MotionProX

The Ultimate Precision Motion Control Solution for Electric Axes

Contents

Description

The MotionProX controller is a versatile and advanced technological PLC function block designed for precise position, speed and acceleration control of an electrical axis. This function plays a crucial role in determining the desired rotational speed (setpoint) for an electrical motor, ensuring that the linear axis achieves the specified position, speed and acceleration set values, depending on the selected operational mode.

This function can be operated in various modes, including absolute movement mode, relative movement mode, jog mode and motor speed mode. The motor speed control mode is particularly suited for applications where position feedback to the PLC program is not required.

Each mode is tailored to specific motion control needs, providing flexibility and precision in a wide range of applications. Additionally, the function is equipped with a stop input, allowing for an immediate motor stop with an adjustable deceleration rate at any given moment, ensuring safety and control.

The MotionProX function is designed with separate inputs for setpoints of position, speed, acceleration and jerk for each operational mode. This allows for highly customized and flexible control configurations tailored to specific application needs.

At the core of this function is an internal trajectory generator, which enables highly accurate and smooth control of the linear axis's position, speed and acceleration. This sophisticated technology ensures optimal performance, making it an essential component in advenced motion control systems.

The diagram illustrates how the MotionProX function block integrates into the overall control system, highlighting its role in achieving precise and reliable position control of the linear axis.

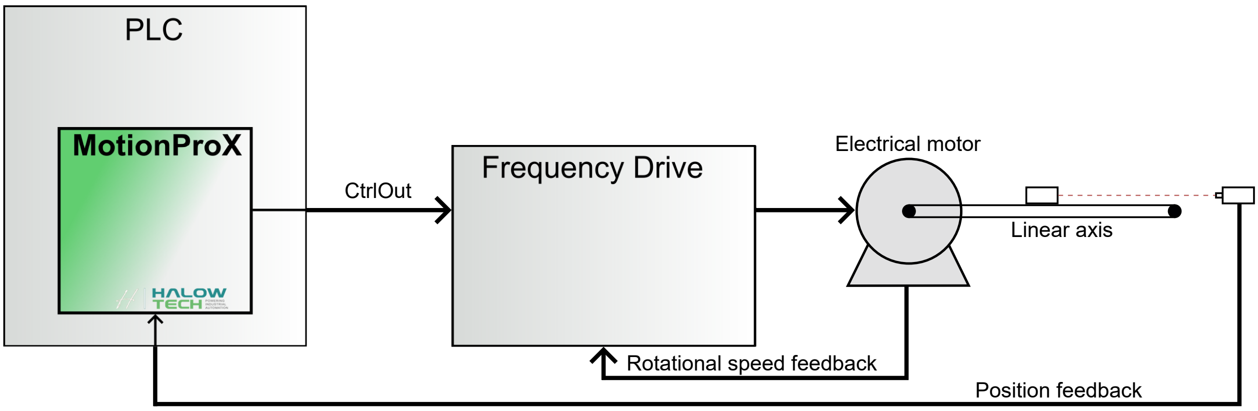
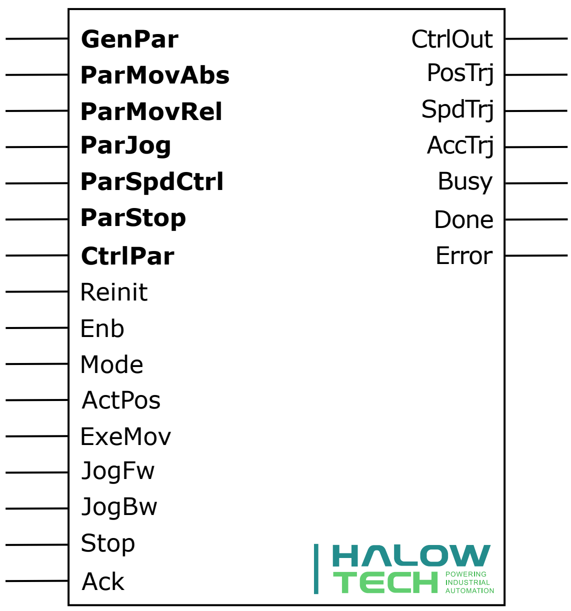


Figure 1: Integration of the function into the overall control system.

Compatible with various PLC platforms like Siemens S7, Siemens TIA Portal, Rockwell Studio AIO, Rockwell RSLogix 5000, Rexroth IndraWorks, B&R Automation Studio, and Beckhoff TwinCAT, MotionProX provides the same high performance on any platform.

Block diagram



Inputs

GenPar		
	SampleTime	<REAL>
	MaxOut	<REAL>
	MinOut	<REAL>
	SupImpCtrl	<REAL>
	Digits	<INT>

GenPar → SampleTime - Calling frequency of the controller, <REAL>

The sample time, in seconds, of the cyclic interrupt task of the PLC at which the function is running. A higher sampling frequency allows for more precise control.

GenPar → MaxOut - Maximum Output, <REAL>

Is used to define the upper limit of the *CtrlOut* output, which represents the speed setpoint for the electric motor.

GenPar → MinOut - Minimum Output, <REAL>

Is used to define the lower limit of the *CtrlOut* output to a specific value.

GenPar → SupImpCtrl - Suppress internal trajectory generator, <REAL>

The *SupImpCtrl* input is intended for applications where multiple axes need to be synchronized in position or speed or where the torque of the electric motor must be manipulated by a superimposed controller. This input allows an external controller to manipulate the position or speed of the linear axis or the torque of the motor, enabling precise coordination and adaptive control in complex systems.

The schematic of the signal flow, as depicted in figure 2, illustrates how the *SupImpCtrl* input interacts with other components within the control system, ensuring that the external controller's commands are effectively integrated into the overall motion control process.

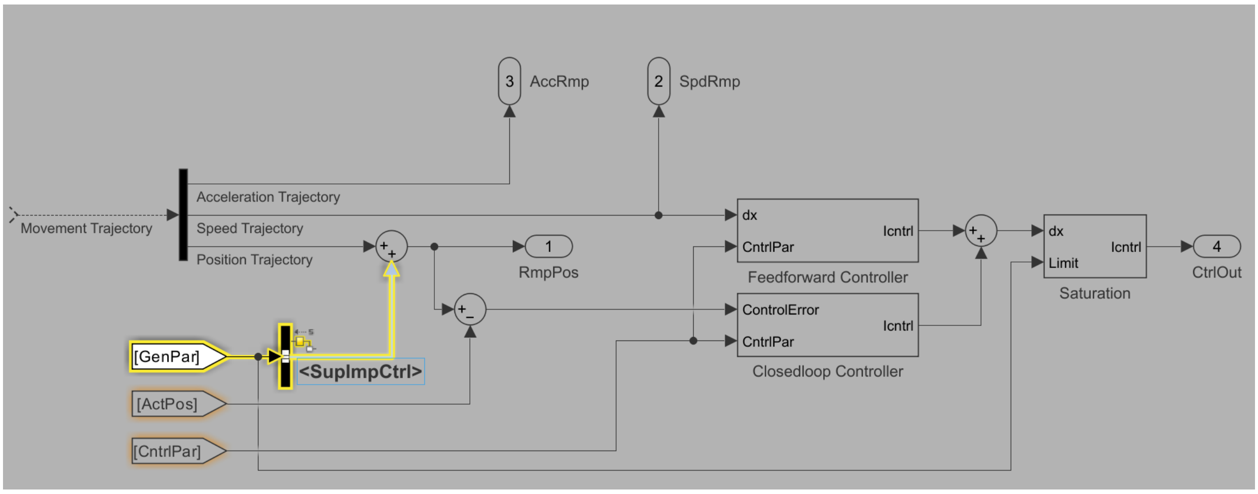


Figure 2: Schematic of the controller signal flow

GenPar → Digits - Resolution of the calculated movement trajectory, <INT>

The *Digits* input is an integer value that determines the resolution of the decimal places for the calculated movement trajectories. This setting controls the precision of the trajectory calculations, ensuring accurate motion control. A value of 4 is recommended, which provides a good balance between precision and computational efficiency, allowing the function to generate smooth and accurate movement paths.

ParMovAbs		
	TgtPos	<REAL>
	MaxSpdSetPnt	<REAL>
	MaxAccSetPnt	<REAL>
	JerkSetPnt	<REAL>

ParMovAbs → TgtPos - Position setpoint for absolute movement, <REAL>

The input allows users to specify the desired target position for the linear axis. It can be defined in any unit appropriate for the application (e.g. meters, centimeters, millimeters, micrometer). When the *ExeMov* input recognizes a rising edge, the function generates a trajectory from the current position to the specified target. The internal position controller ensures that the linear axis follows this trajectory accurately.

To operate the function in absolute move mode, the *Mode* input must be configured with a value of 1. This setting ensures that the function interprets the **ParMovAbs** input as setpoint.
Important: It is essential to understand that any changes of the *TgtPos* , *MaxSpd* , *MaxAccSetPnt* and *JerkSetPnt* during axis movement are internally ignored by the function block. The function block responds to changes only when *PosTrj* matches *TgtPos* . The *Busy* ouput serves as an indicator of the block's readiness: When the *Busy* output of the block is logicaly True , it indicates that the axis is in motion and the controller will not respond to new setpoints. Conversely, if it is logically False , the axis is in steady state, allowing the controller to react to new setpoints. See figure 3.

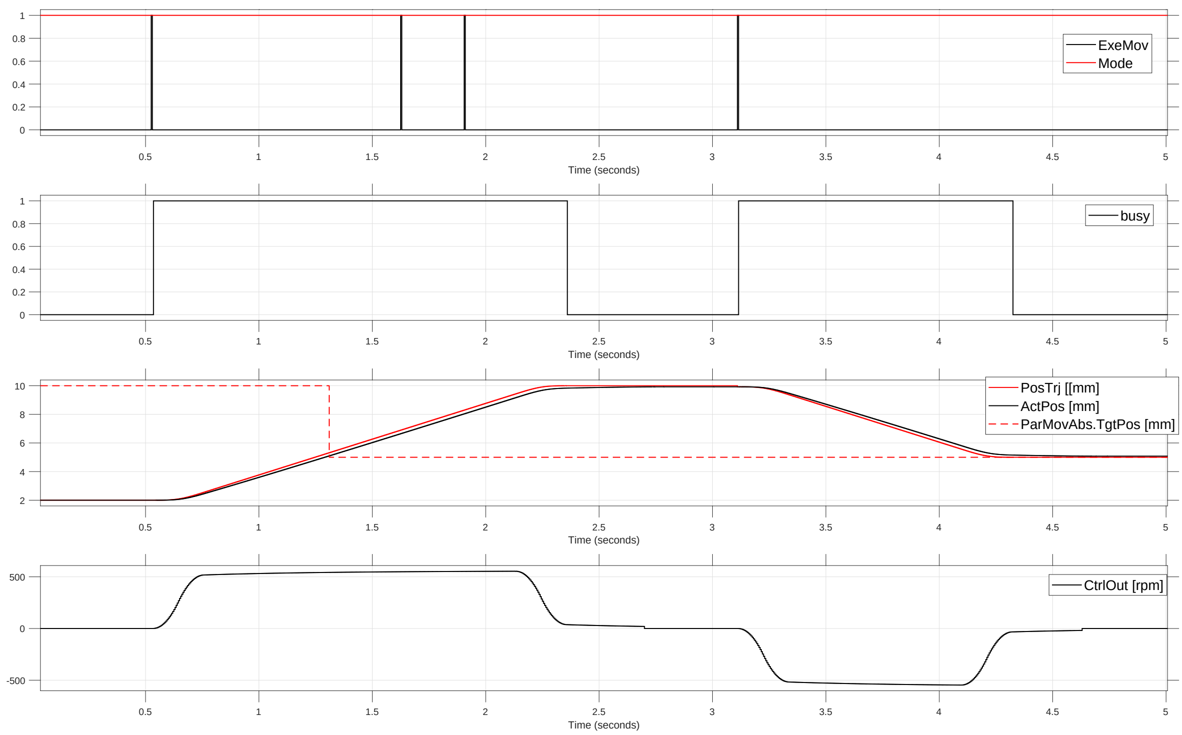


Figure 3: Position trajectory, actual position and busy signal during movement.

