller's CLI is completely text based and will only accept input characters in the ASCII range of 0x20 to 0x127 (0-9, a-z, A-Z, and all other printable ASCII characters such as !, ", and %), 0x08 (backspace), 0x0B (new line), 0x0D (carriage return), 0x1B (escape character) followed by "[A" (up arrow on keyboard), and 0x1B (escape character) followed by "[B" (down arrow on keyboard). All other inputs are regarded as invalid and will result in the output of a 0x07 character (system bell) to the host PC and are otherwise ignored.


desel-lock : sudo – Konsole
File Edit View Bookmarks Settings Help
desel-lock : sudo

```
Type 'help' and press enter for a list of commands.
Be sure to save settings when done!

Started up in config mode. RX input enabled.
CRC check passed with checksum 0xB3F4.
No errors logged.

>help
The following commands are available;
    calibrate
    get
    load
    log
    reset
    save
    set
    timings
    tune
Enter help -c <command> for help on a command.
Enter <command> to execute a command.
Note: only the first copy of an argument is matched, IE. 'help -c foo -c bar'
is equivalent to 'help -c foo'.
>help -c set
Usage: 'set -p <param> -v <new value>'
        -p <param>; parameter to apply change to
        -v <new value>; new value to apply.
Sets a new value to a parameter. Must lie within minimum and maximum values.
Does not update nonvolatile memory (EEPROM).
>
 CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

Figure 4: The `help` command displays all valid commands, while `help -c <command>` shows more in-depth information on each command; in this example it is shown that the `set` command expects two arguments, `-p <param>` to choose a parameter to set, and `-v <value>` for the value to set.

Command summary

Each command has a brief summary of its function that can be accessed by inputing `help -c <command>`, which is expanded on in more depth below.

calibrate Calibrates the analog circuitry in the device, and need only be run on the first startup or when loading default parameter values from nonvolatile memory. If a motor is attached the shaft must be kept stationary before calling. Called with `calibrate` (no arguments).

get Gets (displays) the current state of one or all system parameters; name, type, value, minimum value, and maximum value. Called with `get [-a | -p <name>]`.

load Loads the saved parameter values from non-volatile memory and overwrites the current values. This is functionally similar to power-cycling (turning off and turning on) the device. Can optionally load the compile-time (factory) default settings with the `-default` flag. Called with `load (-default)`.

log Shows any logged system errors/faults in order of occurrence, and can optionally clear them with the `-clear` flag. Called with `log (-clear)`.

reset Resets the device, which is functionally identical to power-cycling (turning off and turning on) the device. Note that all unsaved parameter values will be lost. Called with `reset` (no arguments).

save Saves all current parameter values to nonvolatile memory. This is required to maintain parameter values across power-cycling (when the device is turned off and turned on). Called with `save` (no arguments).

set Updates a parameter to a new value, which immediately takes effect. Called with `set -p <param> -v <value>`.

timings Displays fixed (compile-time) timings for the different controllers, calculations, and the charge-pump input. Called with `timings` (no arguments).

tune Starts a tuning run of the motor position (PID) controller (see section Usage for a suggested method for tuning the controller gains). All tuning runs apply a change to the position setpoint and record the resulting motion for either display in an ASCII-art plot with adjustable content or CSV-type output to the CLI.
Three tuning types are available; `-pos`, a position step, where the setpoint position is switched between two constant values; `-vel`, a velocity step, where the setpoint velocity is switched between two constant values (giving a ramp in position); and finally `-acc`, an acceleration step, where the acceleration is switched between two constant values (giving a quadratic change in position). These three different types of tuning runs are useful in different stages of the tuning process, which is expanded on in section Usage.

The length of the tuning run can be set with `-l`, allowing for viewing either transient or slow effects. Keep in mind that regardless of the tuning length only 80 data points are saved, making it likely to miss fast events when running with a long tuning run. Additionally, the velocity and acceleration steps apply a zero-order-hold output between the 80 data points, making long runs with position and acceleration steps relatively noisy and stressful for the connected hardware for fast servo systems. **Only the position step is judged to be useful for long tuning runs with fast servo systems.**
A total of seven different plot traces can be displayed in any configuration; `-ref` shows the reference setpoint, `-plant` shows the plant (motor) position, while `-err` shows the position error (plant position subtracted by reference position), which are all expressed in encoder counts. Additionally, `-t` plots the total output current, while `-d`, `-p`, and `-i` show the individual contribution from the derivative, proportional, and integral terms respectively.
The plot height can be changed from the default value with `-h`, and the output mode can be switched to a CSV-type (comma separated value) output with the `-csv` argument.
This command is called with `tune [-pos <val> | -vel <val> | -acc <val>] -l <length> [-ref | -plant | -err | -d | -p | -i | -t] (-h <height>) (-csv)`.

Parameter description

The following parameters allow for configuring Flexible Servo Controller for different applications and motor sizes. All user-configurable parameters can be shown and modified using the **get** and **set** commands respectively (as listed above). See section Usage for a description of how to determine the values for each parameter.

There are two fundamental parameter types; real and integer. Real parameters are parameters represented by real numbers[6] (numbers with a decimal point), such as 1.5, -0.001, or 100. Integer numbers are whole numbers (numbers without a decimal point) such as 2, -5, or 200. All parameters have a defined type, as well as a maximum and minimum value. Attempting to assign a value outside the bounds of the allowed range will result in an error and will not change the parameter. **Keep in mind that the maximum and minimum parameter values are not safe bounds, but rather the limits of what Flexible Servo Controller is capable of achieving and may be beyond the limits of the connected motor!**

All parameters are guaranteed to be atomically changed; a parameter may be changed at any time and will immediately take effect, and at no point will an intermediate value be used.

Note that real numbers may be specified (and are sometimes output) using scientific notation, where a number is expressed in the form $a \cdot 10^b$, such as $2 \cdot 10^{-3}$, which is equivalent to 0.002. Flexible Servo Controller uses the 'e' character to separate the coefficient a from the exponent b. For example, the number `5.2e-3` ($5.2 \cdot 10^{-3}$) is equivalent to 0.0052. Integer numbers may not be specified (and are never output) using scientific

[6]These are internally represented with IEEE 754 32-bit floating point numbers.

i_skip	$f_{SW,max}$ [kHz]	i_skip	$f_{SW,max}$ [kHz]	i_skip	$f_{SW,max}$ [kHz]
0	*100*	5	16.7	10	9.09
1	*50*	6	14.3	14	6.67
2	*33.3*	7	12.5	18	5.26
3	25	8	11.1	30	3.23
4	20	9	10	100	0.99

Note; values of i_skip below 3 are generally unneeded and primarily cause heating in the H-bridge driver!

Table 2: Maximum output switching frequency for various values of **i_skip**.

notation.

i_max A real value, sets the maximum current from the motor position (PID) controller and the maximum braking current expressed in Amperes. This parameter can be used as a safety to limit the output current for low-power motors or to reduce the maximum motor torque. Both when in active (position-control) mode and inactive (brake) mode the motor current will be kept within ±i_max.

i_ripple A real value, sets the nominal ripple current in the motor when in active (position-control) mode expressed in Amperes. This parameter is used together with `i_skip` to limit the maximum switching frequency of Flexible Servo Controller. This parameter may be set to zero to maintain as low a ripple current as possible. A safe default value for most applications is zero.

i_skip An integer value which limits the maximum switching frequency of Flexible Servo Controller, expressed as a divisor of the current controller frequency. With a current controller frequency of 200 kHz (which can be verified with the `timings` command) the maximum switching frequency for some values of `i_skip` is shown in table 2, and can be determined by $\frac{f_{CC}}{2(1+i\_skip)}$ where f_{CC} is the current controller frequency. Keep in mind this parameter only defines the maximum switching frequency, due to the nature of the current controller (as explained in the section Operation) the actual switching frequency varies greatly and is often below this! Note; this parameter by far contributes to most of the heating in the h-bridge and should not be set unnecessarily low. A safe default value for most applications is three.

i_offset A real value which stores an offset current as calculated from the internal self-calibration, expressed in Amperes. Not directly intended for direct modification, the value for this parameter should be determined by running the calibrate command.

i_friction A real value which allows compensating for a certain degree of dynamic friction in the mechanical system, expressed in Amperes. The total output current from Flexible Servo Controller driven through the motor is $I_{tot} = I_{PID} + \operatorname{sgn}(I_{PID}) \cdot I_{friction}$, or equivalently, the output current I_{tot} has an added term $I_{friction}$ that is added to the output and of the same sign as the motor position (PID) controller.

k_df	f_c [Hz]	**k_df**	f_c [Hz]	**k_df**	f_c [Hz]	**k_df**	f_c [Hz]
0.99	*2000*	0.7	742	0.4	212	0.1	35
0.86	1955	0.6	477	0.3	136	0.05	17
0.8	1273	0.5	318	0.2	80	0.01	3.2

*Note; for **k_df** values above 0.86 the there is essentially no effect from the derivative filter, and the bandwidth is limited only by the PID controller sample rate of 2000 Hz.*

Table 3: Motor position (PID) controller derivative cutoff frequencies for various value of **k_df**.

Large values will cause the output to oscillate and will greatly reduce the performance of Flexible Servo Controller. A value of zero is a safe default value.

k_d A real value which sets the motor position (PID) controller derivative gain.

k_df A real value which sets the motor position (PID) controller derivative filtering coefficient. Larger values will filter (smooth) the derivative gain less, while smaller values will filter (smooth) the derivative gain more. See table 3 for a list of cutoff frequencies for varying values of `k_df`. The derivative filter is implemented as a first-order (single-pole) low-pass filter, with a cutoff frequency of $f_c = \frac{f_{pid}}{2\pi\left(\frac{1-k\_df}{k\_df}\right)}$. A value of 0.86 is a reasonable default for systems where the fastest response is desired, while smaller values are useful for systems where an intentionally slower response is desired or higher derivative gains are needed.

k_p A real value which sets the motor position (PID) controller proportional gain coefficient.

k_i A real value which sets the motor position (PID) controller integral gain coefficient.

trk_err An integer value which sets the maximum permissible tracking error, expressed in motor encoder counts. When the difference between the commanded position and the measured position exceeds this value Flexible Servo Controller disables the output enters the inactive state.

v_min A real value which sets the minimum bus voltage, expressed in volts. When the measured bus voltage falls below this value the output is disabled and Flexible Servo Controller enters the inactive state. The output can be re-enabled when the bus voltage rises above `v_min` + 2 V.

v_max A real value which sets the maximum bus voltage, expressed in volts. If the measured bus voltage exceeds this value the output is disabled and brought into a coasting state, Flexible Servo Controller enters the inactive state, and a critical error is logged and saved. See section Usage for more on the motor coasting state.

The output can be re-enabled when the bus voltage falls below `v_max` - 2 V. It is strongly reccomended to set this parameter to at most 45 V, and if possible even less.

cpump_en An integer value which enables or disables the input enable charge pump. If set to one the charge pump is enabled (requiring an alternating input to enable Flexible Servo Controller), while if set to zero the charge pump is disabled (allowing Flexible Servo Controller to be enabled by a logic low level). See section Usage for more on the charge pump.

brake_en An integer value which enables or disables the motor brake, controlling the system behavior when in the inactive mode. If set to one the brake is enabled, while if set to zero the brake is disabled. The motor brake will short the motor terminals when Flexible Servo Controller is in inactive mode, slowing the motor very quickly. If disabled the motor will coast when Flexible Servo Controller is in inactive mode, allowing the motor to rotate and slow down over a longer period. The maximum motor current specified in the `i_max` parameter is respected, making the brake safe to use even with large loads.

high_i_en An integer value which enables or disables the high current mode. If set to one the high-current mode is enabled, while if set to zero the high-current mode is disabled. The high-current mode allows for operation at the limits of what Flexible Servo Controller is capable of, and should be enabled when `i_max` + `i_ripple` is over 20, or if ERR4 is displayed in normal operation. Extra care should be taken when this mode is enabled, as operation in this mode has fewer safety lockouts and has the risk of causing Flexible Servo Controller to overheat and potentially damage itself and/or the connected motor. Unless high current mode is specifically needed, this value should be set to zero.

inp_pow An integer value which multiples the input setpoint steps. The parameter values of 0, 1, 2, 3, 4, 5, and 6 multiply the input by 1, 2, 4, 8, 16, 32, and 64 steps respectively. This parameter is primarily intended for applications with a high-speed motor and high-resolution encoder, where it may otherwise be difficult to generate input steps quickly enough. Keep in mind that the larger multipliers effectively reduce the positioning resolution. Additionally, keep in mind that the maximum permissible tracking error is expressed in the multiplied input (motor encoder counts) and should be set accordingly. This value should generally be as small as possible.

i_nom A real value which stores the output motor's nominal (continuous) current, expressed in Amperes. A motor current exceeding the continuous current for too long will cause Flexible Servo Controller to disable the output and enter the inactive state.

motor_tc A real value which stores the motor current time constant, expressed in seconds. If the motor time constant is not known then a value of around 20 – 60 seconds is a reasonable guess for medium-sized motors.

Motor heating model

An internal model of the motor temperature based on the motor current allows for operation at high motor currents for short times while turning the output off in the event of a prolonged over-current situation. If the following text is unnecessarily complex, setting `i_nom` to an acceptable average motor current and `motor_tc` to 20 – 60 seconds is a reasonable first guess. If during operation the motor becomes undesirably warm decrease either the time constant or the nominal current; decreasing the time constant will cause the motor overheating lockout to trigger earlier while decreasing the nominal current will cause the lockout to trigger for lower currents.

The motor heating model assumes all motor heating is due to resistive self-heating (proportional to the motor current squared) and that the motor temperature is relatively well modeled by a first-order (single-pole) dynamic system. The parameters `i_nom` and `motor_tc` control the nominal (maximum continuous) current and the motor time constant respectively. For illustrative purposes, for a constant motor current the modeled motor temperature will then be equal to

$$t = i^2 \left(1 - e^{-T/\tau}\right)$$

where τ is the motor time constant (specified by `motor_tc`), i is the motor current, T is time, and t is a value that is proportional to the motor temperature relative to the ambient temperature. If t exceeds `i_nom`^{2}, the output is switched off and an event is written to the serial interface.

For example, with the model parameters `i_nom` = 5 and `motor_tc` = 60, a constant motor current of 5A would be permitted, while if the motor current were $\sqrt{2 \cdot 5^2} \approx 7.07$A (giving twice as much heating in the motor), Flexible Servo Controller would switch the output off after 41.6 seconds, as $7.07^2 \left(1 - e^{-41.6/60}\right) \approx 5^2$.

Status messages

Status messages are written to the serial interface when different conditions happen, some of which cause the output to be disabled, as listed below;

EVENT0 Enabled, output on. Written when the enable input is asserted and the motor controller is activated.

EVENT1 Disabled, output off. Written when the enable input is not asserted and the motor controller is deactivated.

EVENT2 Bus under-voltage, output off. Written when the measured bus (supply) voltage falls below the set minimum bus voltage `v_min`. The output is deactivated and Flexible Servo Controller enters the inactive state.

EVENT3 Motor long-term over-current, output off. Written when the motor current has exceeded the long-term maximum values as defined by `i_nom` and `motor_tc`.

EVENT4 Maximum tracking error exceeded, output off. Written when the difference between the commanded position and the measured encoder position differs by more than `trk_err`. The output is deactivated and Flexible Servo Controller enters the inactive state.

Logged errors

The following messages are written to the serial interface and logged to internal memory when an error occurs. Any of these events occurring will immediately disable the output (placing the motor in the coast state).

ERR0 *Output short-circuit.* This indicates that a short-circuit was detected on one or both of the motor leads, either between the leads or to POWER_GND or POWER_POS.

ERR1 *H-bridge overheat.* This indicates that the H-bridge (IC8) temperature has exceeded an internal maximum temperature, which can occur at high bus voltages and with insufficient cooling.

ERR2 *H-bridge under-voltage.* This indicates than an internally regulated voltage in the H-bridge has fallen too low, which should generally never occur. Should this error occur repeatedly there is most likely a hardware fault with the device in question.

ERR3 *Bus over-voltage.* This indicates that the measured bus voltage has exceeded the set maximum bus voltage `v_max`. Ensure that either a two-quadrant (bi-directional) power supply is used or use a braking resistor or some other device to limit the maximum bus voltage.

ERR4 *Output over-current (through current sensor). Restart (power cycle) to restore functionality.* This indicates that the output current has exceeded IC3's internal fault current threshold. This can occur at as low as 22 A, which is why this input is ignored when `high_i_en` is set to one. Should this error occur Flexible Servo Controller must be power cycled in order to restore functionality.

ERR5 *System lockup (watchdog timer overflow).* This indicates that Flexible Servo Controller's internal software has locked-up and, as a failsafe, has restarted. Should this error occur there is either an internal problem with the software for Flexible Servo Controller which must be corrected for reliable operation or the index input on X2 has experienced a prolonged period of very high-frequency logic level transitions beyond the specifications shown in section Electrical Characteristics. In the event of a software error there is the potential for damage to the hardware connected to the motor.

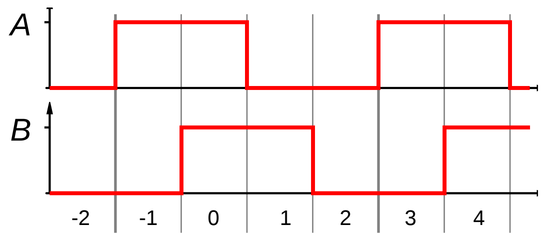


Figure 5: Quadrature-type signal giving a constant speed in one direction.

Usage

This section will cover how to use Flexible Servo Controller; what to connect, how to tune the motor position (PID) controller, and how to use Flexible Servo Controller when fully configured.

External connections

In order to function, Flexible Servo Controller requires high-power connections to a power supply and servo motor as well as logic-level signaling connections to controlling logic and an incremental encoder attached to the servo motor.

Logic-level connections

There are no significant cabling requirements for the isolated I/O interface and encoder feedback (as defined in section Pin Description), though keeping the cable between the encoder and Flexible Servo Controller as short as possible is beneficial, as is avoiding bundling the encoder cable close to the motor power conductors.

Bare-minimum connections to the isolated I/O header (X3) are;

V+ & GND 2.7 V - 5 V supply to the logic isolators (see section Electrical Characteristics for details).

IN_A & IN_B The quadrature reference inputs. These two inputs are to be driven with 90° out-of-phase inputs (as shown in figure 5) to set the motor reference position. If the input prescaler (as described in section Software Interface) is set to zero one step on this input will change the motor reference position one encoder step. If the input prescaler is set to 4 (giving a step multiplier of 16) one step on the input will change the motor reference position 16 encoder steps.

CP/$\bar{\text{E}}$ An enable input, configurable between a charge-pump or active-low enable input. This input controls the entire state of Flexible Servo Controller; when not asserted Flexible Servo Controller is placed in an inactive state (either brake or coast as defined by the `brake_en` software parameter), while when asserted Flexible Servo Controller is brought into an active state and the motor position (PID) controller is activated and will drive the motor to the reference position as set by the IN_A

and IN_B inputs.
When configured as charge pump the input logic level must transition high-to-low or low-to-high at a minimum rate defined in section Electrical Characteristics to assert the input, while when configured as an active-low input keeping the input at a logic low level will assert the input. Generally a charge-pump input is far more robust than a logic-level input, and the charge-pump input mode is strongly recommended. When transitioning to the enabled state the reference position is set to the current motor position (any previous input to IN_A and IN_B is disregarded).

Other useful, though not necessarily required, connections are;

OUT_Z An index output, this is mirrored to the index input on the encoder header and useful for determining the position of the encoder. Note the relatively large propagation delay specified in Electrical Characteristics makes this useful primarily for low-speed motion.

OUT_ACT A device-active output, this output is kept at a logic high level when the motor position is actively controlled.

$\bar{F}$ A fault-indicating output, this output is held at a logic low level when a fault has been detected (as defined in section Software Interface).

TX & RX The serial interface connections, as described in section Initial Startup, are useful for monitoring and settings system parameters.

The jumper JP1 controls the serial input, when connected with a jumper (shorted) on startup Flexible Servo Controller will accept commands on the serial interface, while when left unconnected on startup Flexible Servo Controller will not accept any input on the serial interface. It is generally a good idea to leave JP1 unconnected when fully configured in order to reduce the risk of interference causing to corrupt input commands!

High-power connections

Input and motor cabling must be chosen to be of sufficient conductor area for the desired application; this is primarily dependent on the current demands that the application poses. A low-current application with an average motor current < 5 A can get by with cabling with a conductor diameter of $0.75\,\text{mm}^2$, while a high-current application with sustained currents of upwards of 25 A will require cabling with a conductor area of $4\,\text{mm}^2$.

For high-current and/or high-voltage applications (see section Typical Performance) some form of active cooling may be required. No heatsinking of individual components is generally needed, and only modest airflow levels are required even for the very highest power output. For applications with short ($<$20 second) periods of high output current and large intervals of relatively low current active cooling may not be required; this must be evaluated on a case-by-case basis.

Before operating Flexible Servo Controller at high output currents, ensure that the microprocessor programmer is not connected to J1, as a high-current ground loop may otherwise cause spurious resets of Flexible Servo Controller!

Status LEDs

Three status LEDs indicate the status of Flexible Servo Controller. From left to right (as seen in figure 25a), these are;

LED3 Power status; lit when Flexible Servo Controller is powered up and functioning correctly.

LED2 Active status; lit when Flexible Servo Controller is in its active operation mode and driving the motor.

LED1 Fault status; lit when an error has been logged to memory. Operation may continue, though a system reset (power cycle) may be required. This LED will blink on and off while LED2 and LED3 are unlit if the stored settings are found to be corrupt on startup; this can only be resolved by acknowledging the error, recalibrating the output, and re-entering all previous parameters using the serial (UART) interface.

Power supply and motor requirements

For some applications Flexible Servo Controller will require a relatively powerful power supply. Generally higher supply voltages increase the total efficiency, reduce noise from the motor, and increase the top speed available. However, higher power supply voltages generally make the motor position (PID) controller less stable. A reasonable rule-of-thumb is to select a power supply with a nominal output voltage 150% to 200% of the voltage needed to drive the chosen servo motor at the maximum desired speed, or if this is unknown a power supply of 24 V to 42 V should work for most situations. The current capability of the power supply can be determined by the maximum speed and torque requirements from the motor, a rough rule-of-thumb indicator is to select a power supply with an output current capability of at least

$$I_{PSU} = \nu \frac{2\pi\tau\omega}{60\eta V_{PSU}}$$

where ν is the amount of current headroom in the power supply, τ is the maximum output torque [Nm], ω is the maximum rotational speed [rpm], η is the motor efficiency, and V_{PSU} is the power supply output voltage. For example, for an application with a maximum torque of 1 Nm, a maximum speed of 1500 rpm, a current overhead of 50% (giving $\nu = 1.5$), a motor efficiency of 80% (giving $\eta = 0.8$), and a power supply voltage of 36 V, this gives a current capability requirement of

$$I_{PSU} = 1.5\frac{2\pi \cdot 1 \cdot 1500}{60 \cdot 0.8 \cdot 36} \approx 8.2\,A.$$

If Flexible Servo Controller is intended to be used with a relatively "weak" power supply the maximum motor current can be limited by reducing the `i_max` parameter, thereby also reducing the current demands from the power supply, however this is not an optimal solution as this reduces the maximum motor torque for all speeds, while the power supply limits the total performance only at high speeds and high torque. Keep in mind that if using several Flexible Servo Controllers connected to the same power supply that the current capability will generally be the sum of the requirements of each individual unit.

For most applications a braking resistor or a very large capacitor bank is required in order to limit the bus voltage when the output motor is decelerating the output, which generates power and causes the bus voltage to increase. For most motion-control applications the kinetic energy stored in the mechanical system when moving quickly is relatively large and a braking resistor is the only feasible way of dissipating this energy (as very few power supplies are capable of absorbing energy from the bus, and an impractically large capacitor bank would be required to absorb the kinetic energy). See the Rabid Mantis Braking Resistor (`http://www.rabidmantis.se`) for an example of one method of handling the regenerative power. **Using Flexible Servo Controller without any method of absorbing regenerative power may potentially damage Flexible Servo Controller, the power supply, or both[7]!** For applications with very low kinetic energy a large capacitor bank may be used to reduce the total system complexity, but must be studied on a case-by-case basis.

The connected motor's winding inductance must be large enough to keep the ripple current within reasonable bounds. A rough rule-of-thumb is to require a winding inductance that results in a ripple current of at most 10% of the nominal motor current. Assuming a value of three for the `i_skip` parameter (and a current controller frequency of 200 kHz) this gives a minimum winding inductance of

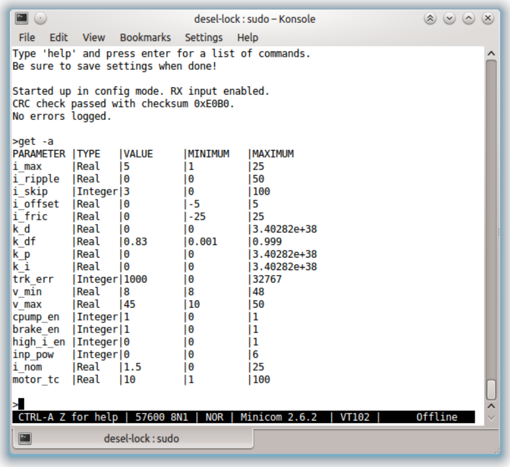
$$\begin{aligned} L_{min} &= \frac{V_{PSU}}{\frac{\frac{0.1 \cdot I_{nom}}{3}}{200 \cdot 10^3}} \\ &= 0.15 \cdot 10^{-3} \frac{V_{PSU}}{I_{nom}} \end{aligned}$$

where L_{min} is the minimum winding inductance, V_{PSU} is the power supply voltage, and I_{nom} is the nominal motor current. For example, for a nominal current of 5 A and a power supply voltage of 24 V this gives

$$L_{min} = 0.15 \cdot 10^{-3} \frac{36}{5} = 1.08\,\mathrm{mH}.$$

Though motors with a winding inductance less than this can be used, significantly smaller values will result in excessive self-heating.

[7] Flexible Servo Controller will enter an error state that creates as little regenerative power as possible when `v_max` is exceeded, however, this should not be used as anything other than a last-ditch safety measure.


desel-lock : sudo – Konsole
File Edit View Bookmarks Settings Help
desel-lock : sudo

