

SERVOSILA Motion Controller Instruction Manual

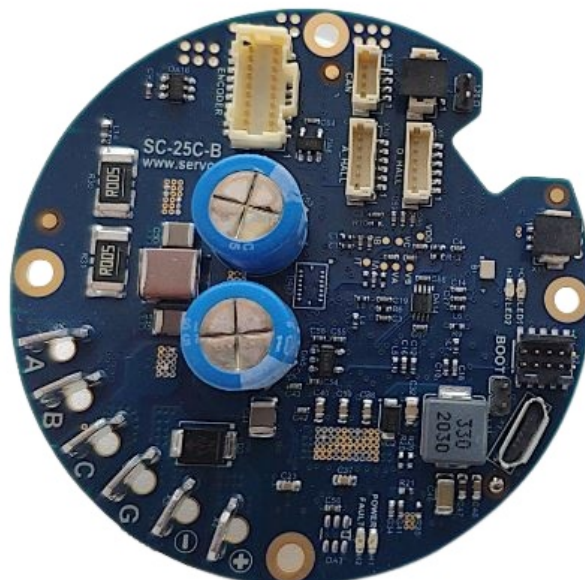
[Home](#) » [SERVOSILA](#) » SERVOSILA Motion Controller Instruction Manual 

Contents

- [1 SERVOSILA Motion Controller](#)
- [2 About Servosila Motion Controller](#)
- [3 Software Architecture](#)
- [4 Servopilot DLL API](#)
- [5 Example Application in Python](#)
- [6 Documents / Resources](#)
 - [6.1 References](#)

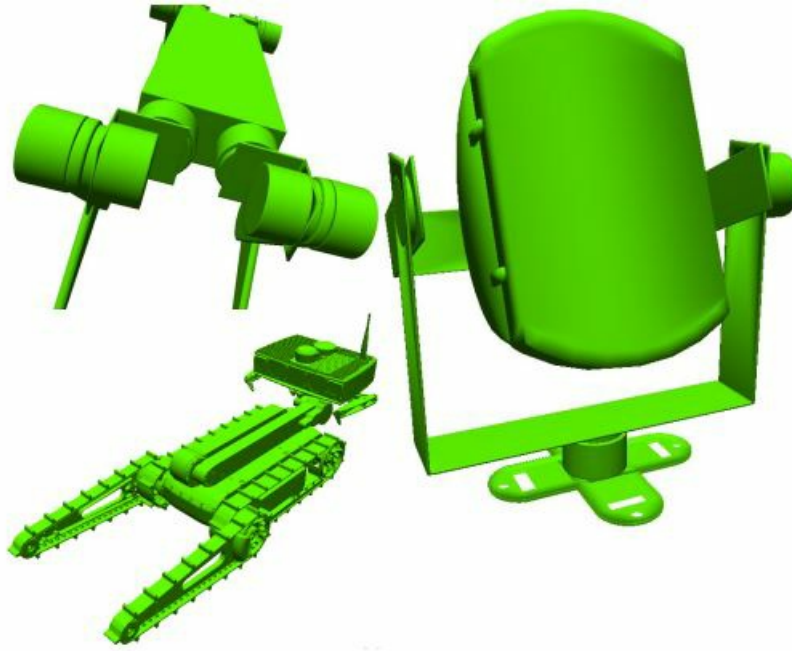


SERVOSILA Motion Controller



About Servosila Motion Controller

Servosila Motion Controller is embedded software for controlling motion of modern multi-axis robotic systems. The software runs on Linux, Windows or as a firmware on embedded MCUs.

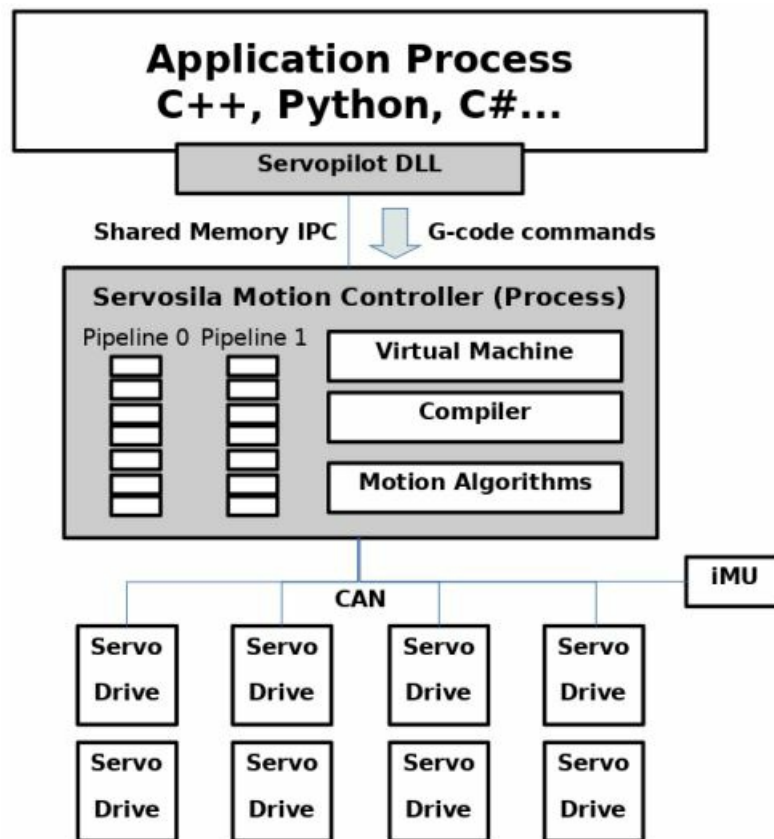


Servosila Motion Controller uses G-code for the following purposes:

- as a way to define geometry of coordinated motions in a text format,
- as a high-level communications protocol between the Motion Controller and higher-level user applications,
- as a simple scripting language for programming multi-axis robotic systems,
- as a target language for generative AI and LLMs.

Software Architecture

Servosila Motion Controller, shown as a gray box on an architecture diagram below, runs as a background process in Linux or Windows. The process communicates to servo drives via CAN or USB network. The Motion Controller process provides a low-latency shared memory interface for inter-process communication with a single Application Process, shown as a white box on the diagram. To abstract away complexities of the shared memory interface, a dynamically linked library called Servopilot DLL, is provided with the Motion Controller. The “thin” DLL exposes a much simpler API (as compared to the shared memory interface) for submitting G-code commands to the Motion Controller and for receiving telemetry and status information back. The DLL API is described in this document.



Internally, the Motion Controller process has a Pipeline 0 and a Pipeline 1 for receiving G-code commands from an Application Process. The pipelines are cyclic buffers of a fixed size. There are two pipelines so that two independent streams of G-code commands can be executed in parallel if needed. When submitting a new G-code command, an Application Process may choose to append the command to one pipeline or the other or to replace commands in a pipeline with a new set of commands. The G-code commands are executed by a Virtual Machine, an internal component of the Motion Controller process. Since G-code commands are submitted by an Application Process in a text form, there is an internal Compiler that translates the text into an internal binary code understood by the Virtual Machine. It is possible to submit a single-line G-code command as well as the text of a complete G-code program. The Compiler processes the texts line-by-line and pushes the commands into pipelines for execution by the Virtual Machine. The Virtual Machine processes the pipelines on a “first in, first out” basis (FIFO). The Motion Controller has a configurable control loop frequency (e.g. 500 Hz) that governs the performance of the entire motion control system. A single Application Process is allowed to attach to a single Motion Control process only. It is possible to run multiple Motion Control processes by assigning them unique `shared_memory_id`’s. The Servopilot DLL API is not thread-safe or re-entrant. The application programs can be written in any programming language that supports loading dynamically linked libraries (DLL). The typical choices are C++, Python, C#, MATLAB, and LabView.

Servopilot DLL API

Connecting to Motion Controller

- extern “C” bool connect(int shared_memory_id);
- extern “C” bool disconnect();

The `connect()` function attaches Servopilot DLL to a shared memory segment used for interprocess communication between an Application Process and the Motion Controller process. The Motion Controller process has a unique `shared_memory_id`, a preconfigured integer number. The function takes this ID as the only argument. The function returns true if a shared memory segment with the given identifier has been successfully attached to. The `disconnect()` function detaches Servopilot DLL from the shared memory segment. Calling this routine at the end of an Application Process is optional.

Submitting G-code commands to Motion Controller process

- extern "C" bool gcode(const char* program_text);
- extern "C" bool gcode_replace(const char* program_text);
- extern "C" bool execute (const char* program_text);
- extern "C" bool execute_replace(const char* program_text);

The gcode() function pushed a G-code command or multiple commands to the pipelines of the Motion Controller process. All previous G-code commands that have been sitting in the pipelines are preserved. A pipeline is FIFO cyclic buffer. The new commands are appended to the pipelines to be executed after all the previous commands from the pipeline have been executed. The function does not wait for the commands to be actually executed; it just pushes the commands to the pipelines and returns. The gcode_replace() function first clears all the pipelines and then pushes new commands into the pipelines for priority execution. As a result, the Motion Controller interrupts all ongoing motions and immediately continues with new motions defined by a newly received set G-code commands. Such a replacement can be done with the speed of control loop frequency, e. g. for torque-based control. The function does not wait for the new commands to be actually executed. The execute() function is the same as gcode() except that the function's call returns only once the new G-code commands have been executed. Note that all previous commands that have been already sitting in the pipelines are executed first. An advantage of execute() over gcode() is in the simplicity of application control flow. A disadvantage is that an execute() call may stall the Application Process for an extended period, for as much time as it takes to execute all G-code commands from all the pipelines. The execute_replace() function is the same as the gcode_replace() function except that the function's call returns only after the newly submitted commands have been executed.

	Clears the pipeline of previous G-code commands	Waits until the new commands are actually executed thus stalling the <i>Application Process</i>
gcode()	No	No
gcode_replace()	Yes	No
execute()	No	Yes
execute_replace()	Yes	Yes

All the functions return true if the new G-code commands have been successfully submitted to the pipelines of the Motion Controller process for execution.