ParMovAbs → MaxSpdSetPnt - Maximum speed setpoint for absolute movement, <REAL>

This input sets the maximum allowable speed of the linear axis during the movement. It is defined in the same unit as the *TgtPos* input, per second. This parameter ensures that the linear axis moves at a controlled speed while reaching the target position defined by *TgtPos* . This sets the top speed for the linear axis during movement. See figure 4.

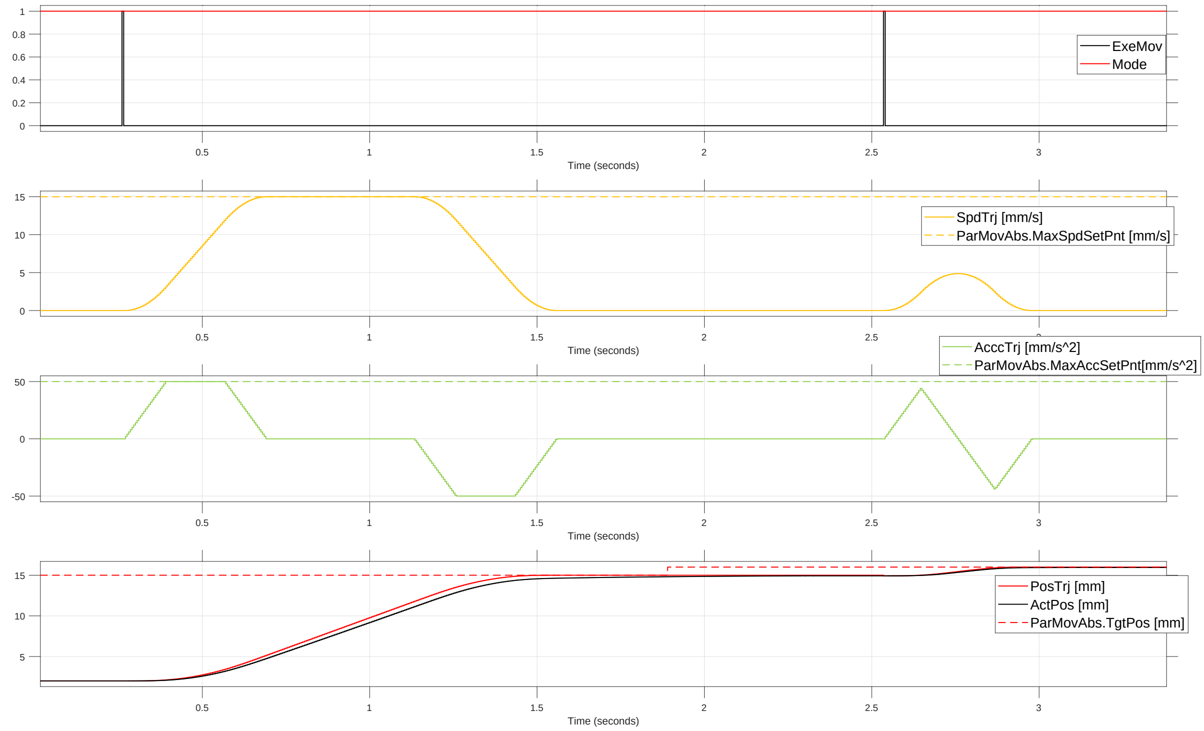


Figure 4: Upper limit of the speed and acceleration trajectory.

ParMovAbs → MaxAccSetPnt - Maximum acceleration for absolute movement, <REAL>

This input determines the maximum allowable acceleration of the linear axis during the movement. It is measured in the same unit as *TgtPos* per second squared. This parameter needs to be customized based on the specific system requirements. *MaxAccSetPnt* plays a critical role in regulating the acceleration of the linear axis as it follows the trajectory towards the target position.

ParMovAbs → JerkSetPnt - Jerk setpoint for absolute movement, <REAL>

JerkSetPnt controls the abruptness or smoothness of motion by regulating the rate of change of acceleration. It is measured in the same unit as *TgtPos* per second cubed. Adjusting *JerkSetPnt* allows users to customize the motion profile according to their desired level of abruptness or smoothness. For example, if the maximum acceleration needs to be achieved within half a second, the *JerkSetPnt* value should be set to double the maximum acceleration value. See figure 5.

Setting a jerk value as a setpoint is important for controlling the smoothness of acceleration and deceleration transitions. By managing the rate at which acceleration changes, users can prevent sudden, harsh movements that could cause mechanical wear or instability in the system. This ensures a smoother and more controlled operation, which is especially beneficial in applications where precision and mechanical longevity are critical.

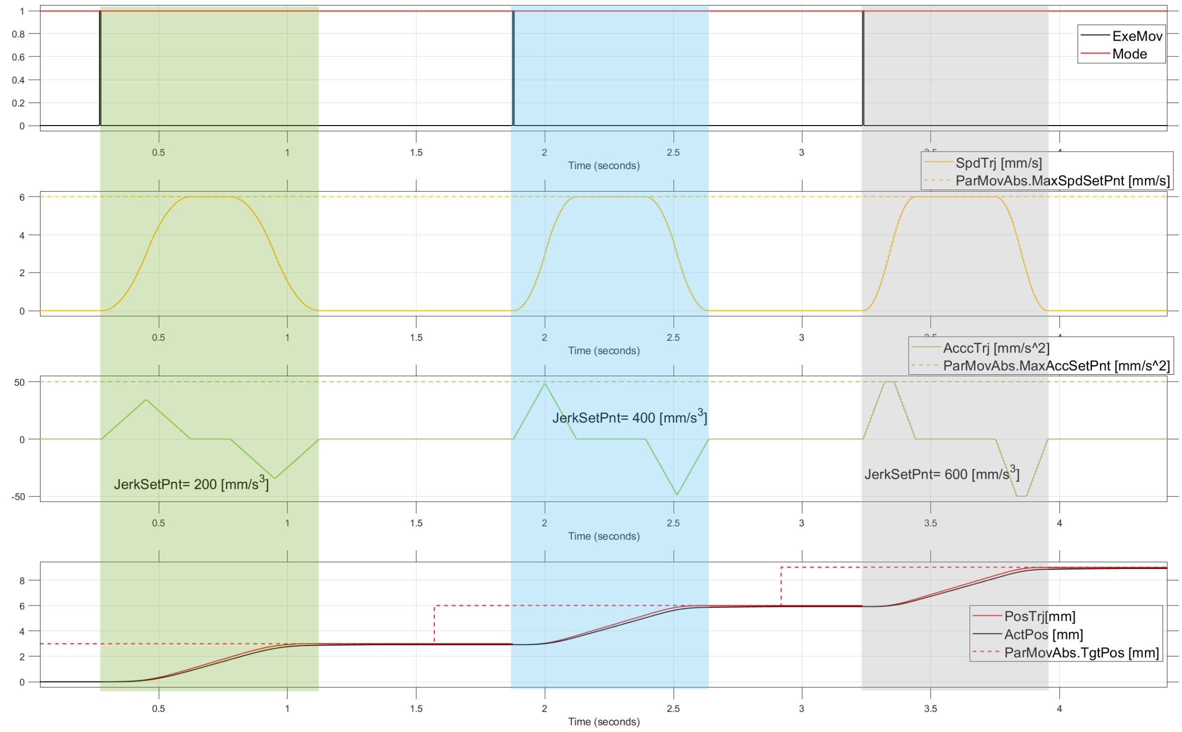


Figure 5: Movements with different jerk setpoints.

ParMovRel		
	Distance	<REAL>
	MaxSpdSetPnt	<REAL>
	MaxAccSetPnt	<REAL>
	JerkSetPnt	<REAL>

ParMovRel → Distance - Distance for the relative movement, <REAL>

The *Distance* input defines the distance the linear axis should move in relative movement mode, similar to the *TgtPos* input of the struct **ParMovAbs** in absolute movement mode, but with the key difference that it specifies the relative distance to be traveled rather than an absolute position. A positive value causes forward movement, while a negative value results in backward movement. The motion is initiated when a rising edge is detected on the *ExeMov* input. It is important to note that any changes to the parameters set by *Distance* during the motion will not be processed by the function until the current movement is completed. To operate the function in relative move Mode, the *Mode* input must be configured with a value of 2. This setting ensures that the function interprets the *Distance* input as the setpoint. See figure 6.

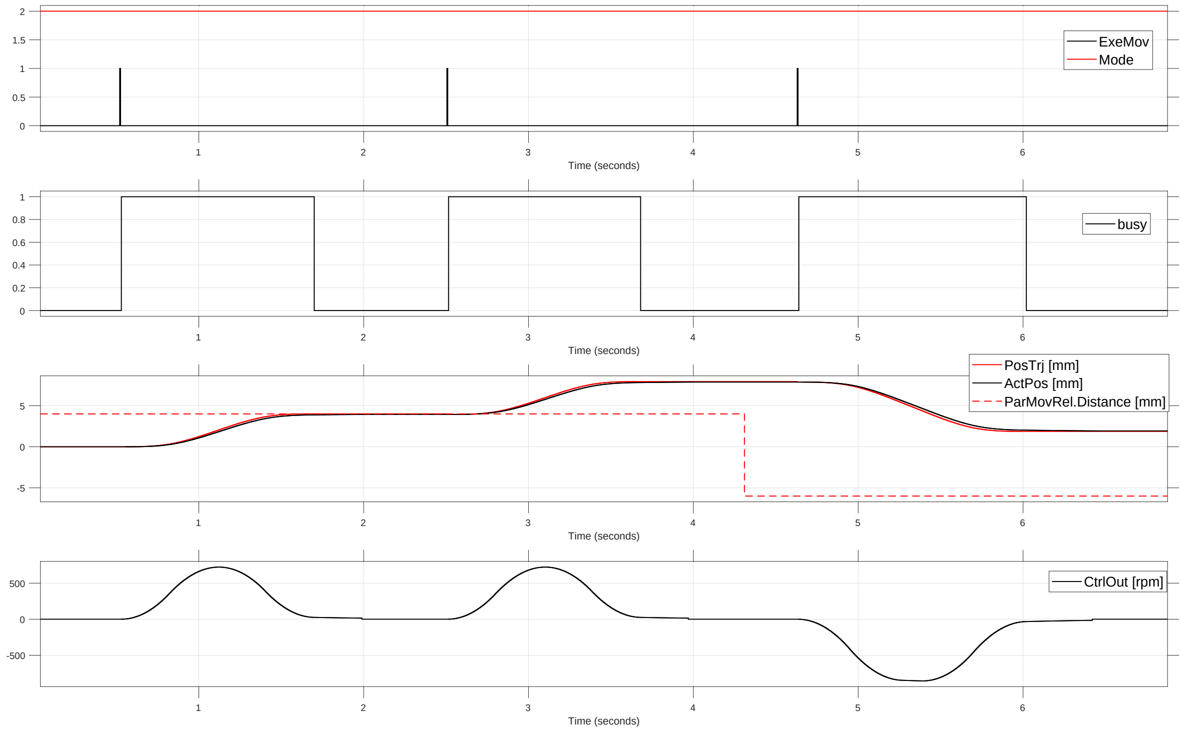


Figure 6: Movements in relative mode.