```
Type 'help' and press enter for a list of commands.
Be sure to save settings when done!

Started up in config mode. RX input enabled.
CRC check passed with checksum 0xE0B0.
No errors logged.

>get -a
PARAMETER |TYPE   |VALUE     |MINIMUM   |MAXIMUM
i_max     |Real   |5         |1         |25
i_ripple  |Real   |0         |0         |50
i_skip    |Integer|3         |0         |100
i_offset  |Real   |0         |-5        |5
i_fric    |Real   |0         |-25       |25
k_d       |Real   |0         |0         |3.40282e+38
k_df      |Real   |0.83      |0.001     |0.999
k_p       |Real   |0         |0         |3.40282e+38
k_i       |Real   |0         |0         |3.40282e+38
trk_err   |Integer|1000      |0         |32767
v_min     |Real   |8         |8         |48
v_max     |Real   |45        |10        |50
cpump_en  |Integer|1         |0         |1
brake_en  |Integer|1         |0         |1
high_i_en |Integer|0         |0         |1
inp_pow   |Integer|0         |0         |6
i_nom     |Real   |1.5       |0         |25
motor_tc  |Real   |10        |1         |100

>
 CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

Figure 6: Zero-initialized controller gains.

Motor position (PID) controller tuning

Manual PID controller tuning is widely regarded as equal parts science and operator skill, making it difficult to define a simple formula that works for all motor and encoder combinations. What follows below is first a guide to determining the correct motor polarity and afterwards some guidelines for determining reasonable controller gains.

To begin with, ensure that the steps described in section Initial Startup have been followed, and afterwards connect the motor terminals, the motor encoder, and the desired logic control signals described above to the device that is to control Flexible Servo Controller. To begin with, disconnect the motor shaft from any load, allowing the shaft to rotate freely. **Ensure that the charge pump or enable input is not active before continuing**! Finally, start up the power supply and the serial terminal application on the host PC.

To begin with, set the parameters `i_ripple`, `i_skip`, `v_min`, `v_max`, `cpump_en`, `brake_en`, `high_i_en`, `inp_pow`, `i_nom`, and `motor_tc` to their application specific desired values using the `set` and `get` commands[8] (described in section Software Interface). Verify that all the motor position (PID) controller gains (`k_d`, `k_p`, `k_i`) and the motor friction parameter (`i_friction`) are set to zero and that the maximum motor current is set to a small value, such as 5A (as shown in figure 6).

For most motion-control applications the controller speed (the ability to as quickly as possible move to the commanded position) is of high importance, and for these applications a value of `k_df` of 0.83 is suitable (as shown in table 3). For applications where the the output should intentionally be slower and less responsive a lower value can be chosen (see previously described table).

[8]For example, to display the values of all parameters use the command `get -a`, and to set the minimum bus voltage to 12.5 volts use the command `set -p v_min -v 12.5`.