Process Synchronization

- extern "C" int synchronize();

The synchronize() function allows the Application Process to wait until the Motion Control process finishes executing all previously submitted G-code commands. The function's call stalls until all pipelines are empty. This call may stall the Application Process for an extended period, for as much time as it takes to execute all G-code commands from all the pipelines.

Process State Management

- extern "C" bool pause();
- extern "C" bool resume();
- extern "C" bool reset();

- extern "C" int get_mode();

These functions manage the state of the Motion Controller process. The pause() function temporarily suspends the execution of G-code commands by the Motion Controller process. This function is used to pause operation of a robotic system. The operation is restarted using a resume() call.

Result of a get_mode() call	A corresponding mode of operation of the Motion Controller
0	OFF
1	PAUSED
2	FAULT
3	RUNNING

Axes Telemetry

- extern "C" double get_axis_cursor(int axis_number);
- extern "C" double get_axis_position(int axis_number);
- extern "C" int get_axis_work_zone_count(int axis_number);
- extern "C" int get_axis_fault_bits(int axis_number);
- extern "C" bool is_axis_online(int axis_number);

The get_axis_cursor() function returns a rotary or an angular reference position that the Motion Controller transmits as a commanded position to the axis at a particular moment. In contrast, get_axis_position() returns an actual telemetry position of the axis, where the axis physically is at a given moment. Since axes have physical inertia, the G-code program's virtual "cursor" axis position is usually ahead of the physical axis' actual position. The results of both get_axis_cursor() and get_axis_position() are returned in millimeters or degrees depending on the axis type (linear or rotary). The get_axis_work_zone_count() function returns the telemetry axis position in encoder counts. The get_axis_fault_bits() function returns a Fault Bits data received from the axis via telemetry. Refer to your servo drive's Device Reference for information on how to interpret the returned value. Zero (0) means "No Fault". Non-Zero (!=0) means some sort of a fault in the axis. The is_axis_online() function tells whether or not the axis' servo controller is broadcasting telemetry on the CAN or USB network.

Example Application in Python

```
# This is a sample program that accompanies Servopilot PLC software for Servosila Brushless Motor Controllers.
# https://www.servosila.com/en/motion-control
#

import ctypes
import time

# Before running this program:
# 1. Make sure the script is located in the same directory as servopilot-client.dll, or change the path to the DLL in the code below.
# 2. Make sure a Servopilot process is running and is connected to the same network (CAN or USB/Serial)
#    that your servo drive (axis) is connected to.
# 3. The Servopilot process
#    a) must have its Shared Memory Server API enabled in its configuration, so that the DLL could connect to the Servopilot process.
#    b) must have at least 1 axis (servo drive) added to the configuration.

# First, load Servopilot Client DLL.
# If this fails, check the path to the DLL.
servopilot = ctypes.CDLL("./servopilot-client.dll")

# Second, connect the Servopilot Client DLL to the running Servopilot process via its Shared Memory API.
# The only parameter (Shared Memory ID) must be the same as configured in the running Servopilot process.
# If this fails, make sure that a Servopilot process is running, that Servopilot has its Shared Memory API enabled,
# ...and that the Shared Memory ID in the call below matches the one configured in the process.
# Returns false (0) if there is a problem, or true (1) if the connection attempt is successful.
connection_result = servopilot.connect(1)

#Handling connection result
if (connection_result==0) :
    print("Cannot connect to the Servopilot process. Connection result is "+str(connection_result))
    quit()


#Example starts here
#
#Gradually increase speed from 0 RPM to 3000 RPM.
for rpm in range(0, 3050, 50):
    # Formulating a G-code string that commands speed.
    # For example, G1030 [0]1000 means "send Electronic Speed Control 1000 RPM command to Axis 0"
    command = "G1030 [0]"+str(rpm)
    print(command)
    # This line submits the G-code command to the Servopilot process for execution,
    # ...but does not wait until the command is actually executed.
    # Note that the string's encoding needs to be changed to ASCII
    #...before submitting the string to the DLL as that is how the DLL wants it to be encoded.
    servopilot.gcode(command.encode("ascii"))
    # Letting the motor run for 1 second at the commanded speed before looping and increasing the speed again.
    time.sleep(1)

# This (optional) G-code de-energizes and resets all axes (motors).
# Otherwise, the motor just keeps running after the program's execution is finished.
# Note the symbol "b" before the string.
# The symbol tells Python that the string constant needs to be encoded as ASCII binary array of characters.
# This is how the DLL wants it to be encoded.
servopilot.gcode(b"G1000")

#The End :-)
```




- Visit us at www.servosila.com/en/motion-control
and
- YouTube: <http://www.youtube.com/user/servosila>
- www.servosila.com

Documents / Resources

	<p>SERVOSILA Motion Controller [pdf] Instruction Manual Motion Controller, Controller</p>
---	--

References

-  [Servosila](https://www.servosila.com)

-  [Servosila](#)
-  [Brushless/Brushed Motor Controllers - Servosila](#)
-  [Brushless/Brushed Motor Controllers - Servosila](#)
- [User Manual](#)