ParMovRel → MaxSpdSetPnt - Maximum speed setpoint for relative movement, <REAL>

See the description of *MaxSpdSetPnt* of the structure **ParMovAbs** .

ParMovRel → MaxAccSetPnt - Maximum acceleration setpoint for relative movement, <REAL>

See the description of *MaxAccSetPnt* of the structure **ParMovAbs** .

ParMovRel → JerkSetPnt - Jerk setpoint for relative movement, <REAL>

See the description of *JerkSetPnt* of the structure **ParMovAbs** .

ParJog		
	TgtSpd	<REAL>
	MaxAccSetPnt	<REAL>
	JerkSetPnt	<REAL>
	EnbPosCtrl	<REAL>

ParJog → TgtSpd - Target speed setpoint, <REAL>

When the *Mode* input is set to 3, the function operates in Jog Mode. In this mode, the linear axis moves forward with the configured speed, acceleration, and jerk as long as the *JogFw* input is True . If the *JogBw* input is activated, the linear axis moves backward. When *JogFw* and *JogBw* is set to False , the linear axis decelerates to a stop using the configured acceleration and jerk. This input defines the speed at which the linear axis moves while the *JogFw* or *JogBw* input is True , ensuring precise control over the jogging speed during manual operation. See figure 7.

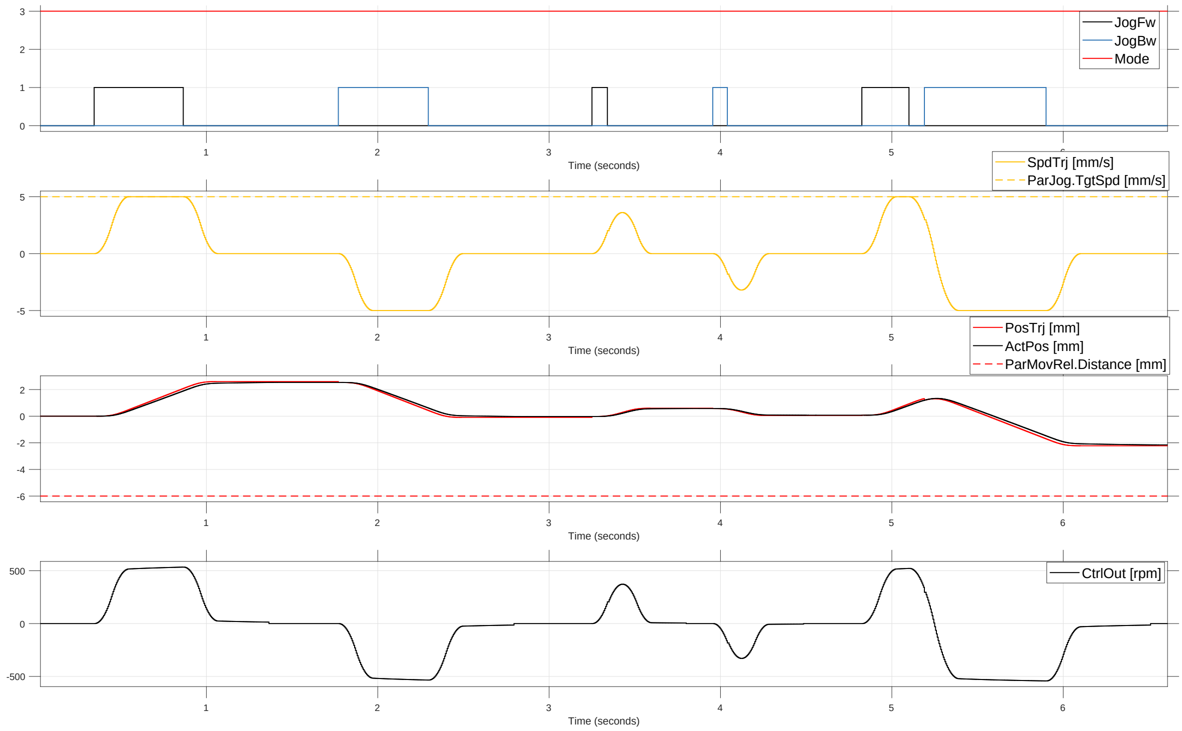


Figure 7: Movements in jog mode.

ParJog → MaxAccSetPnt - Maximum acceleration setpoint for jog mode, <REAL>

This input determines the maximum allowable acceleration of the linear axis during the movement in Jog Mode. It is measured in the same unit as Actual Position per second squared.

ParJog → JerkSetPnt - Jerk setpoint for jog mode, <REAL>

JerkSetPnt controls the abruptness or smoothness of motion in Jog Mode by regulating the rate of change of acceleration. It is measured in the same unit as actual position per second cubed. Adjusting *JerkSetPnt* allows users to customize the motion profile according to their desired level of abruptness or smoothness. Setting a jerk value as a setpoint is important for controlling the smoothness of acceleration and deceleration transitions. By managing the rate at which acceleration changes, users can prevent sudden, harsh movements that could cause mechanical wear or instability in the system. This ensures a smoother and more controlled operation, which is especially beneficial in applications where precision and mechanical longevity are critical.

ParJog → EnbPosCtrl - Enable position controller, <REAL>

This input allows the user to enable or disable closed-loop position control in Jog Mode. When this input is set to true, the linear axis operates in closed-loop control, meaning that the position feedback is actively used to maintain accurate position control during jogging operations. If this input is set to false, the linear axis operates in feedforward mode only, with no position feedback control, relying solely on the predefined speed and acceleration settings. This provides flexibility depending on whether precise position control or simple open-loop control is required during Jog Mode.

ParSpdCtrl		
	TgtSpd	<REAL>
	MaxAccSetPnt	<REAL>
	JerkSetPnt	<REAL>

ParSpdCtrl → TgtSpd - Target of speed control mode, <REAL>

The *TgtSpd* input is used to specify the target speed when the function block is configured in Motor Speed Mode (*Mode* $= 4$). In this mode, the user can send a speed setpoint to the motor, which is executed with the configured acceleration and jerk parameters. The function generates and sends a motor speed setpoint independently of the linear axis's position, allowing direct control of the motor's rotational speed.

Whether you're managing conveyors, pumps, fans, or other motion-driven applications, MotionProX ensures smooth, safe, and accurate motor speed transitions, protecting both your motor and mechanical components from excessive stress.

The unit for this input must be in a rotational speed format, such as RPM (revolutions per minute), rad/s (radians per second), or as a percentage of the motor's nominal speed. This input provides precise control over the motor's speed, making it ideal for applications where maintaining a specific motor speed is critical, regardless of the linear axis position.

In Motor Speed Mode, the user can change the target speed at any time. Any change to the *TgtSpd* input is immediately recognized by the function and is implemented using the configured acceleration and jerk parameters. This allows for real-time adjustments to the motor speed, ensuring responsive and precise control in dynamic applications. See figure 8.

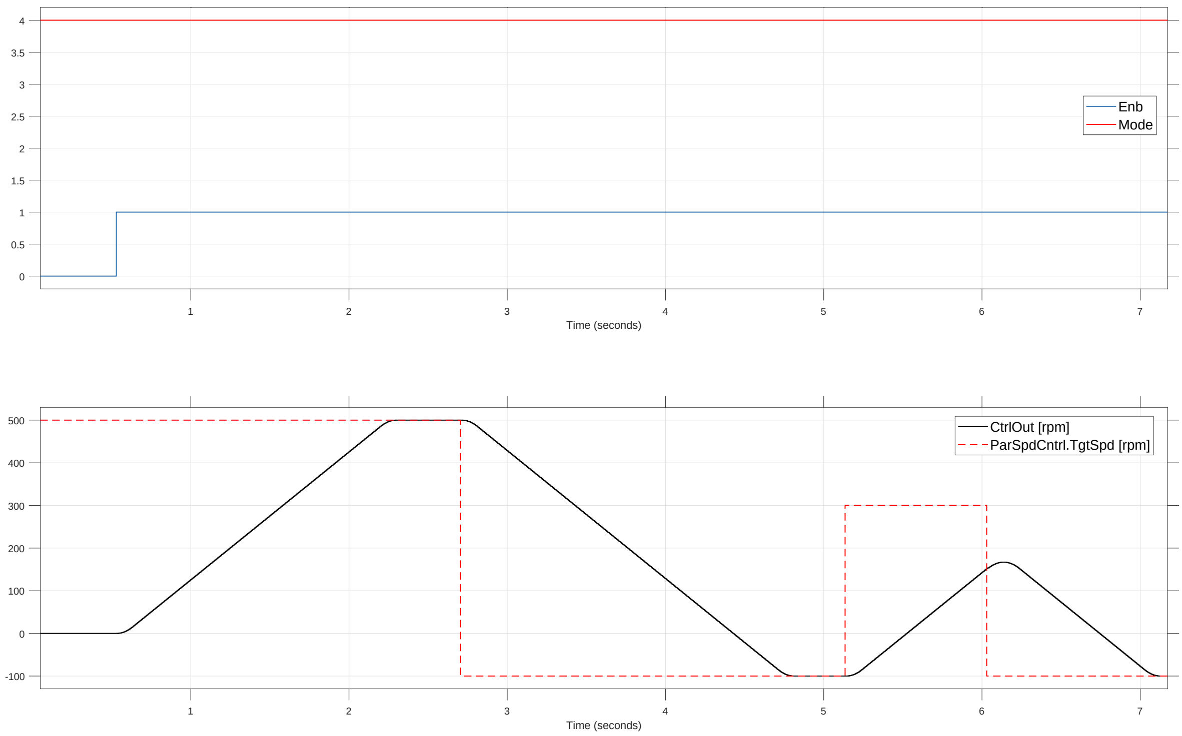


Figure 8: Movements speed control mode.

ParSpdCtrl → MaxAccSetPnt - Maximum acceleration for speed control mode, <REAL>

The *MaxAccSetPnt* input of the **ParSpdCtrl** struct defines the maximum motor acceleration when the function block is operating in Motor Speed Mode (*Mode* $= 4$). The unit for this input can be rpm/s, rad/s^2, or as a percentage of the motor's nominal speed per second. This input controls the rate at which the motor can increase its speed, ensuring that acceleration remains within safe and desired limits.