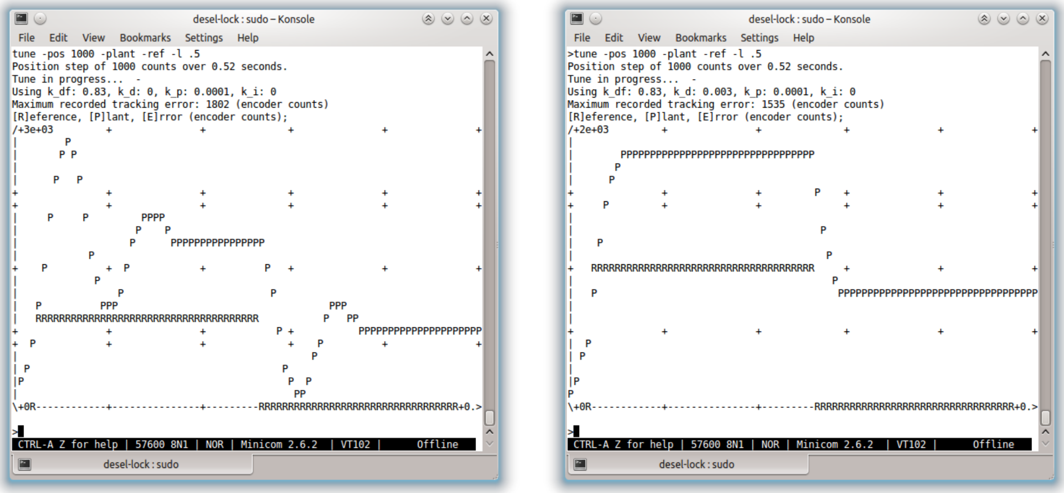
In order to determine whether the motor is connected in the correct direction or not set `k_p` to a small value, such as 0.0001 (equivalent to 1e-4, in other words driving an additional 0.1mA of current in the motor for each encoder step). Before continuing, ensure that the maximum tracking error `trk_err` is set to a relatively large value, such as 10000 and that a self-calibration has been performed and saved (the `i_offset` parameter will be $\neq 0$ if a calibration has been performed).

Now, enable the charge-pump or enable input. If the motor terminals are connected in the wrong direction, the output will briefly rotate and the status messages `EVENT0: Enabled, output on` and `EVENT4: Maximum tracking error exceeded, output off` will be printed in rapid succession. In this case, reverse the motor terminal (LOAD_POS and LOAD_NEG) connections and re-enable the output by turning the charge-pump / enable input off, and then on. When the motor is connected correctly the output shaft will rotate slightly, stabilize at some position, and resist external torque applied (such as by rotating the shaft by hand). A constant hissing sound from the motor is normal and expected. Should both combinations of the motor lead connections cause the Flexible Servo Controller to display `EVENT4: Maximum tracking error exceeded, output off`, try to increase the maximum tracking error `trk_err` and/or increase the proportional gain `k_p` until the output stabilizes. If the output oscillates try reducing `k_p` until the output stabilizes. Once stable, set the parameter `i_max` to the desired absolute maximum output current and save the parameter settings to memory.

At this stage, the motor position (PID) controller is ready to be tuned. Connect the motor to the final load that is to be driven, reduce the maximum tracking error parameter to around 1000 to reduce the risk of motor runaway, and **ensure that should the motor rotate too far in any direction that the charge-pump/enable input is deactivated! Damage to the motor and/or load may occur should the controller drive the motor at maximum power against a mechanical limit, which is possible when manually tuning the controller!**

There are many different ways to manually tune a PID controller, many of which consist of slightly modifying one (or more) controller parameters and studying the result of a change in setpoint. The `tune` command is designed particularly for this application, and is described in section Software Interface. One method which has shown to work relatively well is to gradually increase `k_p` until a position step response shows significant overshoot and ringing, after which `k_d` is increased, reducing the ringing and allowing `k_p` to be increased even more. Eventually, raising k_d will itself causing ringing, which is an indicator of a value that is too large (see figure 7). Be sure to make full use of the length (`-l`) parameter when tuning, start with a relatively large time in order to see the behavior when the controller is slow, and decrease the time when the controller becomes faster and more aggressive.

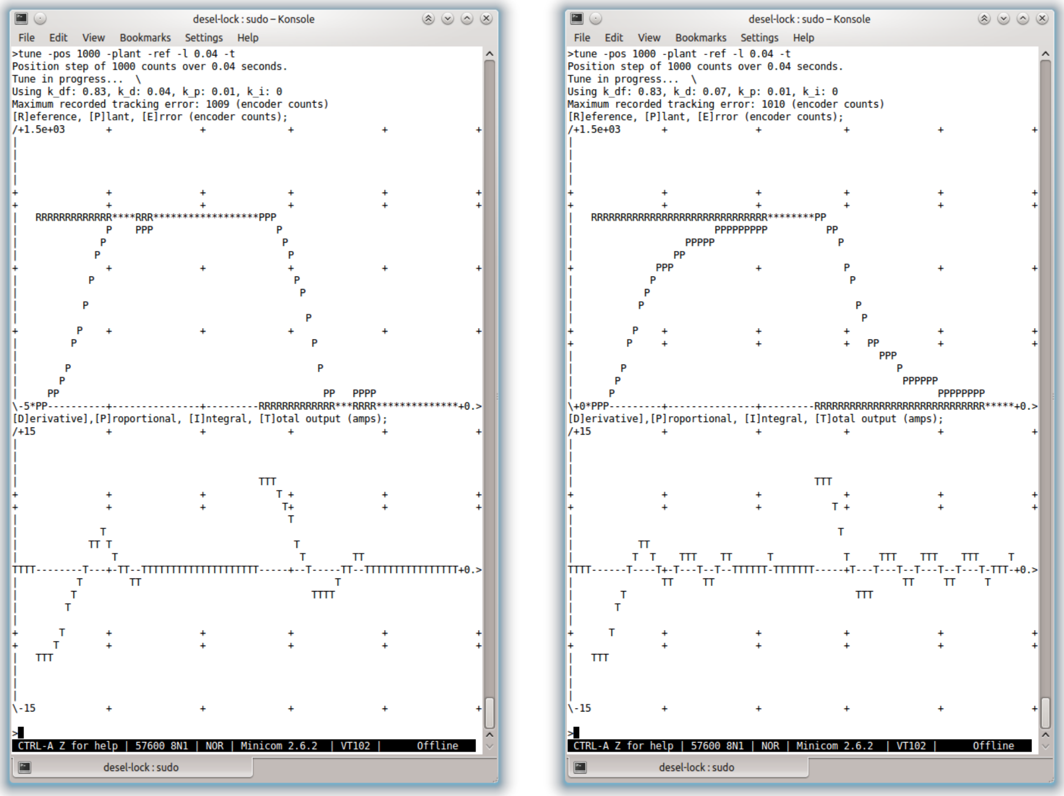
Finding a good value for `k_i` is more easily performed with a velocity step, and in particular when viewing the tracking error, as is shown in figure 8. Slowly raise the k_i parameter until the residual error is very small, while avoiding raising it too much causing excessive ringing or oscillation. Sometimes it may be necessary to reduce `k_d` and `k_p` slightly in order to arrive to the best `k_i` value — it is here that skill and experience



```
tune -pos 1000 -plant -ref -l .5
Position step of 1000 counts over 0.52 seconds.
Tune in progress...  -
Using k_df: 0.83, k_d: 0, k_p: 0.0001, k_i: 0
Maximum recorded tracking error: 1802 (encoder counts)
[R]eference, [P]lant, [E]rror (encoder counts);
/+3e+03         +               +              +               +               +
|        P
|       P P
|
|      P   P
+               +               +              +               +               +
+               +               +              +               +               +
|     P     P         PPPP
|                    P    P
|                   P      PPPPPPPPPPPPPPPP
|            P
+    P          +  P            +          P   +               +               +
|             P
|                 P                         P
|   P          PPP                                    PPP
|   RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR           P   PP
+               +               +            P +           PPPPPPPPPPPPPPPPPPPPP
+  P            +               +              +    P          +               +
|                                                  P
| P                                           P
|P                                             P  P
|                                               PP
\+0R------------+---------------+---------RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR+0.>

>
CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

```
>tune -pos 1000 -plant -ref -l .5
Position step of 1000 counts over 0.52 seconds.
Tune in progress...  -
Using k_df: 0.83, k_d: 0.003, k_p: 0.0001, k_i: 0
Maximum recorded tracking error: 1535 (encoder counts)
[R]eference, [P]lant, [E]rror (encoder counts);
/+2e+03         +               +              +               +               +
|
|        PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
|       P
|      P
+               +               +         P    +               +               +
+     P         +               +              +               +               +
|
|                                          P
|    P
|                                           P
+   RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR     +               +               +
|                                            P
|   P                                         PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
|
|
+               +               +              +               +               +
|  P
| P
|
|P
P
\+0R------------+---------------+---------RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR+0.>

>
CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

(a) For small values of `k_d` increases decrease the overshoot and ringing.

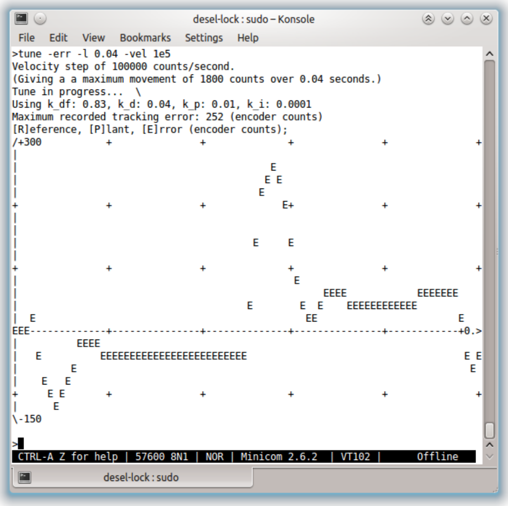


```
>tune -pos 1000 -plant -ref -l 0.04 -t
Position step of 1000 counts over 0.04 seconds.
Tune in progress...  \
Using k_df: 0.83, k_d: 0.04, k_p: 0.01, k_i: 0
Maximum recorded tracking error: 1009 (encoder counts)
[R]eference, [P]lant, [E]rror (encoder counts);
/+1.5e+03       +               +              +               +               +
|
|
|
|
+               +               +              +               +               +
+               +               +              +               +               +
|   RRRRRRRRRRRRR****RRR******************PPP
|               P    PPP                     P
|              P                              P
|             P                                P
+               +               +              +               +               +
|            P                                  P
|                                                P
|           P
|                                                 P
+          P    +               +              +               +               +
|         P                                        P
|
|        P                                          P
|       P
|     PP                                             PP   PPPP
\-5*PP----------+---------------+---------RRRRRRRRRRRRR***RRRR**************+0.>
[D]erivative],[P]roportional, [I]ntegral, [T]otal output (amps);
/+15            +               +              +               +               +
|
|
|
|                                         TTT
+               +               +            T +               +               +
+               +               +             T+               +               +
|                                              T
|              T
|            TT T                               T
|                T                               T        TT
TTTT--------T---+-TT--TTTTTTTTTTTTTTTTTTTT-----+--T-----TT--TTTTTTTTTTTTTTTT+0.>
|          T        TT                                 T
|         T                                        TTTT
|        T
|
+       T       +               +              +               +               +
+      T        +               +              +               +               +
|   TTT
|
|
|
\-15            +               +              +               +               +

>
CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

```
>tune -pos 1000 -plant -ref -l 0.04 -t
Position step of 1000 counts over 0.04 seconds.
Tune in progress...  \
Using k_df: 0.83, k_d: 0.07, k_p: 0.01, k_i: 0
Maximum recorded tracking error: 1010 (encoder counts)
[R]eference, [P]lant, [E]rror (encoder counts);
/+1.5e+03       +               +              +               +               +
|
|
|
|
+               +               +              +               +               +
+               +               +              +               +               +
|   RRRRRRRRRRRRRRRRRRRRRRRRRRRRRR********PP
|                        PPPPPPPPP          PP
|                   PPPPP                     P
|                 PP
+              PPP              +              P               +               +
|             P                                 P
|            P
|           P                                    P
|                                                 P
+          P    +               +              +               +               +
+         P     +               +              +   PP          +               +
|                                                    PPP
|        P                                              P
|       P                                                PPPPPP
|      P                                                       PPPPPPPP
\+0*PPP---------+---------------+---------RRRRRRRRRRRRRRRRRRRRRRRRRRRRR*****+0.>
[D]erivative],[P]roportional, [I]ntegral, [T]otal output (amps);
/+15            +               +              +               +               +
|
|
|
|                                         TTT
+               +               +              +               +               +
+               +               +            T +               +               +
|
|                                             T
|           TT
|          T  T    TTT    TT      T            T     TTT    TTT    TTT     T
TTTT------T----T+-T---T--T--TTTTTT-TTTTTTT-----+T---T---T--T---T--T---T-TTT-+0.>
|               TT     TT                                TT     TT     T
|        T                                       TTT
|       T
|
+      T        +               +              +               +               +
+               +               +              +               +               +
|   TTT
|
|
|
\-15            +               +              +               +               +

>
CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

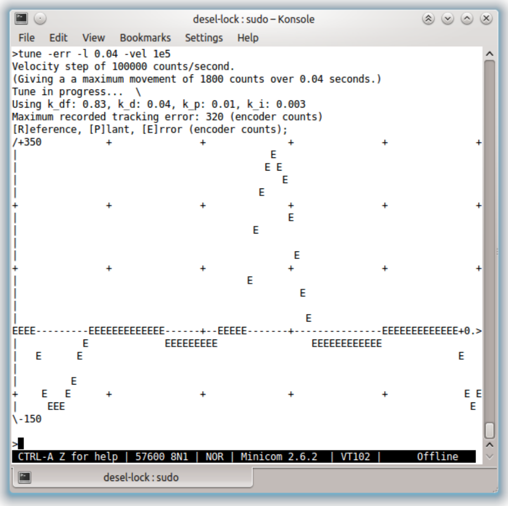
(b) For values of k_d that are above the ideal value ringing increases, which is particularly apparent in the output motor current.

Figure 7: Ringing is reduced by increasing `k_d`, up to a point.



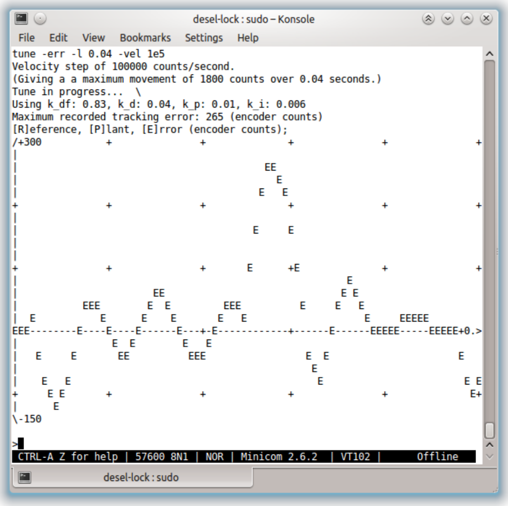
```
>tune -err -l 0.04 -vel 1e5
Velocity step of 100000 counts/second.
(Giving a a maximum movement of 1800 counts over 0.04 seconds.)
Tune in progress...  \
Using k_df: 0.83, k_d: 0.04, k_p: 0.01, k_i: 0.0001
Maximum recorded tracking error: 252 (encoder counts)
[R]eference, [P]lant, [E]rror (encoder counts);
/+300           +               +              +               +               +
|
|                                           E
|                                          E E
|                                         E
+               +               +             E+               +               +
|
|
|                                        E     E
|
+               +               +              +               +               +
|                                               E
|                                                    EEEE            EEEEEEE
|                                       E        E  E    EEEEEEEEEEEE
|  E                                              EE                        E
EEE-------------+---------------+--------------+---------------+------------+0.>
|          EEEE
|   E          EEEEEEEEEEEEEEEEEEEEEEEEE                                     E E
|         E                                                                   E
|    E   E
+     E E       +               +              +               +               +
|      E
\-150

>
CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

(a) A `k_i` value that is too small will not cancel out a constant offset.