Please note that changes to this input are only possible when the motor is running at a constant speed. During the transition from one target speed (*TgtSpd*) to another, any changes to the *MaxAccSetPnt* input will not be processed by the function.

ParSpdCtrl → JerkSetPnt - Maximum jerk for speed control mode, <REAL>

This input defines the jerk of the motor during speed changes in Motor Speed Mode (*Mode* $= 4$). Jerk is the rate of change of acceleration or deceleration and is measured in units such as rpm/s^2, rad/s^3. It represents how quickly the motor's acceleration increases or decreases over time.

Setting a jerk value as a setpoint is important for controlling the smoothness of acceleration and deceleration transitions. By managing the rate at which acceleration changes, users can prevent sudden, harsh movements that could cause mechanical wear or instability in the system. This ensures a smoother and more controlled operation, which is especially beneficial in applications where precision and mechanical longevity are critical.

ParStop		
	DecSetPnt	<REAL>

ParStop → DecSetPnt - Deceleration of the motor stop, <REAL>

The function block includes a stop feature that allows the motor to be halted immediately, regardless of the current mode. As soon as a rising edge is detected on the *Stop* input, the function triggers a stop ramp to bring the motor to zero speed. The deceleration during this stop is defined by the *DecSetPnt* input, which can be configured in units such as [rpm/s], [rad/s^2], or as a percentage of the nominal speed per second. After the stop input is activated, normal operation of the function can only resume once the acknowledge, *Ack* , input is activated. This ensures that the system is properly reset and ready for continued use after an emergency or controlled stop. See figure 9.

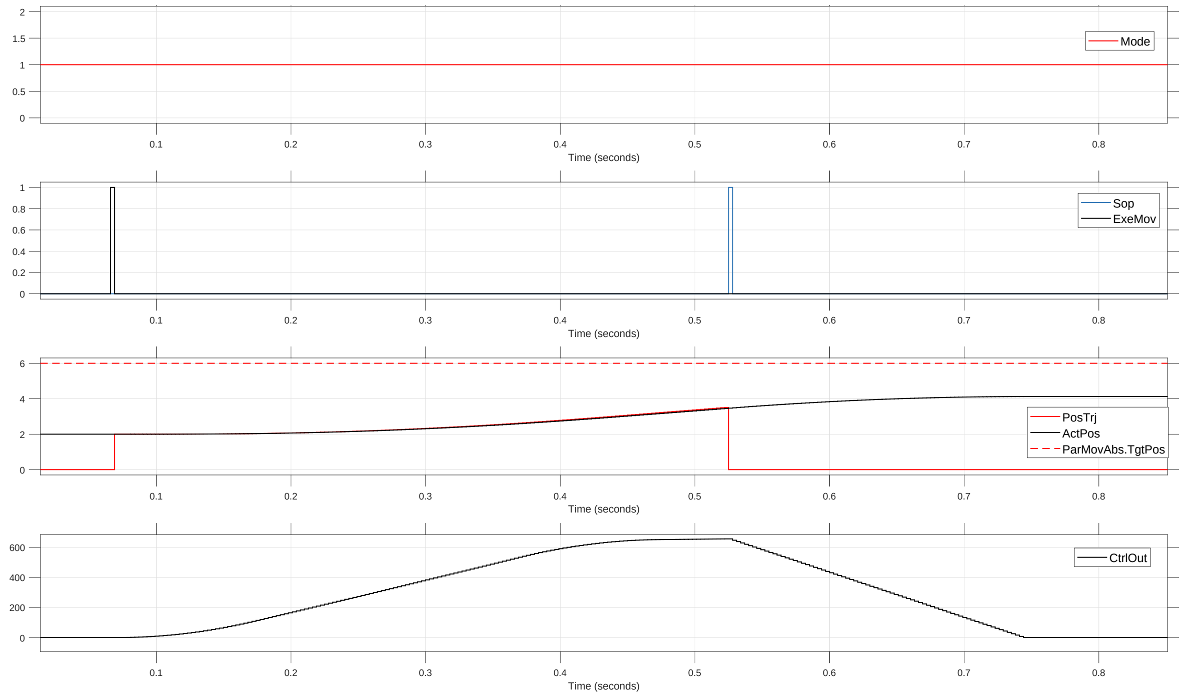


Figure 9: Maximum deceleration of the stop trajectory.

CtrlPar		
	Kp	<REAL>
	Ki	<REAL>
	GainFwdSpdCtrl	<REAL>
	GainBwdSpdCtrl	<REAL>
	ModeIntCntrl	<REAL>
	Window	<REAL>
	TuneTime	<REAL>

CtrlPar → Kp - Controller P-Factor, <REAL>

It represents the amplification factor of the P-controller within the position controller. A higher *Kp* value can enhance control accuracy. However, caution is advised: setting the *Kp* value excessively high may render the closed-loop system unstable. For optimal results and to maintain system stability, it is recommended to initiate with a modest *Kp* value and incrementally adjust upwards to achieve the desired precision. See example 2 for optimal adjustment.

CtrlPar → Ki - Controller I-Factor, <REAL>

Ki stands for the integration factor of the I-controller within the position controller. A *Ki* value set to zero effectively deactivates the I-controller. While a higher *Ki* value results in swifter error integration, enhancing overall accuracy, it can also introduce increased oscillations in the closed-loop system. For best performance, it is advisable to commence with a low *Ki* value and gradually increase it until the desired accuracy level is reached, balancing precision with system stability. See example 2 for optimal adjustment.

CtrlPar → GainFwdSpdCtrl - Positive speed feedforward, <REAL>

GainFwdSpdCtrl is a system parameter that adds a constant value for positive velocities to the control signal, improving trajectory tracking. It reduces the error signal and facilitates smoother following of the desired trajectory. The optimal positive feedforward value depends on the hardware and system dynamics, which can be determined through measurement and analysis, as illustrated in example 2.

CtrlPar → GainBwdSpdCtrl - Negative speed feedforward, <REAL>

GainBwdSpdCtrl is a system parameter that adds a constant value for negative velocities to the control signal, improving trajectory tracking. It reduces the error signal and facilitates smoother following of the desired trajectory. The optimal positive feedforward value depends on the hardware and system dynamics, which can be determined through measurement and analysis, as illustrated in example 2.

CtrlPar → ModeIntCntrl - Mode selection for internal integral controller, <REAL>

This input determines the behavior of the I-Controller. When set to False , the I-Controller remains continuously active. If set to True , the I-Controller operates only in the steady state, deactivating during the linear axis movement. We advise setting this input to True , as allowing the I-Controller to function solely in the steady state often provides superior stability and accuracy for linear axes. This approach typically minimizes, if not eliminates, significant overshoots and, by enabling an increase in the *Ki* factor, greatly enhances accuracy.

CtrlPar → Window - Tolerance window of the target position, <REAL>

The *Window* input is used to define a positional tolerance band around the target position *TgtPos* of the **ParMovAbs** structure. When the actual position *ActPos* of the linear axis is within half of the *Window* value from the target position and remains within this range for at least the specified *TuneTime* (which is configurable in seconds), the function block will send a speed setpoint of exactly 0.0 to the motor.

This functionality is crucial for achieving precise positioning. It ensures that when the axis gets close enough to the target position, the motor is gently brought to a complete stop, preventing overshoot and ensuring the axis remains within the desired positional accuracy. In figure 10 you can differentiate between three states:

1. Window and Positioning: As the actual position (black line) approaches the target position (red dashed line), the difference between them becomes smaller than half of the *Window* value (illustrated by the blue dashed lines).
2. TuneTime Activation: Once the actual position stays within this *Window* range for a duration of at least *TuneTime* (e.g. 0.5 seconds, as indicated on the graph), the function block signals the motor to reduce speed to exactly 0.0 (as shown in the *CtrlOut* graph).
3. Final Positioning: This results in the motor coming to a smooth and controlled stop.

This feature is essential for applications requiring high positional accuracy, as it prevents oscillations around the target position and ensures a stable final position.

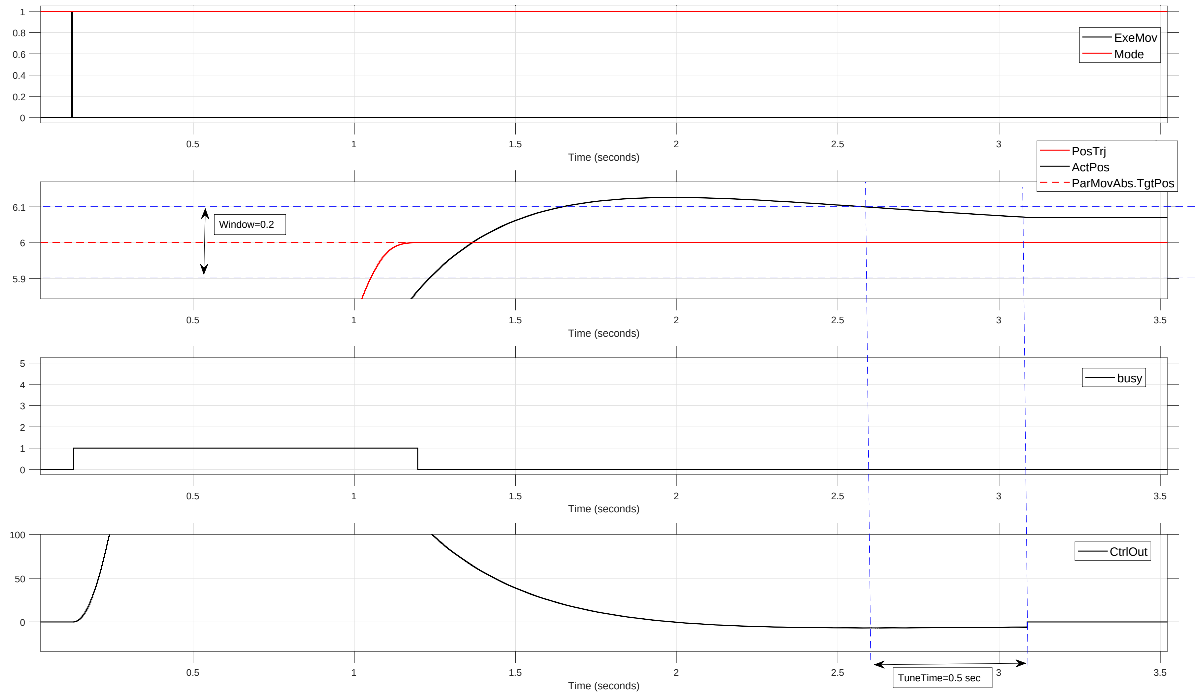


Figure 10: Behaviour of the tolerance window and tune time.

CtrlPar → TuneTime - Amount of time in tolerance window, <REAL>

The *TuneTime* input specifies the minimum amount of time that the actual position *ActPos* must remain within the defined *Window* around the target position *TgtPos* before the function block sends a zero-speed setpoint to the motor. This time parameter is configurable in seconds.

The purpose of the *TuneTime* input is to ensure that the axis has settled within the acceptable positional tolerance before bringing the motor to a complete stop. By requiring the position to stay within the *Window* for a specified duration, *TuneTime* helps to prevent premature stopping due to transient conditions or small oscillations. This results in more stable and precise final positioning, reducing the likelihood of overshoot or drift after the motor has stopped.

Enb - Turn controller on or off, <BOOL>

This input controls the activation or deactivation of the function. When the input signal is turned off False ,the function no longer responds to changes in the setpoints and does not generate any trajectory. The monitoring signal *PosTrj* reflects the current position, while the other monitoring signals *SpdTrj* and *AccTrj* are set to zero. when the function is deactivated, the *CtrlOut* is set to zero, indicating that there is no movement caused by the controller. For an overview, see table 1.

	CtrlOut	*PosTrj*	*SpdTrj*	*AccTrj*
Enb =True	calculated based on the position control algorithm	calculated by position trajectory generator	calculated by speed trajectory generator	calculated by acceleration trajectory generator
Enb =False	0.0	reflects current position *ActPos*	0.0	0.0

Table 1: Behavior of the output signals depending on the *Enb* input

Mode - Operation Mode Selection, <INT>

The function offers four different modes, as seen in table 2, which can be selected using the *Mode* input:

- *Mode* $= 1$: In this mode, the function block operates as an absolute movement position controller. The active set of parameters is **ParMovAbs** The user can define a target position, and upon detecting a rising edge on the *ExeMov* input, the linear axis moves to this target position.
- *Mode* $= 2$: In this mode, the function block acts as a relative movement position controller. The active parameter set is **ParMovRel** . The user can define a distance to move relative to the current position. When a rising edge is detected on the *ExeMov* input, the linear axis moves by the specified distance.
- *Mode* $= 3$: This mode enables manual jog movements. The active parameter set is **ParJog** . In this mode, the user can execute manual movements using two boolean inputs, *JogFw* for forward movement and *JogBw* for backward movement. The linear axis moves forward or backward according to the **ParJog** parameters as long as one of these inputs is active.
- *Mode* $= 4$: In this mode, the function block is configured to send a rotational speed setpoint to the motor. The active parameter set is **ParSpdCtrl** . The user can send a specific rotational speed setpoint with configurable acceleration and jerk to the motor. In this mode, the motor rotational speed is controlled independently of the linear axis position.

	1	2	3	4
Mode	Absolute Move Pos. Control	Relative Move Pos. Control	PushButton Jog Move	Motor Speed Control

Table 2: The different modes of the MotionProX function block

ActPos - Actual position, <REAL>

The input represents the current measured position of the linear axis. It is used in the error signal calculation of the position controller. It should be provided in the same unit as *TgtPos* or *Distance* dependend of the selected mode.

ExeMov - Execute Movement, <BOOL>

The *ExeMov* input is used to initiate movement in either *Mode* 1 (absolute position control) or *Mode* 2 (relative position control). When a rising edge is detected on this input, the function block starts the movement based on the current mode settings. In *Mode* 1, the linear axis begins moving toward the specified target position, while in *Mode* 2, it moves by the defined relative distance. This input effectively serves as the start command for executing the programmed motion.

JogFw - Input for forward movement in jog mode, <BOOL>

In *Mode* 4, the *JogFw* input controls the forward movement of the linear axis. As long as this input is active True , the axis moves forward according to the parameters defined in **ParJog** When the *JogFw* input is deactivated (switches from True to False), the function stops the linear axis using the same parameters, ensuring a smooth and controlled stop. This input is essential for manual jogging operations, allowing the user to control forward movement with precision. See figure 7.

JogBw - Input for backward movement in jog mode, <BOOL>

The *JogBw* input controls the backward movement of the linear axis in *Mode* 4. When this input is active True , the axis moves backward according to the parameters defined in **ParJog** . If the *JogBw* input is deactivated (switches from True to False), the function stops the linear axis using the same parameters, ensuring a smooth and controlled stop. This input is crucial for manual jogging operations, providing precise control over backward movement. See figure 7.

Stop - Safety stop in all modes, <BOOL>

The *Stop* input is a critical safety and control feature that overrides all other modes of operation. Regardless of the current mode the function block is in, when a rising edge is detected on the *Stop* input, the function immediately sends a zero-speed setpoint to the motor. This command is executed with a configurable deceleration rate, ensuring the motor comes to a controlled and safe stop. This input is essential for emergency stops or situations where an immediate halt of the motor is required, providing a reliable way to quickly bring the system to a stop. See figure 7.

Ack - Safety stop in all modes, <BOOL>

The *Ack* input is used to reset the function block after a stop command has been executed via the *Stop* input. Once the motor has been stopped, the function block is locked, preventing any further operations until a rising edge is detected on the *Ack* input. This input serves as a safety feature, ensuring that the system cannot resume normal operation until the stop event has been acknowledged by the user. It allows the user to verify and confirm that the system is ready to continue, ensuring controlled and safe operation after a stop condition.

Outputs

CtrlOut - Control Signal, <REAL>

The *CtrlOut* output is the main output of the function block, responsible for providing the motor speed setpoint. This output ensures that the linear axis reaches the desired position, or the motor achieves the target motor speed, or safely stops the motor, depending on the selected mode. The unit of this output could be defined in rpm, rad/s or as percentage of the motor's nominal speed.
The *CtrlOut* signal must be connected to a motor speed controller, such as a frequency drive, which translates the setpoint into the actual motor speed. *CtrlOut* is essential for translating the function block's control commands into motor speed, making it a crucial link between the control system and the motor drive.

PosTrj - Position trajectory, <REAL>

The *PosTrj* output displays the position trajectory of the linear axis as it moves toward the target position in *Mode* 1 or along the specified distance in *Mode* 2. When the function block is deactivated by setting *Enb* to False , is in *Mode* 4, or during the stop function, this output shows the current position of the linear axis. The unit of *PosTrj* is the same as that used for *TgtPos* or *Distance* dependend on the current *Mode* .
This output is intended solely for display purposes and should not be used to control other functions, as its reliability for such purposes is not guaranteed by Halow-Tech. It provides a visual representation of the expected or current position of the axis, aiding in monitoring the system's operation. See figure 3, figure 6 or 7.

SpdTrj - Speed trajectory, <REAL>

The *SpdTrj* output displays the speed trajectory of the linear axis as it moves toward the target position in *Mode* 1, over the specified distance in *Mode* 2, or during jogging in *Mode* 3. In these modes, it represents the linear speed and shares the same unit as *MaxSpdSetPnt* or *TgtSpd* depending on the current *Mode* . In *Mode* 4, or when the function is deactivated, this output will be zero.
The *SpdTrj* output is intended solely for display purposes and should not be used to control other functions, as its reliability for such use is not guaranteed by Halow-Tech. It provides a visual representation of the expected speed of linear axis, aiding in the monitoring and analysis of system performance. See figure 4, figure 5 or 7.

AccTrj - Acceleration trajectory, <REAL>

The *AccTrj* output represents the acceleration setpoint of the linear axis as it moves toward the target position in *Mode* 1, over the specified distance in *Mode* 2, or during jogging in *Mode* 3. In these modes, it reflects the linear acceleration and uses the same unit as *MaxAccSetPnt* . The *AccTrj* output is intended solely for display purposes and should not be used to control other functions, as its reliability for such use is not guaranteed by Halow-Tech. It provides a visual representation of the expected speed of linear axis, aiding in the monitoring and analysis of system performance. See figure 4 and figure 5.

Busy - Busy signal, <BOOL>

The *Busy* output indicates whether the internal trajectory generator is currently moving toward a target position. While the trajectory is in progress and the axis is moving to the new target, this output will be True . Once the target position is reached, the *Busy* output switches to False .

While *Busy* is True , the function block does not respond to new setpoints or impulses at the *ExeMov* input, preventing any interruptions or changes during the ongoing motion.

However, in *Mode* 3 (Jog Mode) and 4 (Speed Control Mode), the *Busy* output is always False . This means the user can adjust the *TgtSpd* in *Mode* 4 or use the push buttons in *Mode* 3 at any time, and the function block will immediately respond to these changes. This functionality ensures that the system can be dynamically controlled in these modes without waiting for the completion of a previous motion. See figure 11

Done - Done signal, <BOOL>

The *Done* output indicates whether the positioning task has been completed. In Modes 1, 2, or 3, as long as the function block is sending a non-zero speed setpoint to the motor, this output will be False , meaning the positioning task is still in progress. Once the positioning task is complete, the *Done* output switches to True , indicating that the function block is now sending a speed setpoint of exactly 0.0 to the motor.

This output is useful for signaling when the movement is fully completed, allowing the user or the control system to know when it is safe to proceed with the next operation or to issue new commands. It helps ensure that the motor has come to a full stop at the desired position before any further actions are taken. See figure 11

Error - Error conditions of the function block, <Boolean Array of 5 Elements>

The *Error* output is a boolean array with 5 elements, each indicating a specific error condition that prevents the function block from executing certain operations. Understanding these elements is crucial for diagnosing issues and ensuring the function block operates correctly.
This output is essential for monitoring the status of the function block and ensuring safe operation. By checking these elements, users can quickly identify which mode or parameter is causing an issue, allowing them to correct the problem before attempting to execute new movements or speed commands. This ensures that the system operates. For an overview, see table 3.

- Element 1, *Mode* 1 Error: If True , no new movement can occur in *Mode* 1. This indicates that one or more of the inputs *MaxSpdSetPnt* , *MaxAccSetPnt* or *JerkSetPnt* of the struct **ParMovAbs** are zero or negative.
- Element 2, *Mode* 2 Error: If True , no new movement can occur in *Mode* 2. This indicates that one or more of the inputs *MaxSpdSetPnt* , *MaxAccSetPnt* or *JerkSetPnt* of the struct **ParMovRel** are zero or negative.
- Element 3, *Mode* 3 Error: If True , no new movement can occur in *Mode* 3. This indicates that one or both of the inputs *MaxAccSetPnt* or *JerkSetPnt* of the struct **ParJog** are zero or negative.
- Element 4, *Mode* 4 Error: If True , no new rotational speed setpoint can be issued in *Mode* 4. This indicates that one or both of the inputs *MaxAccSetPnt* or *JerkSetPnt* of the struct **ParSpdCtrl** are zero or negative.
- Element 5, Stop condition Error: If True , no new movement or speed setpoint can be issued. This element remains True if the stop function has been activated and will only reset when the function's main output *CtrlOut* is zero and the *Ack* input has been activated.