```
>tune -err -l 0.04 -vel 1e5
Velocity step of 100000 counts/second.
(Giving a a maximum movement of 1800 counts over 0.04 seconds.)
Tune in progress...  \
Using k_df: 0.83, k_d: 0.04, k_p: 0.01, k_i: 0.003
Maximum recorded tracking error: 320 (encoder counts)
[R]eference, [P]lant, [E]rror (encoder counts);
/+350           +               +              +               +               +
|                                           E
|                                          E E
|                                             E
|                                         E
+               +               +              +               +               +
|                                              E
|                                        E
|
|                                               E
+               +               +              +               +               +
|                                       E
|                                                E
|
|                                                 E
EEEE---------EEEEEEEEEEEEE------+--EEEEE-------+---------------EEEEEEEEEEEEE+0.>
|           E             EEEEEEEEE                EEEEEEEEEEEE
|   E      E                                                                E
|
|         E
+    E   E      +               +              +               +             E E
|     EEE                                                                     E
\-150

>
CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

(b) A good `k_i` value will cancel out a constant offset without causing excessive ringing.



```
tune -err -l 0.04 -vel 1e5
Velocity step of 100000 counts/second.
(Giving a a maximum movement of 1800 counts over 0.04 seconds.)
Tune in progress...  \
Using k_df: 0.83, k_d: 0.04, k_p: 0.01, k_i: 0.006
Maximum recorded tracking error: 265 (encoder counts)
[R]eference, [P]lant, [E]rror (encoder counts);
/+300           +               +              +               +               +
|
|                                          EE
|                                            E
|                                         E   E
+               +               +              +               +               +
|
|                                        E     E
|
|
+               +               +       E      +E              +               +
|                                                        E
|                      EE                              E E
|           EEE        E  E         EEE          E     E   E
|  E           E      E    E       E   E                    E     EEEEE
EEE--------E----E----E------E---+-E------------+------E------EEEEE-----EEEEE+0.>
|                E  E        E   E
|   E     E       EE          EEE                E  E                      E
|                                                 E
|    E   E                                         E                        E E
+     E E       +               +              +               +              E+
|      E
\-150

>
CTRL-A Z for help | 57600 8N1 | NOR | Minicom 2.6.2  | VT102 |      Offline
```

(c) A too large `k_i` value will cause a constant ringing or oscillation.

Figure 8: A `k_i` value that is too small will not cancel out any residual error in the velocity step, while too large a value will cause excessive ringing or oscillation.

with manual controller tuning becomes very useful.

The `i_fric` parameter may be used with in systems with significant dynamic friction. In these systems slowly increase this parameter (it may be necessary to reduce `k_d`, `k_p`, and `k_i`) until the velocity step has a relatively low current when traveling at constant speed (as can be shown with the `-t` flag in the tune command). Too large values will cause significant oscillation.

It is now possible to determine the maximum velocity and acceleration the entire system can achieve by using the velocity and acceleration steps respectively; slowly increase the velocity/acceleration until there is an increased residual error, this gives a rough indication of the maximum speed and acceleration the system is capable of achieving.

Finally, set the maximum tracking error to a more modest value; a rough figure of the maximum expected tracking error during normal use can be established using the acceleration step, after which is maximum permissible tracking error may be set to a value 2-5 times greater depending on the type of application. **Do not forget to save the system parameters to memory after the tuning is completed!**

Configured use

Once the internal system parameters have been configured and the motor position (PID) controller has been tuned Flexible Servo Controller is ready for use! It is now generally a good idea to remove the configuration jumper JP1, which will disable all input commands over the serial interface. The serial output will still function as before however, making it possible to trigger PC software on EVENT or ERR keywords.

In ordinary operation, the OUT_ACT output indicates that the output is active, and can be used to monitor the status of Flexible Servo Controller. If an event or error brings Flexible Servo Controller into the inactive state (such as a bus under-voltage) the output can be re-activated by bringing the charge-pump / enable input inactive, and then activating the output again. Note; Due to hardware limitations, if an over-current error occurs the entire device must be power cycled (reset) in order to restore functionality!

Bill of materials

Note; A value of 4n7 corresponds to a value of 4.7n, and in the case of a capacitor corresponds to 4.7 nF. The suggested part number is only that — a suggestion — and may be replaced with any other equivalent matching the specifications listed under value, rating, and type.

Component name	Value	Rating	Type	Suggested Part No.
C1, C11	10n	25 V 10%	X7R ceramic, SMD 0805 package	MCCA001261
C2, C16	1u	50 V 10%	X7R ceramic, SMD 1206 package	MCCA001425
C7, C9	10n	100 V 10%	NP0/C0G ceramic, SMD 1206 package	C3216C0G2A103J115AA
C3	470 – 680u	63 V 30%	Axial electrolytic, 44 mm pin grid. Very high (application-specific) ripple current.	PEG225MF3470QE1
C4, C15, C18	10u	25 V 10%	X7R ceramic, SMD 1206 package	12063C106KAT2A
C5, C10, C12, C13, C14, C17, C19, C20, C21, C22, C23, C24, C25, C26, C27, C28	470n	25 V 10%	X7R ceramic, SMD 0805 package	C2012X7R1E474K125AA
C8	2n2	100 V 10%	NP0/C0G ceramic, SMD 1206 package	C3216C0G2J222J115AA
C6	470u	6.3 V 30%	SMD electrolytic, Panasonic size code ’F’ (8 mm can)	EEEFK0J471P
D1, D2	2A	60 V	SMD SMB package Schottky diode	SS26
D3, D4, D5, D6	5.0V	-	SMD SOT-353 package quad ESD-protection diode array	PESD5V0L4UG
F1	-	-	Regular size automotive fuse (ATO, ATC, or APR) holder	MCCQ-121
	25A	-	Standard 25A ATO, ATC, or APR automotive fuse. Use a fuse with a lower rated current as a safety for low-current applications.	0287025.PXCN
F2, F3	200mA	30 V	PTC resettable fuse, SMD 1210 package (for 5 V encoder output)	1210L020
	50mA	30 V	PTC resettable fuse, SMD 1210 package (for 3.3 V encoder output)	1210L005

COMPONENT NAME	VALUE	RATING	TYPE	SUGGESTED PART NO.
IC1	-	-	5 V Buck regulator, SMD SO-8 package.	A8499
IC2	-	-	3.3 V Linear regulator, SMD SOT-89 package	MCP1700T-3302E/MB
IC3	-	-	Hall-effect current sensor, SMD SO-8 package	ACS711KLCTR-25AB-T
IC4, IC5, IC9, IC11	-	-	Digital isolator (2 channel), SMD SO-8 package	ADUM1201A
IC6	-	-	Microprocessor, SMD TQFP44 package	ATXMEGA32A4U-AU
IC7	-	-	TTL-compatible hex inverting Schmitt trigger, SMD TSSOP14 package	74HCT14PW
IC8	-	-	H-bridge driver, SMD TSSOP package	A3941
IC10	-	-	5V-tolerant low voltage hex inverting Schmitt trigger, SMD TSSOP14 package	74LCX14MTC
J1	-	-	6-way 2-row 2.54 mm pitch through-hole pin-header	M20-9980346
JP1	-	-	2-way 1-row 2.54 mm pitch through-hole pin-header	M20-9990245
	-	-	2-way 2.54 mm pitch jumper	881545-1
L1	22u	0.6 A	SMD 1812 package	LQH43PN220M26L
L2, L3, L4, L5, L6, L7	500Ω@100MHz	500 mA	Ferrite bead, high suppression at low frequencies, SMD 0603 package	MMZ1608B601C
LED1, LED2, LED3	-	20 mA	Generic SMD 1206 package LED	KPT-3216YC
R1	27k	1%, 63 mW	Thick film SMD 0603 package	MC0063W0603127K
R2, R34	33k	1%, 63 mW	Thick film SMD 0603 package	MC0063W0603133K
R3, R11, R22, R33, R35	10k	1%, 63 mW	Thick film SMD 0603 package	MC0063W0603110K
R16, R23, R32			Mounted normally, **not** mounted in alternate bias resistor configuration. Thick film SMD 0603 package	
R19, R27, R38			**Not** mounted normally, only mounted in alternate bias resistor configuration. Thick film SMD 0603 package	
R4, R5, R9, R12, R13	560k	1%, 63 mW	Thick film SMD 0603 package	MC0063W06031560K
R6, R7, R8	10R	5%, 2 W	Thick film SMD 2512 package	CRM2512-JW-100ELF
R10, R18, R26, R40	100k	1%, 63 mW	Thick film SMD 0603 package	MC0063W06031100K

COMPONENT NAME	VALUE	RATING	TYPE	SUGGESTED PART NO.
R14, R15, R17, R20, R21, R25, R28, R30, R37, R41, R42, R43, R44, R45	100R	1%, 63 mW	Thick film SMD 0603 package	MC0063W06031100R
R24	470R	1%, 63 mW	Thick film SMD 0603 package	MC0063W06031470R
R29, R31, R36, R39	4R7	1%, 63 mW	Thick film SMD 0603 package	MC0063W060314R7
Q1, Q2, Q3, Q4	N-channel MOSFET	60 V, $\leq 3\,\text{m}\Omega$ $R_{DS(on)}$, $\leq 250\,\text{nC}$ $Q_{gate,tot}$	SMD D2PAK 7-pin package	IRFS3006-7PPbF
SJ1, SJ2	-	-	Solder jumper. Cut trace in SJ1, solder (short) SJ2 to enable 3.3 V encoder output. (Output defaults to 5 V).	-
X1	-	25 A	Optional 10.16 mm, 4-way screw terminal. Use 4/8 mm ring terminals otherwise.	CTB77VP/4
X2	-	-	5-way 1-row 2.5 mm pitch through-hole terminal block	PTSA 0.5/5-2.5-Z
X3	-	-	10-way 1-row 2.5 mm pitch through-hole terminal block	PTSA 0.5/10-2.5-Z

Mechanical Description

Flexible Servo Controller consists of a PCB and associated components, with a finished size of 100 mm by 50 mm, with a build height of 25 mm (35 mm when X1 is mounted), limited by the input capacitor C3. Mounting holes are listed in table 6 following the coordinate system shown in figure 9. PCB manufacturing requirements are shown in table 5 and should be generally achievable at any PCB house.

Table 5: PCB manufacturing requirements.

PARAMETER	REQUIREMENT	UNIT
PCB thickness	Any (nominal 1.6)	mm
PCB layers	2	-
Copper fill thickness/density (tested)	35/1	µm / oz/ft^2
Trace isolation (minimum)	0.2032/8	mm/mil
Trace width (minimum)	0.2032/8	mm/mil
Trace to board edge (minimum)	0.25	mm
Drill to board edge (minimum)	0.4532	mm
Drill diameter (minimum)	0.3	mm
Via annular ring (minimum)	0.2032/8	mm/mil

Table 6: Mounting hole locations.

HOLE DIAMETER [mm]	POSITION X [mm]	POSITION Y [mm]
4.1	6.5	2
4.1	6.5	48
4.1	95	5
4.1	95	45

Note; The holes at (6.5, 2) and (6.5, 48) are half-open, see figure 24.

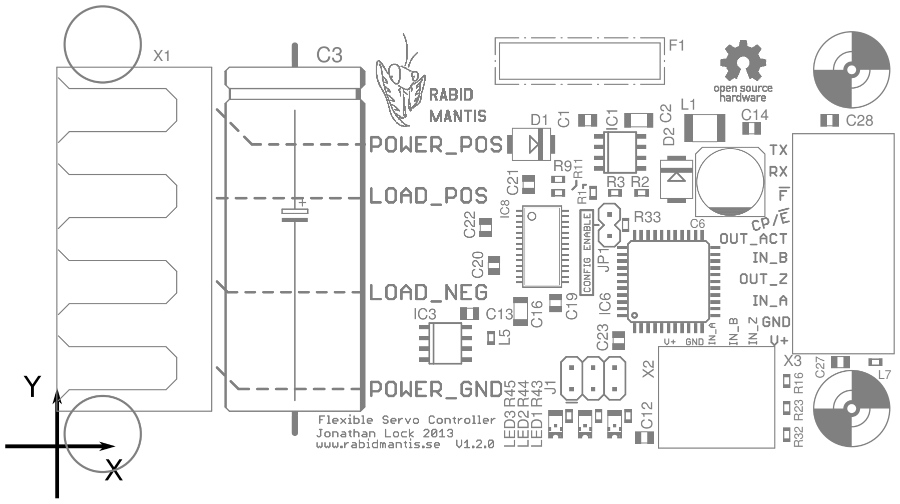


Figure 9: Coordinate system for mounting holes.