	Element 1 =True	Element 2 =True	Element 3 =True	Element 4 =True	Element 5 =True
Error	Invalid parameters **ParMovAbs**	Invalid parameters **ParMovRel**	Invalid parameters **ParJog**	Invalid parameters **ParSpdCtrl**	Not acknowledged after stop

Table 3: The different error codes of the MotionProX function block

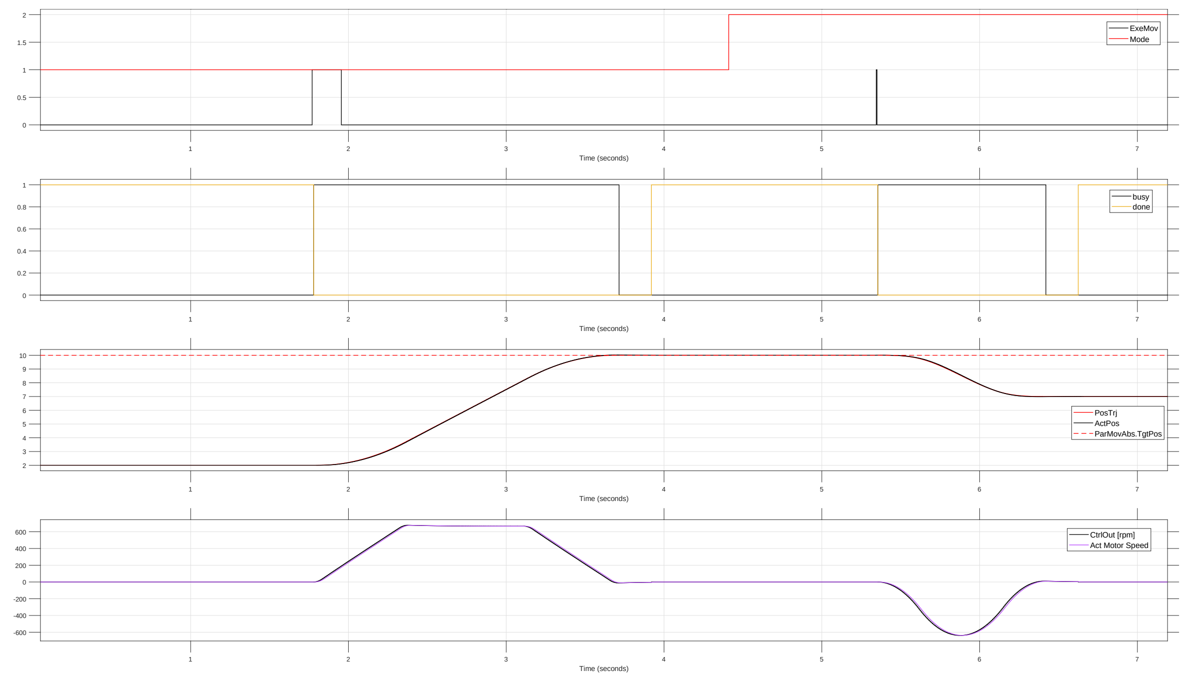


Figure 11: Behaviour of the *Busy* and *Done* output.

FAQ

Can I change the target position while the linear axis is moving in Mode 1 or 2?

The function block will not respond to any setpint changes during the movement and will continue to move toward the original target. The function block only reacts to the new setpoints when the *Busy* output is False .

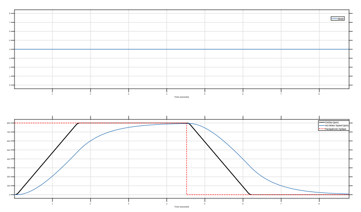
When can I change the mode?

The mode will be changed, when the *Busy* output is False (in *Mode* 1 or 2), when the motor speed is zero in *Mode* 4, or when the *Done* output is True in *Mode* 3.

What happens if I set the Enb input to False while the linear axis is moving or the motor speed is not zero?

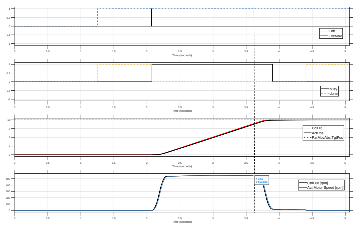
If you deactivate the function block by setting the *Enb* input to False while the linear axis is in motion or the motor is running at a non-zero speed, the main output of the block will abruptly drop to zero. This sudden change in motor speed can potentially damage the motor or other mechanical components. Therefore, it's important to ensure that the motor has come to a stop or the movement is complete before deactivating the block.

Examples



Example 1: Fine-tuning the frequency drive's control parameters.

Adjust the parameters so that the motor can quickly and accurately match the rotational speed setpoint.



Example 2: Configure the MotionProX function block parameters.

Step-by-Step guide to configure the block parameters.

Important for all examples: These values are provided solely as examples to illustrate the approach for setting control parameters. Under no circumstances should these values be directly applied to your machine, not even as initial starting points. The control parameters must be explicitly adjusted for each machine individually.

Fine-tuning the frequency drive's control parameters

In this examples, we'll walk you through the steps to configure the technology block for accurate positioning of an electric axis. This step is essential before proceeding to the next steps, where you'll tune the control parameters of the position controller.

The Technology Block calculates the motor's rotational speed setpoint, which is output through *CtrlOut* . This output needs to be connected to a Frequency Drive that controls the motor's speed. To ensure precise positioning, it's crucial that the motor can accurately follow the rotational speed setpoint provided by *CtrlOut* . To achieve this, start by operating the Technology Block in Mode 4. In this mode, you'll send rotational speed setpoints along with predefined acceleration and jerk values to the frequency drive.

The first step, described in this example, is to fine-tune the frequency drive's control parameters so that the motor can quickly and accurately match the rotational speed setpoint, with minimal overshoot. The motor must also be capable of following these speed setpoints at the maximum allowable acceleration and jerk rates, maintaining precision and stability.

Once you've ensured that the motor's speed control is highly responsive and stable, you can proceed to use the Technology Block in Modes 1, 2, and 3 for precise positioning tasks. This initial setup is key to achieving the accurate positioning required in these modes.

Important Safety Note for Operating in Mode 4: When operating MotionProX in Mode 4, it's crucial to pay close attention to the position of the linear axis. In Mode 4, rotational speed setpoints are sent to the motor regardless of the position of the linear axis. This means the motor will follow the speed commands without considering the current position, which can lead to unintended movements of the axis.
To avoid any potential issues, it's recommended to perform this step while the linear axis is not yet coupled to the motor, allowing the motor to spin freely without affecting the axis. This precaution helps prevent any accidental damage to the mechanical components or misalignment of the system.
If this step is not carefully managed, the motor could drive the axis unexpectedly, leading to possible mechanical damage or even safety hazards. Ensuring the motor is decoupled or free to rotate during this phase is essential for a smooth and safe setup process.

The following is an example of the problems that can arise when setting the frequency drive. At the end of the example, it is shown how precisely the entire system must react in order to use the controller.

1. Issue of Motor Speed Control in figure 12:

   (a) Large Lag Error: The actual motor speed (blue line) lags significantly behind the motor speed set trajectory (black line), indicating poor tracking performance. The motor takes too long to reach the desired speed, resulting in a slow response.

   (b) No Steady-State Accuracy: The actual motor speed never reaches or maintains the target speed (red dashed line).

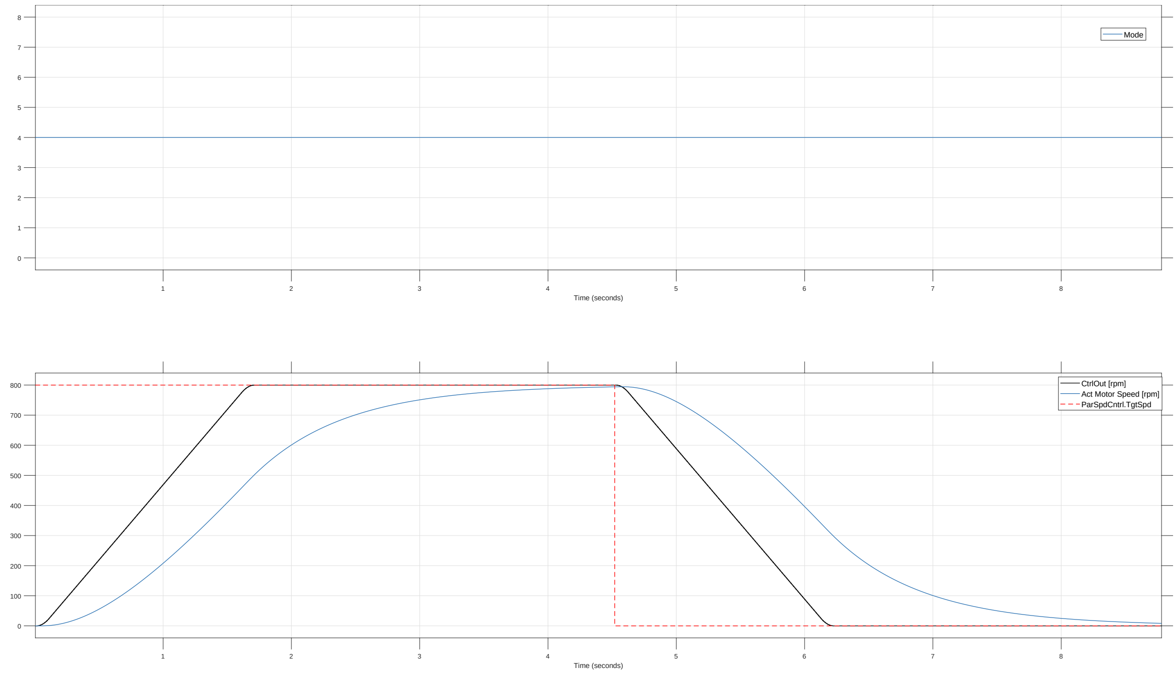


Figure 12: Frequency Drive with poorly set controller parameters. Large Lag Error and no Steady-State Accuracy.

2. Issue in figure 13:
   (a) Large Lag Error: The actual motor speed (blue line) significantly lags behind the set trajectory (black line) during both the acceleration and deceleration phases. This indicates a delayed response in following the desired speed profile.
   (b) Overshoot and Oscillation: After reaching the target speed, the actual motor speed overshoots the setpoint (red dashed line) and exhibits oscillations. This indicates poor damping and instability in the speed control, leading to oscillatory behavior instead of smooth convergence to the target speed.

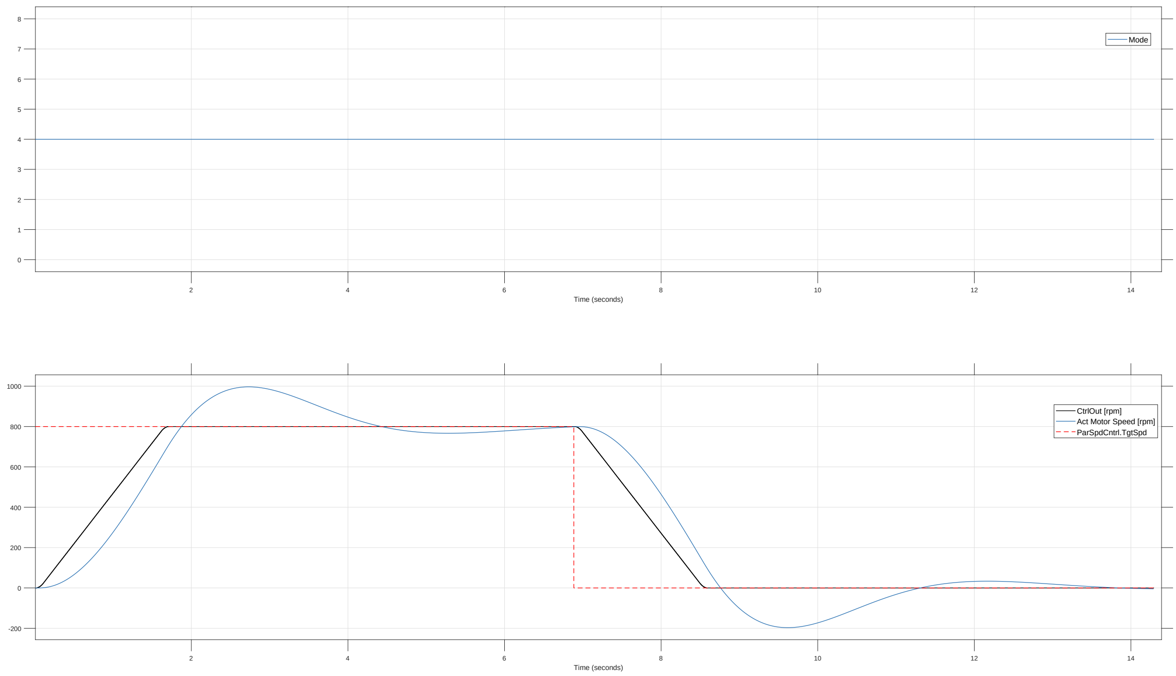


Figure 13: Frequency Drive with poorly set controller parameters. Large Lag Error, Overshoot and Oscillation.