Electrical Characteristics

PARAMETER	SYMBOL	TEST CONDITIONS/COMMENTS	MIN	TYP	MAX	UNIT
DC specifications						
Power supply voltage	$V_{sup,pwr}$	Over-voltage cutoff can be set up to the absolute maximum ratings. Recommended maximum; 45V.	8		45	V
Output current range		Current driven through output load, high current mode enabled.	±25			A
Quiescent current		Output inactive, $V_{in} = 12\,V$, $I_{Encoder} = 35\,\text{mA}$		36		mA
Quiescent current		Output inactive, $V_{in} = 24\,V$, $I_{Encoder} = 35\,\text{mA}$		22		mA
Quiescent current		Output inactive, $V_{in} = 36\,V$, $I_{Encoder} = 35\,\text{mA}$		18		mA
Quiescent current		Output inactive, $V_{in} = 42\,V$, $I_{Encoder} = 35\,\text{mA}$		16		mA
Quiescent current		Output active, $L_{load} = 2.95\,\text{mH}$, i_ripple = 0, i_skip = 3, $V_{in} = 12\,V$, $I_{Encoder} = 35\,\text{mA}$		125		mA
Quiescent current		Output active, $L_{load} = 2.95\,\text{mH}$, i_ripple = 0, i_skip = 3, $V_{in} = 24\,V$, $I_{Encoder} = 35\,\text{mA}$		130		mA
Quiescent current		Output active, $L_{load} = 2.95\,\text{mH}$, i_ripple = 0, i_skip = 3, $V_{in} = 36\,V$, $I_{Encoder} = 35\,\text{mA}$		150		mA
Quiescent current		Output active, $L_{load} = 2.95\,\text{mH}$, i_ripple = 0, i_skip = 3, $V_{in} = 42\,V$, $I_{Encoder} = 35\,\text{mA}$		170		mA
Logic specifications, control header (X3)						
Logic supply voltage	V_{IO}		2.7		5.5	V
Logic supply quiescent current	$I_{Q,IO}$			3.6	4.4	mA
Logic high input threshold	$V_{IO,IH}$		$0.7 \cdot V_{IO}$			V
Logic low input threshold	$V_{IO,IL}$				$0.3 \cdot V_{IO}$	
Logic high output voltage		$I_{load,IO} = -20\,\mu\text{A}$ $I_{load,IO} = -4\,\text{mA}$	$V_{IO} - 0.1$ $V_{IO} - 0.9$	V_{IO} $V_{IO} - 0.6$		V V
Logic low output voltage		$I_{load,IO} = 20\,\mu\text{A}$		0.0	0.1	V

PARAMETER	SYMBOL	TEST CONDITIONS/COMMENTS	MIN	TYP	MAX	UNIT
		$I_{load,IO} = 4\,\text{mA}$		0.6	0.9	V
Logic input current, $V_{IO,I} = V_{sup,IO}$		$V_{sup,IO} = 5\,\text{V}$ $V_{sup,IO} = 3.3\,\text{V}$	-10 -10	+0.01 +0.01	+10 +10	μA μA
Logic input current, $V_{IO,I} = 0\,\text{V}$		$V_{sup,IO} = 5\,\text{V}$ $V_{sup,IO} = 3.3\,\text{V}$	-40 -23	-50 -33	-60 -43	μA μA
Quadrature input data rate		Applies to both encoder input and setpoint input			1	MHz
Controller header (X3), in unisolated mode		See datasheet for IC6 for input and output logic level and current specifications.				
Logic supply voltage	$V_{sup,IO,3v3}$		3.2	3.3	3.4	V
Logic supply output current		Note; output is not short-circuit protected and shared with 3.3 V encoder output!	50			mA
Logic specifications, encoder header (X2)						
Encoder supply voltage	V_{ENC}	V_{ENC} configured for 5 V output V_{ENC} configured for 3.3 V output	4.75 3.2	5 3.3	5.25 3.4	V V
Encoder supply current	I_{ENC}	V_{ENC} configured for 5V output V_{ENC} configured for 3.3 V output	500 150			mA mA
Encoder fuse current	$I_{ENC,fuse}$	V_{ENC} configured for 5 V output V_{ENC} configured for 3.3 V output	200 50		400 150	mA mA
Logic high input threshold	$V_{ENC,IH}$		1.2		2.2	
Logic low input threshold	$V_{ENC,IL}$		0.6		1.5	
Logic input hysteresis				0.4	1.2	V
Logic input current, $V_{ENC,I} = 0$		V_{ENC} configured for 5 V output V_{ENC} configured for 3.3 V output			$\pm$55 $\pm$38	μA μA
Dynamic characteristics						
Charge-pump threshold frequency		Recommended charge-pump frequency; 400 Hz – 50 kHz	198	199	200	Hz
Enable propagation delay, charge-pump enabled		Time from charge-pump signal present to motor controller output active.	2.5		6	ms

PARAMETER	SYMBOL	TEST CONDITIONS/COMMENTS	MIN	TYP	MAX	UNIT
Enable propagation delay, charge-pump disabled		Time from enable signal present to motor controller output active.			3	ms
Disable propagation delay, charge-pump enabled		Time from charge-pump signal absent to motor controller output disabled.	5		6	ms
Disable propagation delay, charge-pump disabled		Time from enable signal absent to motor controller output disabled.			3	ms
Index propagation delay		Time from logic change on index input on X2 to logic change on index output on X3			1	ms
Minimum index period		Minimum period on index input on X2. (Time between sucessive index pulses). Extended periods with values below this may cause the device to lock up and enter a failsafe state.	50			μs
UART baud rate error		Nominal baud rate; 56700 bps	-0.6%	-0.1	+0.4	%
Output offset current		Uncalibrated[9]		±0.9		A
Output current gain error		Percentage of maximum output current		±4		%
Bus voltage measurement error		Maximum error reduced to 0.4% + 1V if R9 and R10 are replaced with 0.1% tolerance resistors.			2% + 1V	(% of reading + V)
Bus voltage measurement bandwidth				1.6		kHz

[9]Calibrated offset current not reliably determined, but is smaller in magnitude.

Typical Performance

Note; All thermal tests have been performed with a constant current output through an inductor, without any heatsinking, with i_skip = 3, i_ripple = 0, *held horizontally on an open surface, and at an ambient temperature of 26 ˚C. Thermal tests with active cooling consisted of an additional 60x60 mm axial fan placed 10 cm above directly facing and blowing towards Flexible Servo Controller with an air flow of 0.66 m^3/min (23.3 CFM).*

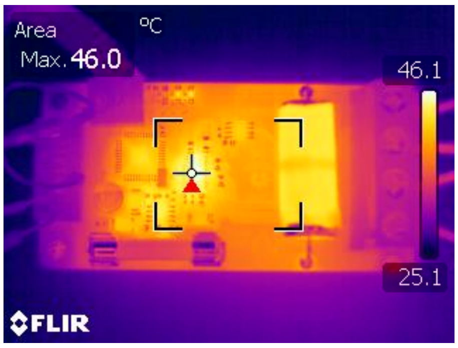


Figure 10: Thermal image of operation with $I_{LOAD} = 5\,\text{A}$, $V_{BUS} = 20\,\text{V}$, $L_{LOAD} = 1.8\,\text{mH}$, with no active cooling.

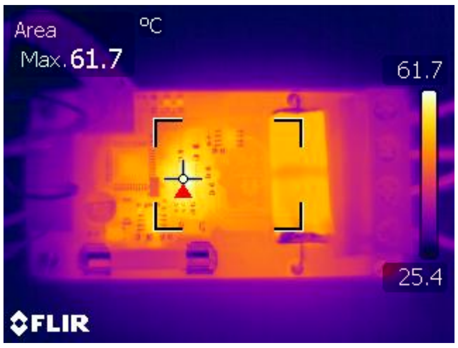


Figure 11: Thermal image of operation with $I_{LOAD} = 5\,\text{A}$, $V_{BUS} = 30\,\text{V}$, $L_{LOAD} = 1.8\,\text{mH}$, with no active cooling.

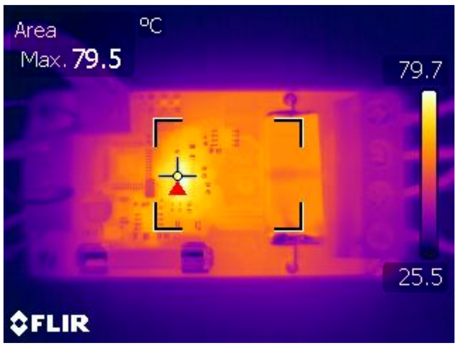


Figure 12: Thermal image of operation with $I_{LOAD} = 5\,\text{A}$, $V_{BUS} = 40\,\text{V}$, $L_{LOAD} = 1.8\,\text{mH}$, with no active cooling.

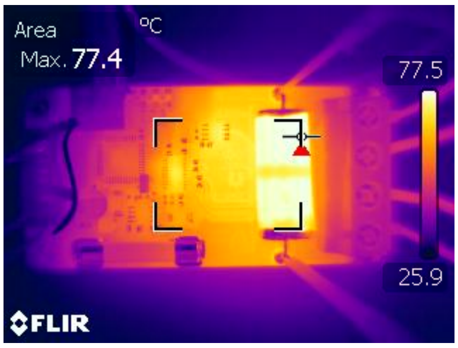


Figure 13: Thermal image of operation with $I_{LOAD} = 10\,\text{A}$, $V_{BUS} = 20\,\text{V}$, $L_{LOAD} = 1.8\,\text{mH}$, with no active cooling.

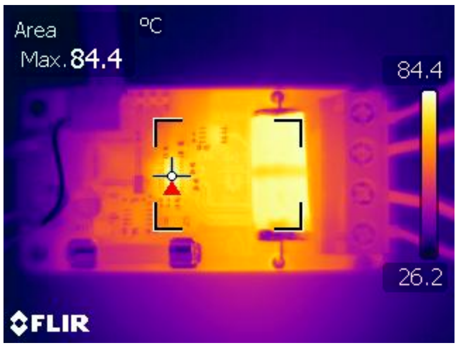


Figure 14: Thermal image of operation with $I_{LOAD} = 10\,\mathrm{A}$, $V_{BUS} = 30\,\mathrm{V}$, $L_{LOAD} = 1.8\,\mathrm{mH}$, with no active cooling.

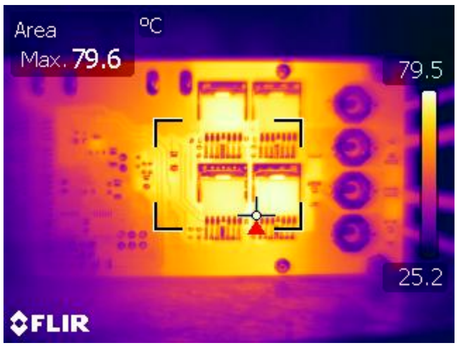


Figure 15: Thermal image of bottom of board with $I_{LOAD} = 10\,\mathrm{A}$, $V_{BUS} = 30\,\mathrm{V}$, $L_{LOAD} = 1.8\,\mathrm{mH}$, with no active cooling. MOSFET temperature imbalance due to low loss in load, causing the upper left and lower right MOSFET to be in the conductance region a larger proportion of time.

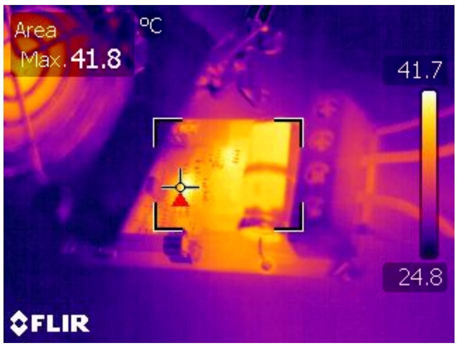


Figure 16: Thermal image of operation with $I_{LOAD} = 10\,\mathrm{A}$, $V_{BUS} = 20\,\mathrm{V}$, $L_{LOAD} = 380\,\mu\mathrm{H}$, with active cooling.

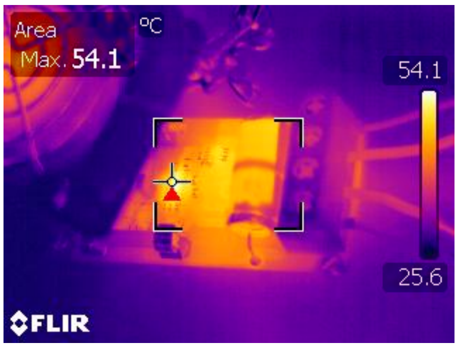


Figure 17: Thermal image of operation with $I_{LOAD} = 10\,\mathrm{A}$, $V_{BUS} = 30\,\mathrm{V}$, $L_{LOAD} = 380\,\mu\mathrm{H}$, with active cooling.

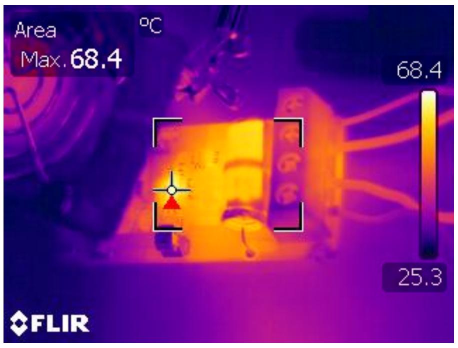


Figure 18: Thermal image of bottom of board with $I_{LOAD} = 10\,\text{A}$, $V_{BUS} = 40\,\text{V}$, $L_{LOAD} = 380\,\mu\text{H}$, with active cooling.

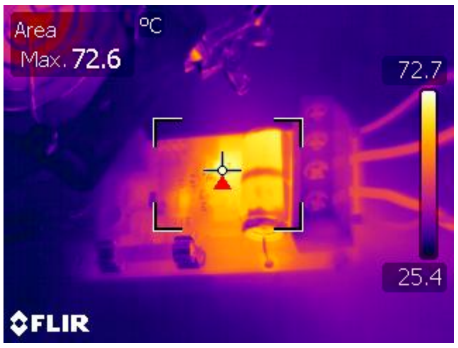


Figure 19: Thermal image of operation with $I_{LOAD} = 20\,\text{A}$, $V_{BUS} = 30\,\text{V}$, $L_{LOAD} = 380\,\mu\text{H}$, with active cooling.

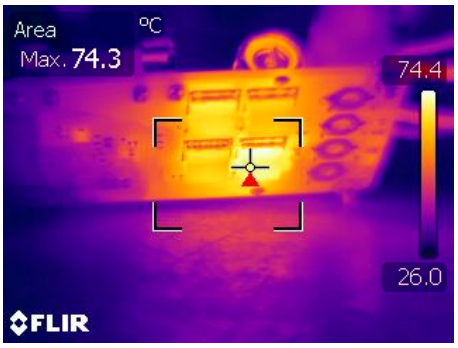


Figure 20: Thermal image of bottom of board with $I_{LOAD} = 20\,\text{A}$, $V_{BUS} = 30\,\text{V}$, $L_{LOAD} = 380\,\mu\text{H}$, with active cooling. MOSFET temperature imbalance due to low loss in load, causing the upper left and lower right MOSFET to be in the conductance region a larger proportion of time.

Operation

This section will describe how Flexible Servo Controller operates, both from an electrical and software perspective. As Flexible Servo Controller is electrically relatively simple this description will be both shorter and more detailed than the software description, which is due to its complexity at best an overview with the most important sections described in more slightly more detail.

Electrical operation

Flexible Servo Controller is, at its core, a high-current H-bridge, a current sensor, and a microcontroller reading the motor current and position (from an external incremental encoder) and controlling the H-bridge state based on these inputs. Functionally, Flexible Servo Controller can be divided into separate electrical subsections, which are described below.

Power electronics

The high-power section of Flexible Servo Controller, this is primarily located on the left side of the PCB as shown in figure 24a and the upper part of the schematic in figure 23. The input voltage, powering Flexible Servo Controller, is supplied through X1 (or through the use of ring terminals or similar connection mechanism), which is decoupled with C3, and current-limited with the fuse F1. This decoupled voltage is passed to the H-bridge, consisting of MOSFET switch transistors Q1, Q2, Q3, Q4, EMI-filters R7/C7, R6/C8, R8/C9, and current sensor IC3.

The MOSFET switch transistors are driven by the H-bridge driver (as is described below in more detail) to an essentially completely "on" or "off" state, allowing for applying both positive and negative voltages to the motor connected to X1, as well as the ability to short-circuit or open-circuit the connected motor.

IC3 generates an electrically isolated analog voltage proportional to the motor current, referenced to the ground potential used by the microcontroller (reducing error due to ground plane voltage differentials from high load currents), as well as an active low fault output that is triggered and latched when the measured current exceeds a threshold current (nominally 25 A). L5 and C13 are used to heavily decouple the power supply to IC3, which is important as the analog output voltage is proportional to the supply voltage, making it important that IC3's and the microprocessor's analog reference have closely matched voltages.

The EMI filters R7/C7, R6/C8, and R8/C9 reduce the higher switching frequencies in the output cables (which may be relatively long and may otherwise cause interference), as well as absorbing some voltage transients due to PCB trace and MOSFET package parasitic inductance.