3. The behavior in figure 14 is better than in both figures 12 and 13, but there is an issue:
   (a) Large Lag Error, where the actual motor speed (blue line) still lags behind the set trajectory (black line) during acceleration and deceleration phases, indicating that the response is still delayed in following

the desired speed profile.

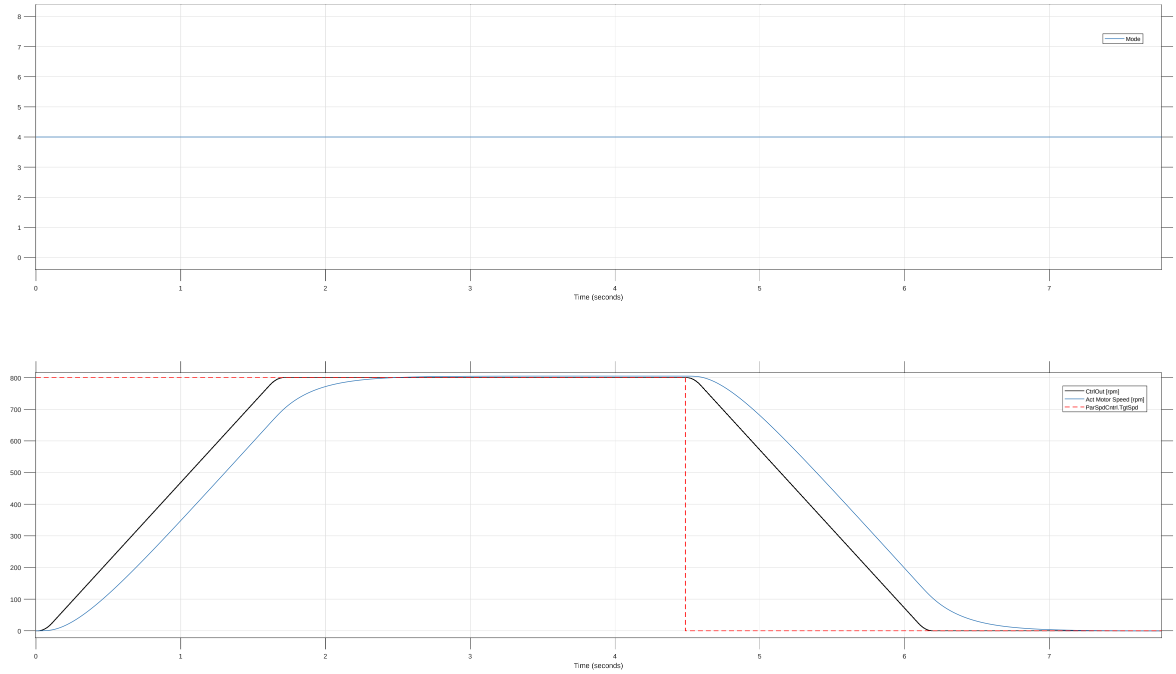


Figure 14: Frequency Drive with poorly set controller parameters. Large Lag Error.

4. The behavior in figure 15 is an improvement compared to the previous figures, but there is still a relative large Lag Error:
   (a) The actual motor speed (blue line) continues to lag behind the set trajectory (black line), indicating a delayed response in following the desired speed profile. While performance has improved, the motor still struggles to match the speed changes accurately.

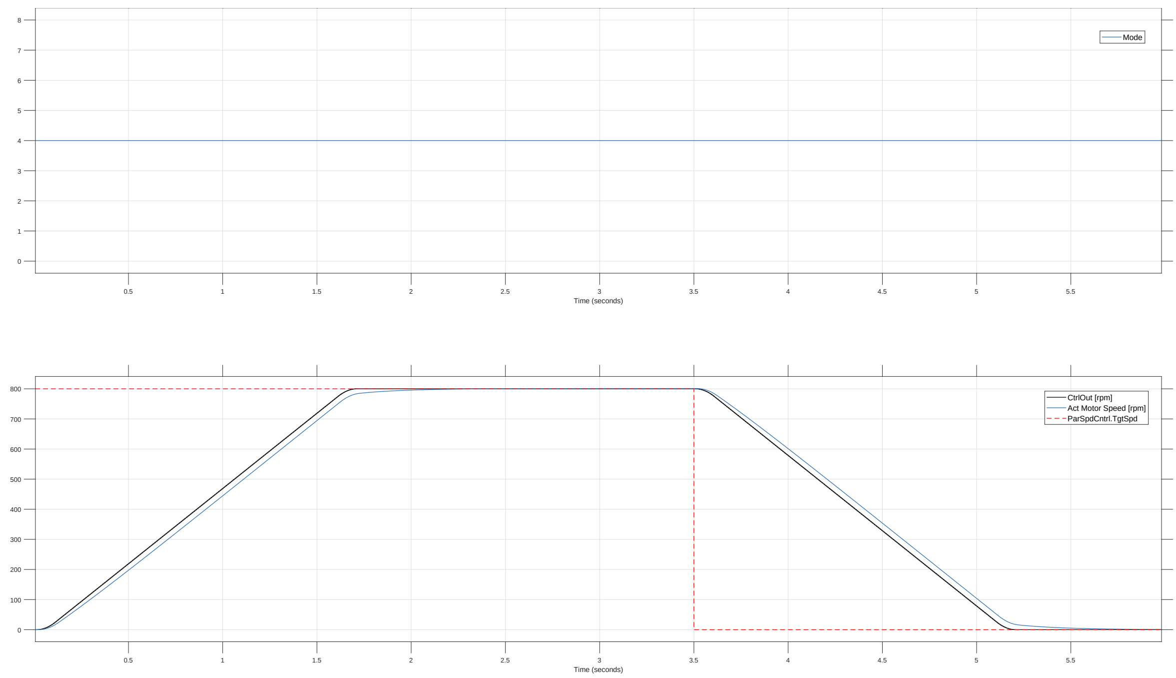


Figure 15: Frequency Drive with poorly set controller parameters. Relatively large Lag Error.

5. Optimal set of controller parameters in figure 16: The lag error between set trajectory (black line) and motor actual speed (the blue line) is negligibly small, there is no over shoot and no oscillation. steady- state accuracy is very high.

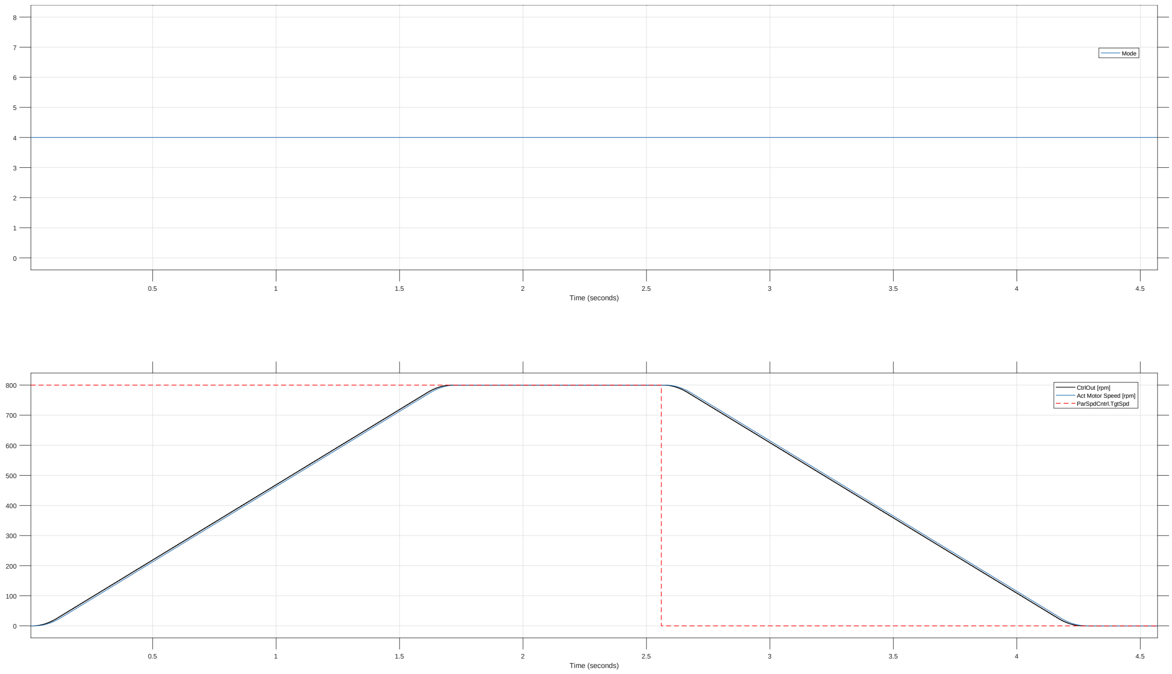


Figure 16: Frequency Drive with optimal set of controller parameters.

Configure the MotionProX function block parameters

This guide provides detailed instructions on how to set up and configure the MotionProX Technology Block for precise positioning tasks of an electric linear axis. To proceed with the following steps, the frequency drive must be configured as described in the **previous Example**. This setup is a prerequisite for successful tuning of the position control parameters.

1. General Informations
   - Cyclic Interrupt Task Setup: Ensure that the Technology Block is called within a cyclic interrupt task in your PLC. The feedback for the actual position must be read at least as frequently as the cycle time of this cyclic interrupt task.
   - Sampling Time: The task's cycle time should be relatively fast, depending on the required accuracy and motor dynamics. A sampling time of 1 to 5 milliseconds is a good starting point.
   - Communication with the Frequency Drive: The Technology Block only calculates the speed setpoint for the frequency drive. Control and status words for communication with the drive must be programmed separately by the user. The speed setpoint calculated at the *CtrlOut* output should be sent directly to the frequency drive without modification.
2. Define Units
   - Consistency in Units: Before configuring the block, decide on the units you will use for the system. These units must remain consistent throughout the configuration.

     For example, if you choose millimeters for the linear axis, then all related variables should be in millimeters:
     Actual Position $[\text{mm}]$, Target Position $[\text{mm}]$, Speed Setpoint $[\frac{\text{mm}}{\text{s}}]$, Acceleration Setpoint $[\frac{\text{mm}}{\text{s}^2}]$, and Jerk $[\frac{\text{mm}}{\text{s}^3}]$.

     For motor parameters, you might use rpm:
     Motor Target Speed $[\text{rpm}]$, Motor Acceleration $[\frac{\text{rpm}}{\text{s}}]$, Motor Jerk $[\frac{\text{rpm}}{\text{s}^2}]$.