F1 gives a hardware current failsafe that primarily is intended to reduce the risk of cable fires and severely loading the power supply supplying Flexible Servo Controller, though in some cases this fuse can also be used to protect the motor from excessive power output. Keep in mind that at low speeds Flexible Servo Controller essentially acts as a step-down (buck) converter, making it possible to have a far higher output current than input current.

C3 is important to size correctly, in particular for high output-current applications as the ripple current can be very large. As can be seen in section Typical Performance, even a capacitor with a very high ripple current rating experiences severe self-heating at high output currents.

The body diodes in the switching MOSFET's together with D1 protect Flexible Servo Controller from reverse input polarity by conducting a large current and destroying F1, while D1 keeps the rest of Flexible Servo Controller from experiencing the reverse voltage.

In general, the power-electronics section has very wide traces and uses many vias in order to reduce the conduction losses in the PCB, and as Flexible Servo Controller only uses a two-layer PCB this results in much of the bulkiness of the device.

Voltage regulation

Flexible Servo Controller has internal regulators that supply the various active electronics on-board, as well as supplying power to an external incremental encoder. The sanitized power supplied through D1 (as described previously) is fed to IC1, which is a switching step-down (buck) converter, which generates a 5 V supply for several on-board components as well as the external encoder. This converter is implemented with the structure recommended in the datasheet without any major deviations; C2 decouples the high-voltage input, R1 sets the off-time, C1 acts as the charge-pump capacitor, while D2, L1, and C6 form the step-down power electronic stage. Note that the PCB uses wide (low-inductance) traces for the latter parts, which minimizes the output ripple current. R2 and R3 feed back the output voltage for the converter IC. Note that C6 is selected to be an electrolytic capacitor (rather than a ceramic capacitor) as IC1's datasheet specifies a minimum resistance in the output decoupling capacitor.

L2 and C5 form a low-pass filter that reduces switching noise to the input of IC2, a linear regulator that generates a 3.3 V supply primarily for the microprocessor and current sensor, which is locally decoupled with C4, as well as elsewhere with C24 and C25 which are placed physically closer to the microprocessor (IC6), and C13 which is placed near the current sensor (IC3). It is important the the 3.3V supply not exhibit too much ripple, as the current sensor generates an analog voltages used by the microprocessor for measuring the output load current.

The 5 V supply is also used to power the external encoder, which is protected from a short-circuit by F2, and protected from EMI by L3, L4, C10, and C12. Note C12's relatively "odd" electrical connection, all DC return current from the encoder supply is passed back to a single connection point near the switching converter (instead of through the relatively sensitive ground plane). The relatively high inductance in these traces cause the potential for a significant voltage differential at high frequencies between the encoder ground and the ground for IC10, which may cause spurious behavior in electrically noisy environments. C13 adds a low-impedance path for these high-frequency disturbances (at cost of increased noise on the ground plane) to reduce the risk of spurious logic level transitions.

H-bridge driver

IC8 is the primary driver of the power electronic MOSFET switches, which and is supported by several passive components. The electrical connections to this IC essentially follow the "standard" connection recommended in the datasheet, without any significant deviations. C16 decouples the incoming high-voltage power, C17 decouples the internal 5 V generator, while R22 and R24 form a voltage divider that sets the MOSFET over-current fault to approximately 0.22 V (which for the typical on-resistance of 1.5 mΩ corresponds to a MOSFET current of approximately 150 A). Note that this voltage divider can/should be changed to adjust the H-bridge fault state current if other MOSFETs are used, though as the MOSFET on-resistance varies significantly from part to part this is a relatively coarse tool.

C18 decouples the regulated voltage used to drive the MOSFET gates, while C19 acts as a charge-pump capacitor for the MOSFET gate drive voltage. C20 and C22 are the charge-pump capacitors for the high-side MOSFETs Q1 and Q2, R29, R31, R36, and R39 are gate resistors that reduce any gate voltage overshoot, R34 sets the dead-time between the MOSFET on states, and R4, R5, R12, and R13 ensure that the high-power MOSFETs will tend to turn off should the H-bridge driver's outputs enter a high-impedance state. Note that C21 acts as a high-frequency decoupling between the IC8's local ground plane and the measured ground voltage for the low-side switch MOSFET's Q3 and Q4 (as is reccomended in the datasheet for IC8).

Input isolation and sanitization

Flexible Servo Controller has several "glue" ICs and digital isolation ICs that are used to interface components with different voltage levels and ensure galvanic isolation between the control signals and the high power voltage bus.

IC10 is a low-voltage hex Schmitt-inverter with 5 V-tolerant inputs that converts the 5 V logic levels from the external motor shaft encoder to 3.3 V-compatible levels for the microprocessor. The Schmitt inputs have internal hysteresis that slightly reduces the risk of erroneous logic level transitions in electrically noisy environments. D4, R17, R25, and R37 form an ESD-protection section, while R16, R23, R32 are the (default) input bias resistors, ensuring a defined logic level in the absence of a connected encoder, as well as acting as pull-up resistors for encoders with an open-drain or open-collector output. The alternate bias resistors R19, R27, and R38 are only needed for use with open-source or open-emitter encoders. Both bias resistor triplets should never be used at the same time, as this will bias the inputs to an intermediate voltage level which is very susceptible to induced EMI. C26 decouples the voltage supply for IC10.

IC7 is a TTL-compatible hex Schmitt-inverter that converts 3.3 V logic levels to 5 V-biased levels for the H-bridge driver. It is important that this IC has TTL-compatible inputs, as this will guarantee correct operation with the 3.3 V logic voltage levels generated by IC6, unlike a standard device with CMOS logic level inputs. R35 ensures that the H-bridge is placed in reset when IC6 (the microprocessor) is in a reset state or starting up, as all outputs are in a high-impedance state during this time. C23 decouples the voltage supply for IC7.

IC4, IC5, IC9, and IC11 are monolithic digital isolators that ensure galvanic isolation and also act as logic level converters for the input signals. These IC's have push-pull outputs which are referenced to each isolation side voltage supply. The right half of these IC's on the schematic are electrically connected to the microprocessor, while the left half is connected to X3. Each input on the left half is biased with a pull-up resistor (R10, R18, R26, R40), and both inputs and outputs are protected with an ESD diode / series resistor pair (D3, D4, R14, R15, R20, R21, R28, R29, R41, R42). The left side of the logic isolators must be externally powered; this external supply is filtered and cleaned with L7, F3, D6, C27, and C28. F3 and D6 act as protection against an applied reverse polarity. Note that the choice of pull-up resistors (rather than pull-down resistors) is made as the digital isolators outputs will default to a logic high state when the when the associated input is unpowered; this ensures that the motor output is disabled when either the power supply for the isolated I/O's or the electrical connection is removed when the charge pump input is disabled.

Microcontroller interface

IC6 (the microprocessor used in Flexible Servo Controller) is electrically connected following the electrical design guidelines from the manufacturer. The recommended decoupling capacitors (C15, C24, C25) have been slightly increased in size to reduce the number of different components. J1 is a 2-row 6-way pin-header for programming IC6; R33 biases the reset input; JP1 is directly connected to an I/O pin to allow for unlocking the serial input, allowing for modifying software parameters. L6 and C15 filter out EMI present on the 3.3 V bus, which is used as the analog reference; R9, R11, and C11 scale and low-pass filter the high-power bus voltage, which is then fed into an ADC input. R43/LED1, R44/LED2, and R45/LED3 are directly driven from the microprocessor outputs. Though their color can be changed keep in mind the relatively low 3.3 V bus places some limits on the color choices available.

The analog voltage generated by IC3 is fed into two different integrated analog comparators (AC1 and AC2) along with the two DAC outputs (DAC0 and DAC1, which are connected to AC3 and AC5). Though there are some options for internally routing the DAC outputs to the comparator inputs, the specific configuration desired (and elaborated more on in Software operation) requires an external connection.

Most of the digital I/O is bit-banged (set manually without using integrated hardware such as hardware PWM), except for a few exceptions;

- ENC_A and ENC_B (the motor encoder inputs) along with QUAD_A and QUAD_B (the setpoint quadrature inputs) are placed on sequential input pins in order to use the hardware quadrature decoder.

- RX and TX (the serial interface pins) are connected to port D's UART1 interface (though they could as easily have been placed on any other hardware UART interface).
- LED1, LED2, and LED3 are connected to hardware PWM outputs, though these outputs are currently only driven to a logic on or off state with the current software version.

Other uses

The electrical structure of Flexible Servo Controller allows for re-using the hardware for completely different applications (with varying degrees of software changes required). In general, the hardware can be used for many applications requiring a moderately high-power H-bridge, such as a power inverter (12 VDC → 230 VAC converter), a constant-current output source for inductive loads, a solid-state Tesla coil driver, a hard-switched medium-power inductive heating device, and so on and so forth. The relatively high degree of flexibility and calculation power in IC6 allows for many different possible applications.

Software operation

The total amount of source code used in Flexible Servo Controller is relatively large, even though only standard C and AVRLIBC libraries have been used, making it impractical to describe every function in detail in this document. What follows is first a description of each .h/.c file pair used and their purpose, followed by a more detailed description of some parts of the software. For a full understanding of the entire software operation see the source code in its entirety.

The source code .h/.c files used in Flexible Servo Controller are;

board.h This file contains all hardware definitions and low-level software definitions, such as fuse settings, ports and pins for the connected hardware, the resistor values for the R9/R11 bus voltage divider, start-up routines that configure integrated peripherals (such as the UART interface, interrupt levels, DAC output, and quadrature hardware decoding), and some utility functions to read the state of other hardware on the PCB, such as the fault-flag state of the H-bridge driver (IC8).

config.h This file contains most of the compile-time constants that are useful to modify when using Flexible Servo Controller in slightly different situations. The UART baud rate; bus voltage lockout hysteresis; charge-pump cutoff frequencies; default, minimum, and maximum parameter settings and names; plot settings; error message texts; startup messages; and so on are contained here.

main.c Primarily handling the startup and initialization of Flexible Servo Controller, this file also runs the shell interface (including running shell commands), prints event/error messages to the serial interface, runs the motor self-heating model, and sets the status LED status at the lowest priority level. Note that most of the initialization is run atomically (disabling interrupts), with only the section handling corrupt EEPROM (nonvolatile memory) with interrupts enabled.

state.h/.c These files contain primarily a state machine that reads the enable input and current environment status (such as the multiple fault inputs and bus voltages), and transitions into different device states accordingly, enabling and disabling internal functionality as applicable (such as the motor output). The different states and their triggers and effects are described in more detail below.
This file also handles the charge pump and index inputs, both of which are pin change interrupts that run at the lowest possible priority. The encoder index interrupt is called on a pin logic level change from the encoder input and directly drives the encoder output. As this interrupt is below the priority level for the PID controller there may be a significant delay between an index input logic level and an associated change on the index output. As this interrupt is driven by a logic-level change a long period of very rapid transitions will block the main loop from executing, potentially long enough to trigger the watch dog timeout, resetting the device; however, as this interrupt is below the priority for the PID and current controller interrupts the motor position and current will be maintained at safe levels during this time, in the end causing no damage to the connected hardware.
The charge-pump interrupt has a "trick" that allows for arbitrarily high input frequencies to its associated pin-change interrupt without blocking lower-priority functions; when triggered the charge pump interrupt will disable itself from being called again, and it will only be enabled again by the state-machine interrupt (which is run at a fixed period). This means that for charge pump logic level transition frequencies below the state-machine interrupt frequency all logic level transitions will be captured, while for higher frequencies some/many transitions will be ignored (interrupts will be generated at most once per state machine period). As the charge pump time-out period is far higher than the state-machine interrupt period enough of the logic level transitions are guaranteed to be captured, leading to correct behavior even at arbitrarily high input frequencies.

util.h This file contains smaller utility functions that do not in themselves warrant an entire .h/.c file pair and don't really belong anywhere else. At the moment, this is only used for setting the microprocessor clock frequency and turning the status LED's on or off.

control/cc.h/.c These files contains functions to configure the H-bridge output, and in particular an adjustable constant-current output controller that behaves similarly to a sigma-delta converter with variable switch frequency, which is described in more detail in the section Constant-current control algorithm.

control/control.h/.c These files contain functions for starting and stopping the motor position PID controller as well as routines for setting up and performing a tuning run (logging the controller setpoint, plant position, error, and individual controller coefficients). The PID and tuning implementation is described more in section PID implementation.

dynbuf/dynbuf.h/.c A general-purpose circular FIFO buffer, the functions contained in these files are used in several locations in other files where fast buffers are needed.

nv_settings/eeprom.h/.c These files contain rudimentary functions to read, modify or save the parameters defined in nv_settings.h to EEPROM. These files allow for storing two redundant copies of system parameters to EEPROM along with individual CRC checksums. This allows for a relatively simple redundant data storage mechanism that can recover the stored system settings in the event of corrupted memory. Additionally, all parameters are read and modified atomically, making them an easy way to safely pass data between different contexts (such as between an interrupt and an idle loop).

nv_settings/nv_settings.h This header file contains the definitions for the configurable system parameters, whose maximum, minimum, and default values are defined in config.h.

plot/plot.h/.c An ASCII-art based plotting library with optional Y-axis auto-scaling to "human-readable" values, tick marks, and axis markers. These files are used for the plots generated when tuning the PID controller, and are almost only pure C code and are easily ported to other platforms.

shell/command.h/.c A catch-all file pair that contains all the commands used in the shell interface along with an array storing the command name, help string, and function pointer to the command. Most of the commands used in Flexible Servo Controller are relatively simple and heavily use the functions available in the shell.h file and other source files described here.

shell/shell.h/.c These files parse CR- and/or LF-terminated (return-key) lines handled by terminal.h/.c, try to match the first word against a command stored in command.h/.c, calling the associated function if a match was found and giving easy tools to check for flags and get their arguments.

shell/terminal.h/.c These files parse the raw character-by-character input from uart.h/.c, implementing a simple input buffer for using character-based terminal programs on a host PC (such as minicom) along with up/down arrow support for recalling previous lines. These files allow for checking for the existence of a new finished line and getting the data character by character.

shell/uart.h/.c These relatively low-level files interface the UART module in the microcontroller; with an interrupt that is called when data is received, placing the new data into a buffer to be read by terminal.h/.c, sending XON/XOFF commands depending on the buffer status, and writing data to the UART transmit buffer when applicable.

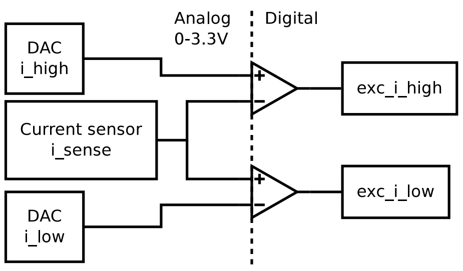


Figure 21: Analog input stage for current measurement (electrically equivalent to a 2-bit flash ADC).

stackcount/stackcount.h/.c A file pair for debugging purposes only, these files allow for checking the amount of free space on the stack, which is useful for monitoring RAM usage during development. Heavily based on code available at `http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=52249` by MichaelMcTernan, posted in 2007, with only minor modifications.

syslog/syslog.h/.c These files implement a basic system log, which logs errors to EEPROM that can be read at a later state, even after a power cycle. Errors detected by the state.h/.c file pair are call functions in these files which add data to the circular buffer and is then immediately saved to EEPROM.

Position control in Flexible Servo Controller

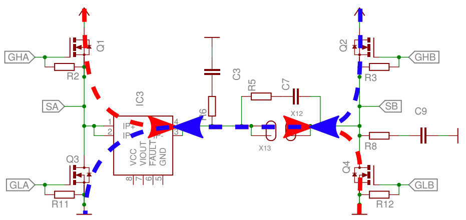
Flexible Servo Controller uses a cascaded control structure, with a high-speed current-control loop contained within a slower position control loop. The current control loop regulates and maintains a set motor current, while the position control loop regulates the motor position. These controllers are implemented with radically different algorithms which are described in more detail in the coming sections.

Constant-current control algorithm

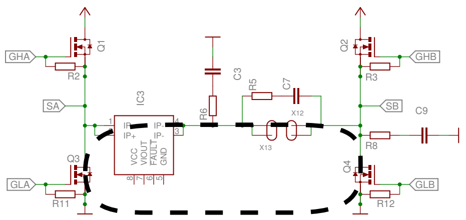
The high-speed constant-current control loop regulates the current through the connected load with feedback from the current sensor IC3 and by driving the H-bridge output state. The constant-current controller can run in an active constant-current state, which attempts to maintain a setpoint current; in a brake mode, which slows the motor connected to the output as quickly as possible while maintaining maximum current limits; and in an open mode, which effectively lets the motor “coast” naturally (equivalent to electrically disconnecting the motor from Flexible Servo Controller). Algorithm 1 shows the source code used for the constant-current control loop, which is largely only externally controlled by modification of the `cc_state` enum variable, `sat_neg` and `sat_pos` variable pair, and the DAC output levels (which are connected to the analog comparator inputs).

Algorithm 1 Constant-current algorithm source code. `sat_neg` and `sat_pos` are set automatically on every call of the PID controller loop.

```
ISR(CC_VECT){
    //Remaining number of interrupt calls to skip before output can change.
    //Note this value is preserved even when switching between states!
    static uint8_t skip_rem = 0;
    switch(cc_state){
        case cc_state_control:
            if(skip_rem){
                skip_rem--;
            }else{
                if(check_pos_AC()){
                    board_bridge_en_servo_neg();
                    skip_rem = skip_ref;
                    sat_pos = 0;
                }else if(check_neg_AC()){
                    board_bridge_en_servo_pos();
                    skip_rem = skip_ref;
                    sat_neg = 0;
                }
            }
            break;
        case cc_state_brake:
            if(skip_rem){
                skip_rem--;
            }else{
                static uint8_t output_short = 0;
                if(check_pos_AC() || check_neg_AC()){
                    board_bridge_en_brake_open();
                    if(output_short == 1){
                        skip_rem = skip_ref;
                        output_short = 0;
                    }
                }else{
                    board_bridge_en_brake_short();
                    if(output_short == 0){
                        skip_rem = skip_ref;
                        output_short = 1;
                    }
                }
            }
            break;
        case cc_state_off:
            if(skip_rem){
                skip_rem--;
            }
            board_bridge_en_brake_open();
            break;
        case cc_state_undefined:    //If output_short is undefined, set h-bridge
            into a safe output_short.
            board_bridge_en_brake_open();
            break;
        default:                    //Should never occur
            board_bridge_en_brake_open();
            return;
    }
}
```



(a) H-bridge states for forward (red) and reverse (blue) currents. In the forward state Q1 and Q4 are active (low-impedance) while Q2 and Q3 are inactive (high-impedance). In the reverse state Q2 and Q3 are active while Q1 and Q4 are inactive.



(b) H-bridge states for brake (shown) and open mode (not shown). In the brake state Q3 and Q4 are active (low-impedance) while Q1 and Q2 are inactive (high-impedance). In the open mode (not shown) all transistors (Q1, Q2, Q3, and Q4) are inactive.

Figure 22: H-bridge switch states for forward, reverse, brake, and open.

In the active constant-current state the measured current from IC3 (an analog voltage proportional to the motor current) is fed into two analog comparators on the microprocessor, as is shown in figure 21. For a given setpoint current i_{SP} the two integrated digital-to-analog (DAC) converters generate two reference voltages V_{high} and V_{low} which are equivalent to a measured current of $i_{SP} + \frac{1}{2}i_{ripple}$ and $i_{SP} - \frac{1}{2}i_{ripple}$ from the current sensor IC3 (where i_{ripple} is some non-negative ripple current). This structure causes the output from the integrated comparators `exc_i_high` and `exc_i_low` to be active when the measured current exceeds the maximum and minimum desired current respectively.

A fast (nominally 200 kHz) control loop running iat a high interrupt priority monitors the status from the comparators and switches the output state of the H-bridge to decrease the current in the motor when `exc_i_high` is active and to increase the current in the motor when `exc_i_low` is active. This is shown in figure 22a, where one mode will increase the measured current in the motor and the other mode will decrease the measured current in the motor. For large inductive loads (as is the case for DC motors) the current is well-behaved and will exhibit close to a linear ramp when switched between the two active states.

In order to limit the maximum switching frequency a lock-out timer is used that blocks the H-bridge state from changing for a small time after a change has occurred. A side effect of this is that for a reasonable lock-out timer time the desired ripple current can be set to zero, which effectively results in a motor ripple current that is as small as possible while maintaining a maximum switching frequency, which is conceptually similar to a sigma-delta converter. One effect of this method of current control is that the output switching frequency is far from fixed and tends to exhibit somewhat of a “spread-spectrum” effect where subharmonics are present, leading to a white-noise type sound from the connected motor.

As will be described later, the motor position PID controller needs some way to determine if the output from the current controller is in saturation, IE. if the current controller can not reach the set current. In this application, the output is defined to be saturated when the H-bridge output does not switch more than twice per PID controller period. As the PID controller runs significantly less often than the current controller period (nominally 1/100'th of the current controller period) this implies that the output is regarded as saturated when the H-bridge state has only changed twice over 100 calls to the current controller.

When in the brake mode the H-bridge state is set to one of the two shown in figure 22b; when the measured motor current is below `i_max` (as set in the system parameters) the H-bridge is set to the shorted state, which for a rapidly rotating motor will cause the motor current to rise; while when the measured motor current is above `i_max` the H-bridge is set to the open state, causing the motor current to decrease. Note that this effectively uses the motor as the inductor in boost converter, causing Flexible Servo Controller to generate a net positive current, which raises the bus voltage and is the reason a device capable of absorbing current (such as a braking resistor) is required for operation. Below a certain speed the measured motor current will never exceed `i_max`, allowing for the output to be continuously shorted.

Both the current sensor and the DAC outputs have a constant offset voltage which can to a certain degree be canceled out by calibration (which is performed with the software command `calibrate`); if the motor current is known to be zero (such as if the shaft is kept stationary for a period) the DAC level can be swept until the comparators trigger, giving a known offset current.

PID implementation

The slower position control PID controller regulates the motor position by reading the encoder position and generating a setpoint current which is sent to the constant-current controller. Algorithm 2 shows the core of the PID controller source code, with input prescaling, derivative filtering, integral anti-windup, and output shaft friction compensation.

The derivative filter is a first-order IIR low-pass filter algebraically equivalent to

$$d_{filt,n} = k_{df}\,(err_n - err_{n-1}) + (1 - k_{df})\,d_{filt,n-1}$$

which for high values of k_{df} causes the filtered derivative $d_{filt,n}$ to be heavily based on the most recent change in the controller error $err_n - err_{n-1}$ (equivalent to first-order backward differentiation), while lower values of k_{df} cause the filtered derivative to be more based on the previous filtered value $d_{filt,n-1}$. Though this both a relatively crude approximation of an ideal derivative as well as a relatively simple filtering method with poor stop-band attenuation the limited floating-point capabilities of the microprocessor (emulated floating-point with a native 8-bit bus) place heavy limitations on the number of operations that can be performed in the controller loop. Practical tests have shown these approximations function very well and should be "good enough" for most applications.

One oddity with the current controller implementation is that it is only known whether a specific setpoint current has caused the output stage to saturate after it has been

Algorithm 2 PID controller source code (tuning/logging and controller gain initialization removed. `inp_pow` has been defined as a `uint8_t`, while all gains, `i_max`, and `i_friction` are defined as `float`).

```
void controller_idle(void){
    // Collect plant and reference input
    uint16_t ref = board_query_ref() << inp_pow;
    uint16_t plant = board_query_enc();

    //Generate error level
    err = plant-ref;    //This wraps around nicely, giving an error in the range -
        INT16_MAX to INT16_MAX

    //Generate derivative contribution
    d_filt += kdf * (((int16_t)(err - last_err)) - d_filt); //Update derivative
        IIR low-pass filter, equivalent to d_filt = kdf * ((int16_t)(err -
        last_err)) + (1 - kdf)*d_filt
    last_err = err; //This must be after last_err is used in d_filt
    float dpow = d_filt * kd;


    //Generate proportional contribution
    float ppow = err * kp;

    //Generate integral contribution
    float ipow = (err_acc + err) * ki;

    //Generate total contribution
    out = dpow + ppow + ipow;

    //Limit to current bounds and apply output
    uint8_t sat = 0;
    if(out > i_max){
        out = i_max;
        sat = 1;
    }else if(out < -1 * i_max){
        out = -1 * i_max;
        sat = 1;
    }

    //Update accumulated error only if output not saturated (anti-windup for
        integral term)
    if(!sat && !cc_sat()){
        err_acc += err;
    }

    if(out > 0){
        out += i_friction;
    }else if(out < 0){
        out -= i_friction;
    }
    cc_control(out);
}
```

commanded[10]. This causes the integral anti-windup algorithm to "miss" the first time the output becomes saturated and erroneously accumulate the process error once, as well as "miss" the first time the output re-enters the controllable region and erroneously not accumulate the process error once. However, this effect is negligible for reasonable controller gains as the PID controller at least one order of magnitude faster (and often more) than the connected servo motor.

Note that the actual source code used differs in several ways, most significantly in that the setpoint position is actively switched between the quadrature input and a buffer with values generated by the tune command.

Flexible Servo Controller state machine

Flexible Servo Controller has a primary state machine that controls the total system state based on the enable and fault inputs, bus voltage, and so on. This is implemented with a Moore-type finite state machine using a switch/case structure running in a period medium-priority interrupt. The system states and their associated actions are;

startup The initial system state, this state does nothing and is permanently left once Flexible Servo Controller's start-up software initialization is completed, which ensures (among other things) that EEPROM parameters have been loaded to RAM before altering the system output.

idle The "ordinary" output inactive system state. This state will either enable the motor brake or motor coast output state (as configured in the system settings) and update the status and LED outputs accordingly. If a fault occurs (either determined by inputs from other integrated circuits on the PCB or internally determined based on the bus voltage or motor temperature model) Flexible Servo Controller will enter the fault state. If the enable input becomes asserted and the bus voltage is in an acceptable range Flexible Servo Controller will enable the motor position (PID) controller and enter the active state.

active The normal output active system state. This state will enable the output active LED and output. Flexible Servo Controller will enter the idle state should the enable input no longer be asserted or the fault state should a fault occur.

fault This state will immediately be entered should a fault occur, in which case the output active LED and output will be disabled and the current and position controllers will be disabled (placing the H-bridge in the open state, letting the motor coast). Flexible Servo Controller will attempt to clear any errors from the H-bridge driver (IC8) should any be present, and only when there are no more errors present will the state transition to idle_latch. Note that the current sensor's (IC3) fault output latches to an active state when the measured current exceeds a maximum internal current, which can only be cleared by power cycling the device, meaning

[10]The saturation current varies heavily with the speed of the motor and the supplied voltage, both of which will change greatly during operation making the specific level difficult to determine beforehand.

that should this error occur the only way to clear the error is by power cycling Flexible Servo Controller in entirety[11].

idle_latch This state is only entered from the fault state and keeps the output in the defined device-inactive state (brake or open) which is maintained as long as the enable input is held active. Only when the enable input is inactive will the state transition to idle. This state acts as a lock so that should a fault occur in the active state that the enable input must be inactivated and activated again in order to transition back to the active state.

The primary state machine source code is shown below in its entirety;

```
void state_update(void){
    switch(state){                          //Main state machine
        case state_startup:                 //Default state on startup
            board_clr_active_out();         //Clear the enable output...
            board_clr_led(board_led_en);    //...and enable LED.
            if(initialized){
                state = state_idle;         //If initialized, transition to the "
                    normal" state
            }
            break;
        default:          //state should always be one of the specified states...
            for(;;){};  //...if not then hang the device as something very bad has
                happened
            break;
        case state_idle:    //"Ordinary" system state when output not active
            if(setting_get_int_idx(brake_idx,SETTING_VAL)){
                cc_brake(); //Apply the brake if enabled in settings...
            }else{
                cc_disable();           //...otherwise open-circuit the motor.
            }
            board_clr_active_out();     //Clear the enable output...
            board_clr_led(board_led_en);//...and enable LED.

            if(query_faults()){         //If there's a fault...
                state = state_fault;    //...set the state to fault...
                controller_disable();   //...disable the position controller...
                cc_disable();           //...and the current controller...
                board_set_fault_out();  //...and set the fault output.
            }else if(query_enable_input() && query_vbus_go()){ //If the enable
                pin is high and the bus voltage is OK...
                state = state_active;   //...set the state fo active...
                add_msg(state_idle2active); //...write the idle-to-active message
                    ...
                controller_enable();        //...and (re-)enable the controller. (
                    Clearing any terms with memory)
            }
            break;
        case state_idle_latch:       //If we are transitioning to the idle state...
            if(setting_get_int_idx(brake_idx,SETTING_VAL)){
                cc_brake();          //Appy the brake if enabled...
            }else{
                cc_disable();        //...otherwise just open-circuit the output.
            }
```

[11]IC3 can not be directly driven from an I/O pin from the microprocessor as the supply voltage to IC3 would fall significantly from the bus voltage (on the order of 50-100mV), causing significant measurement errors as the analog output voltage is directly proportional to the supply voltage. In normal operation and when properly configured the fault output on IC3 will never trigger, making this a non-issue.

```
            board_clr_active_out();      //Clear the active output...
            board_clr_led(board_led_en);//...and active LED.

            if(query_faults()){          //If there's a fault now...
                state = state_fault;     //...set the state to fault...
                controller_disable();    //...disable the position controller...
                cc_disable();            //...and current controller...
                board_set_fault_out();   //...and set the fault output.
            }else if(!query_enable_input()){//Only if the enable input is *not*
                set...
                state = state_idle;      //...do we move to the "normal" idle state
            }
            break;
        case state_active:               //If we're in the active mode...
            board_set_active_out();      //...show this on the output...
            board_set_led(board_led_en);//...and light the enable LED.
            if(query_faults()){          //If there is a fault...
                state = state_fault;     //...transition to the fault state...
                controller_disable();    //...disable the position controller...
                cc_disable();                //...and the current controller...
                board_set_fault_out();       //...and set the fault output.
            }else if(!query_enable_input()){//If the enable input is not set
                anymore...
                add_msg(state_active2idle); //...write the active-to-idle message
                    ...
                state = state_idle_latch;    //...go to the idle_latch state...
                controller_disable();        //...and disable the position
                    controller. (the current controller will be set correctly on
                    the next call of state_update()).
            }else if(query_vbus_undervoltage()){     //If there's a bus
                undervoltage...
                add_msg(state_undervoltage);//...write this...
                state = state_idle_latch;    //...go to the idle_latch state...
                controller_disable();        //...and disable the position
                    controller. (the current controller will be set correctly on
                    the next call of state_update()).
            }else if(query_over-current()){ //If there's a long-term over-current
                present...
                add_msg(state_over-current);//...write this...
                state = state_idle_latch;    //...go to the idle_latch state...
                controller_disable();        //..and disable the position
                    controller. (the current controller will be set correctly on
                    the next call of state_update()).
            }else if(query_tracking_error_exceeded()){  //If the tracking error is
                exceeded...
                add_msg(state_track_error); //...write this...
                state = state_idle_latch;    //..go the the idle_latch state...
                controller_disable();        //..and disable the position
                    controller. (the current controller will be set correctly on
                    the next call of state_update()).
            }
            break;
        case state_fault:   //If there was a fault detected in some other state...
            board_clr_active_out(); //Clear the active output...
            board_clr_led(board_led_en);//...and active LED...
            controller_disable();     //...and disable the position controller...
            cc_disable();             //...and current controller.
            if(ff_state != FF_NOMINAL){ //If the fault was from the H-bridge...
                board_bridge_reset();    //...clear them.
            }
            if(!query_faults() && !query_enable_input() && query_vbus_go()){
                        //If there are no more faults present...
                state = state_idle_latch;      //...go to the idle_latch state...
                board_clr_fault_out();         //...and clear the fault output.
            }
            break;
    }
}
```

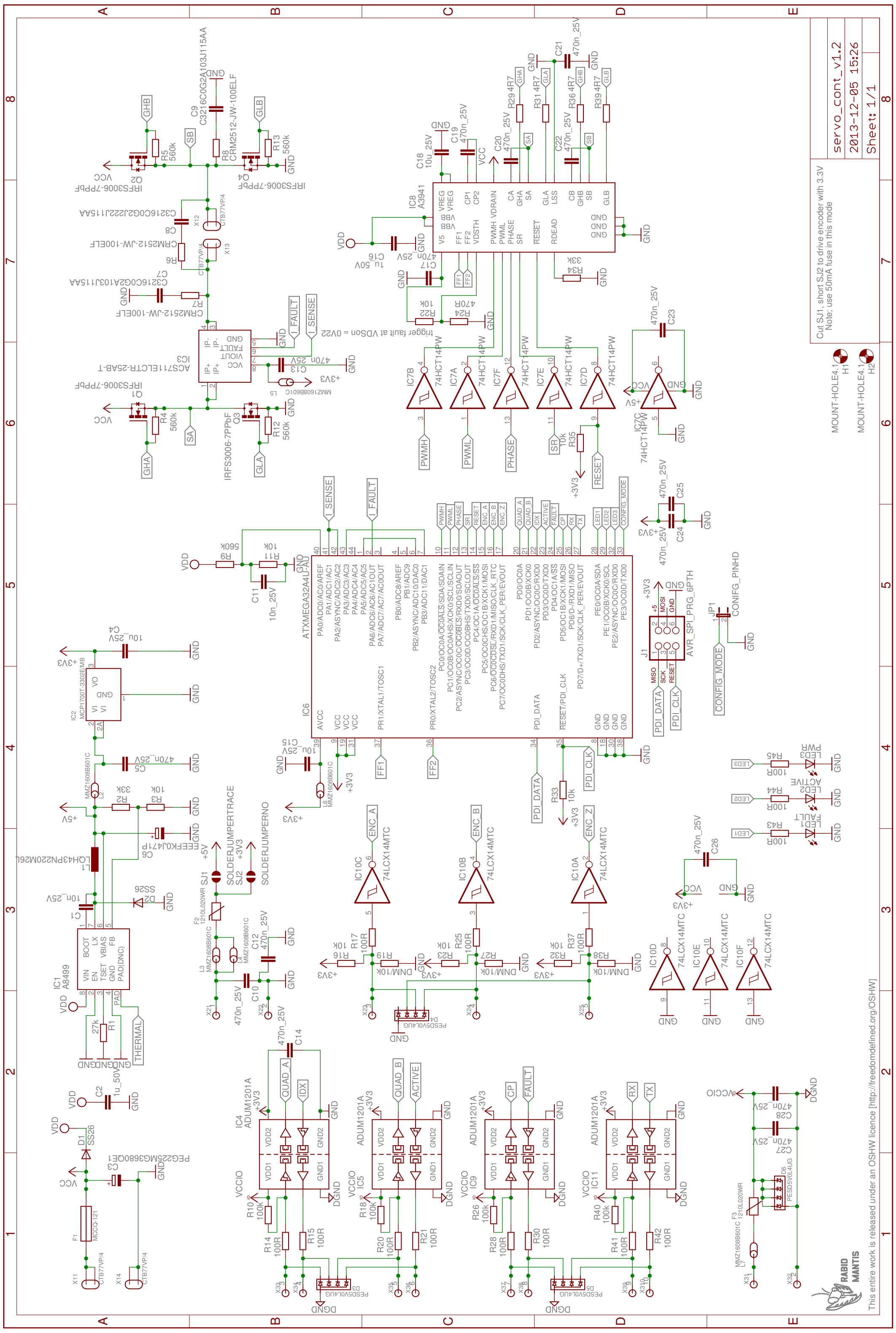
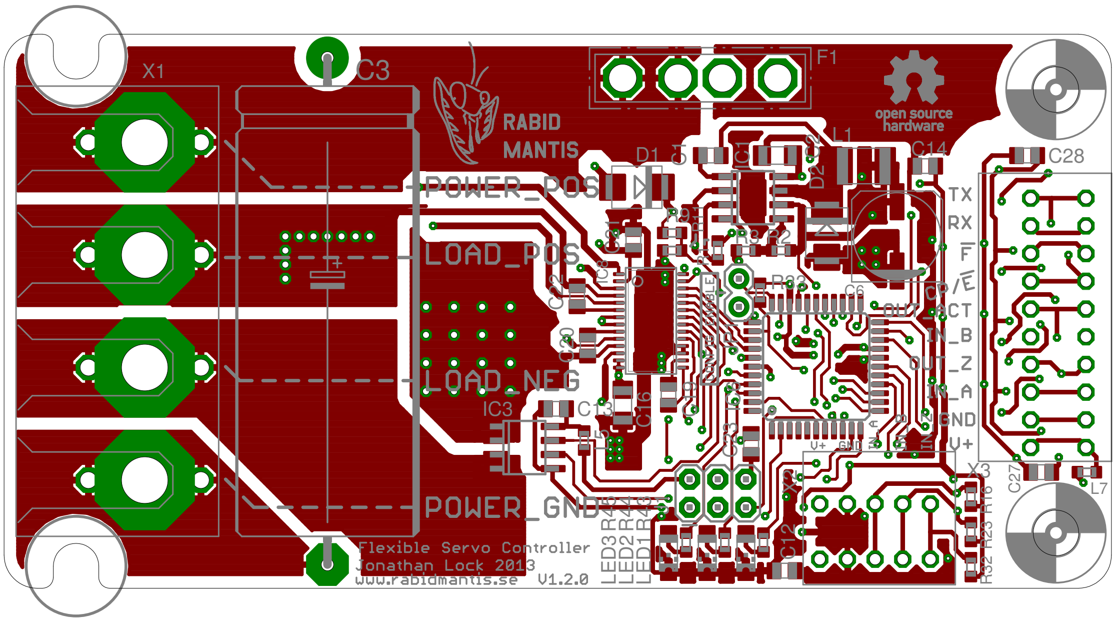
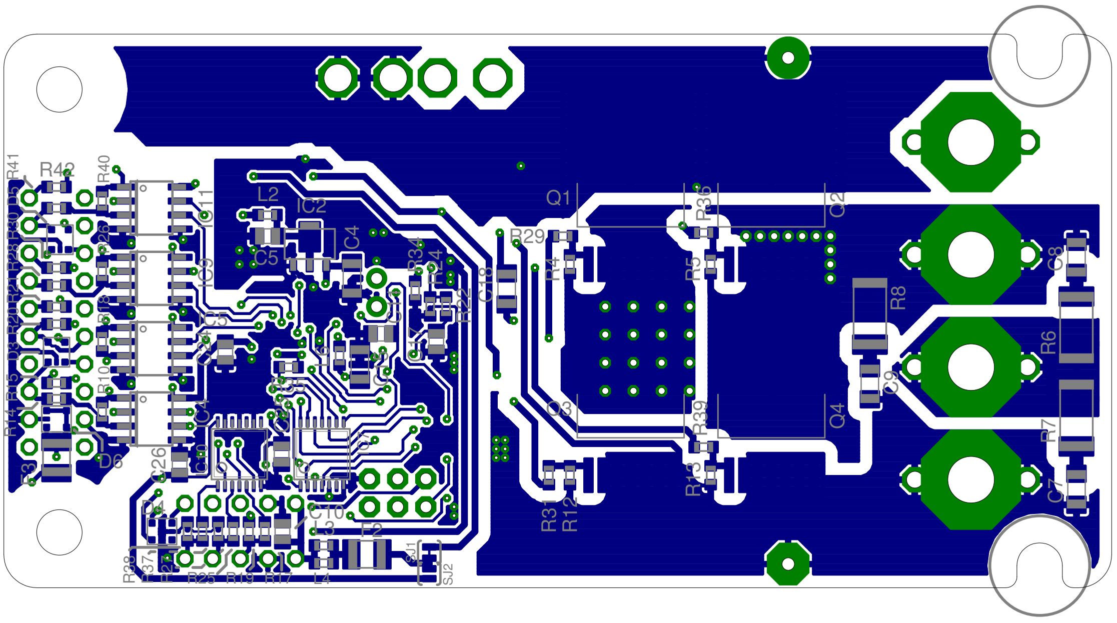


Figure 23: Flexible Servo Controller schematic.

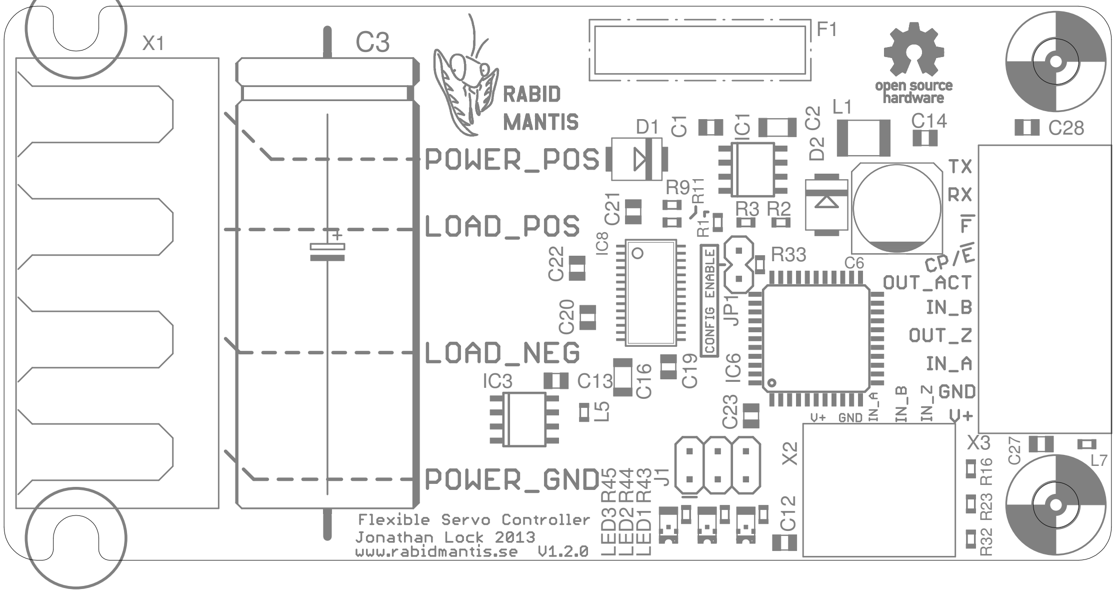


(a) Complete top layer as seen from above.

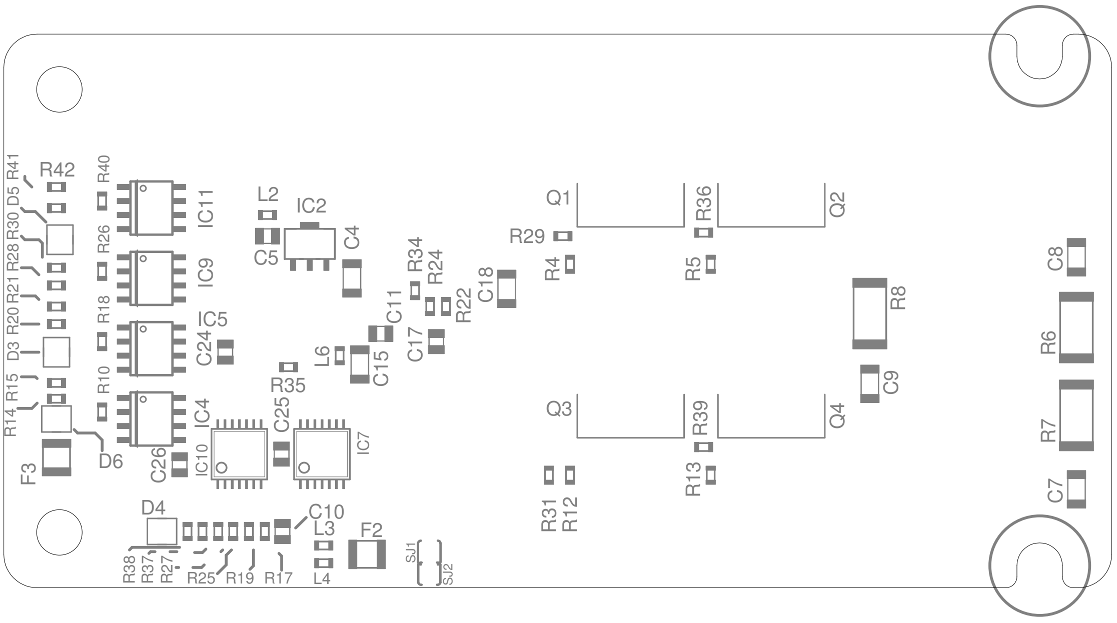


(b) Complete bottom layer as seen from below.

Figure 24: PCB details.



(a) Top layer component outline as seen from above.



(b) Bottom layer component outline as seen from below.

Figure 25: Component placement details.