   - Other Unit Options: You can choose other units, such as meters, centimeters, micrometers for the linear axis, or rad$/$s or percentage of nominal motor speed for the motor. The key is to select a unit system and maintain consistency across all parameters and modes.
3. Set General Parameters
   - Sample Time: Enter the PLC task's cycle time (in seconds) into the *SampleTime* input.
   - Max and Min Output: Define the maximum and minimum speed the Technology Block can output using *MaxOut* and *MinOut* , for example, 1500 rpm maximum and -1500 rpm minimum.
4. Configure Setpoints for Mode 1
   - Mode 1 Setup: Switch the block to Mode 1. Using the setpoint structure **ParMovAbs** , define a target position with corresponding speed, acceleration, and jerk setpoints. Ensure that all setpoints (speed, acceleration, jerk) are greater than zero.
5. Set K_p and Target Accuracy
   - K_p Setting: Initially, enter a small value for *Kp* of the **CtrlPar** structure.
   - Define Accuracy Window: Set the accuracy window using *Window* input in the same units chosen for the linear axis (e.g., 0.2mm).
   - Tune Time: Enter a value in seconds for *TuneTime* (e.g., 0.3s).
   - Other Control Parameters: You can leave other control parameters at zero for now; they will be adjusted later.
6. Activate the Block and Execute Movement
   - Block Activation: Activate the block by setting *Enb* to True . Trigger a movement by sending a pulse to the *ExeMov* ' input and observe the linear axis movement.

- The goal at this stage is to ensure that the actual position and the generated position trajectory (*PosTrj* output) are parallel. Do not focus on target accuracy at this point. See figure 17.

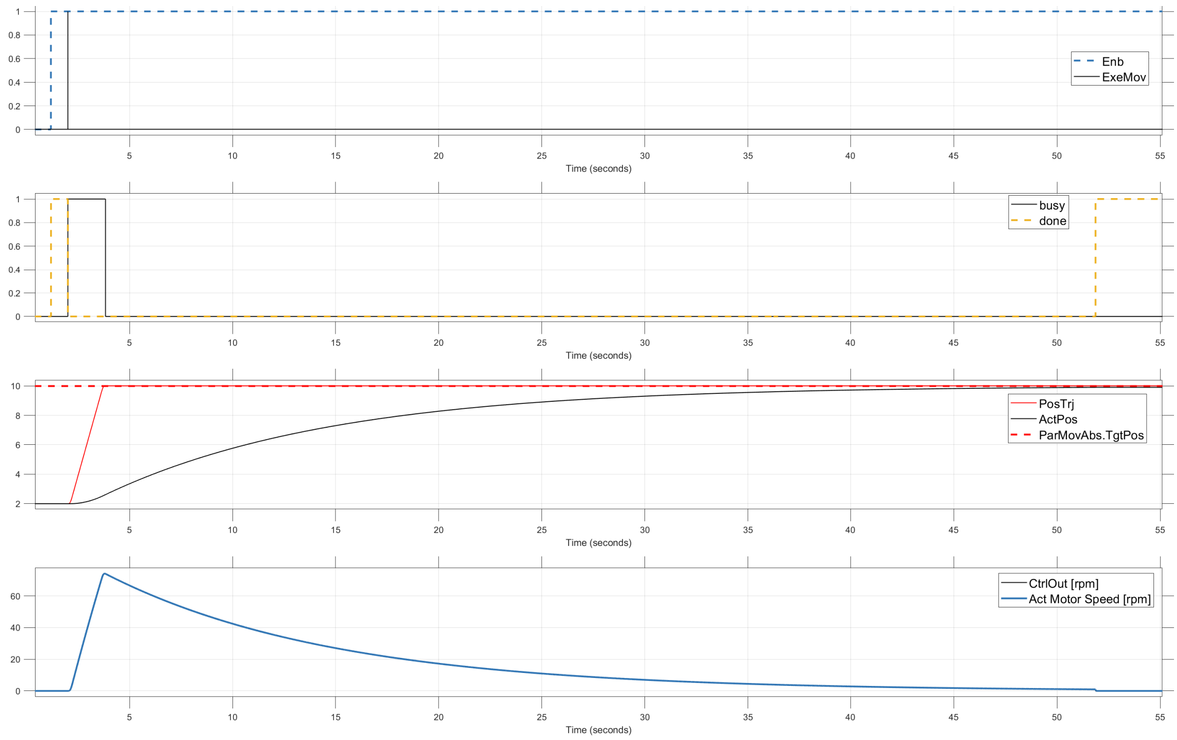


Figure 17: Example with Kp too small! ($K_p = 10$, Window $= 0.2$mm, *TuneTime* $= 0.3$s)

7. K_p Tuning
   - Incremental K_p Adjustments: Gradually increase the K_p value. After each increment, execute a new movement by defining an appropriate target position and sending a pulse to *ExeMov* .
   - Objective: Continue this process until the position trajectory (*PosTrj*) and the actual position are parallel, indicating proper tuning. see figures 18, 19 and 20.

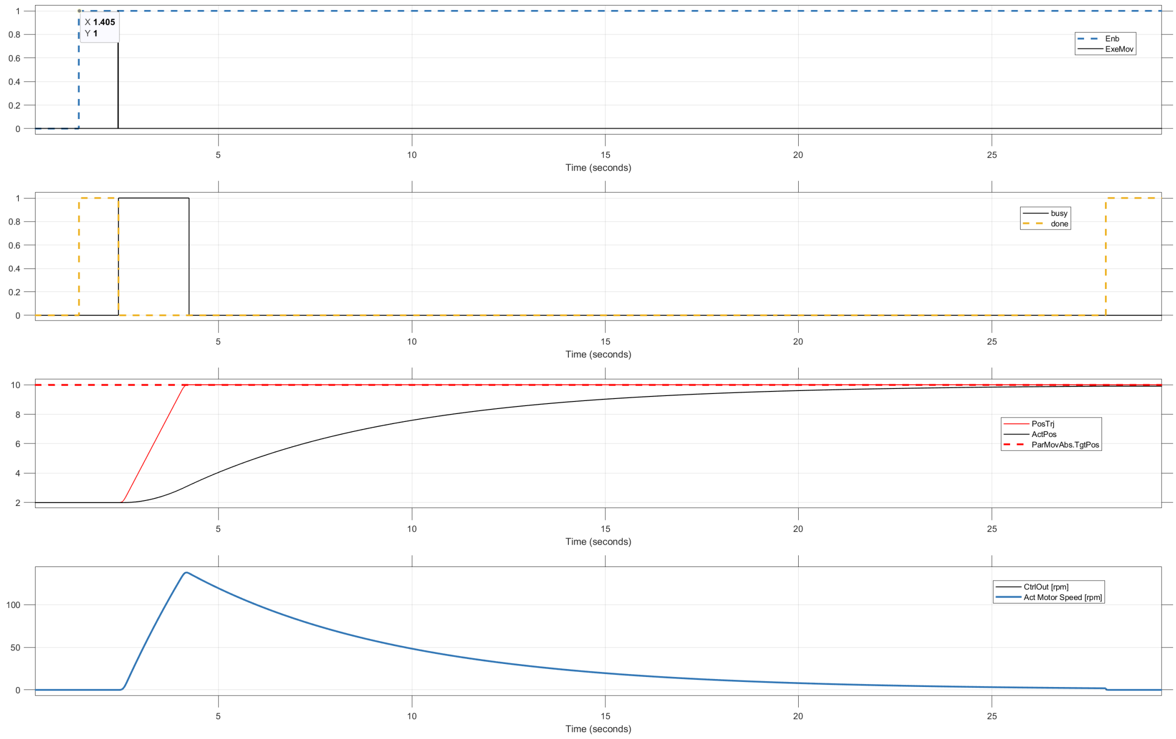


Figure 18: Kp increased to 20, still small! ($K_p = 20$, $K_i = 0.0$, *GainFwdSpdCtrl* $= 0.0$, *GainBwdSpdCtrl* $= 0.0$, Window $=$ 0.2mm, *TuneTime* $= 0.3$s)

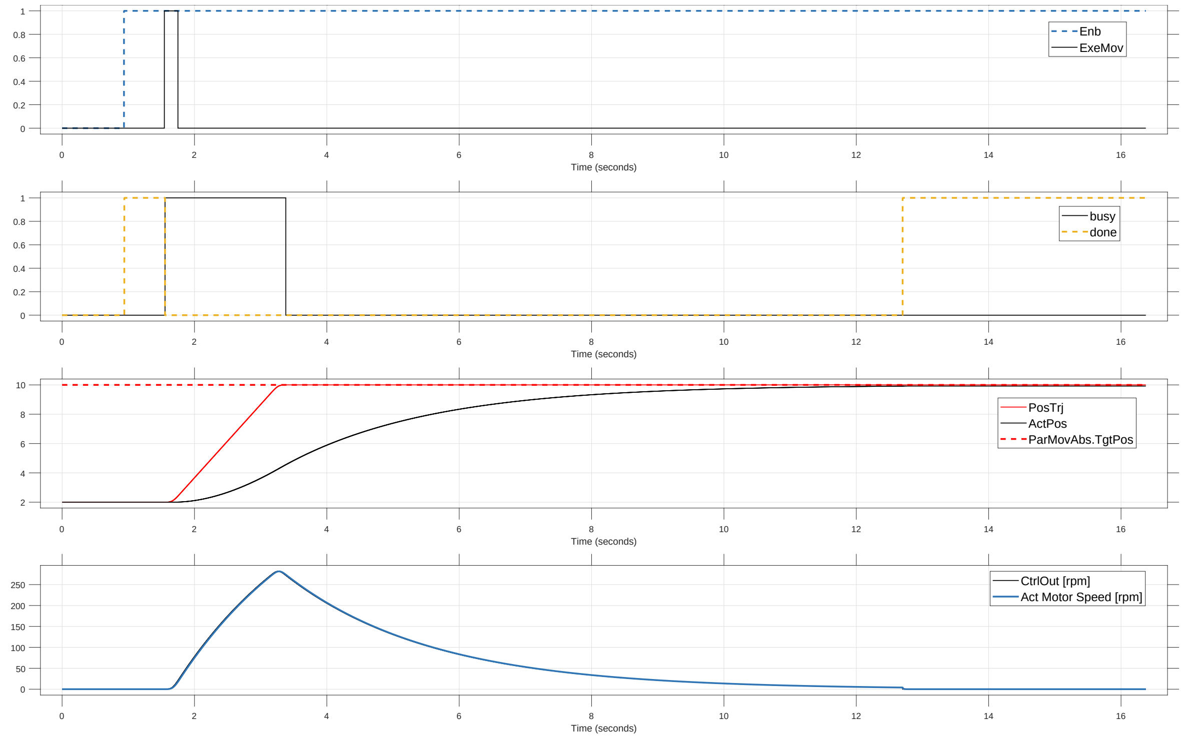


Figure 19: Kp increased to 50, still small! ($K_p = 50$, $K_i = 0.0$, $GainFwdSpdCtrl = 0.0$, $GainBwdSpdCtrl = 0.0$, Window $= 0.2$mm, $TuneTime = 0.3$s)

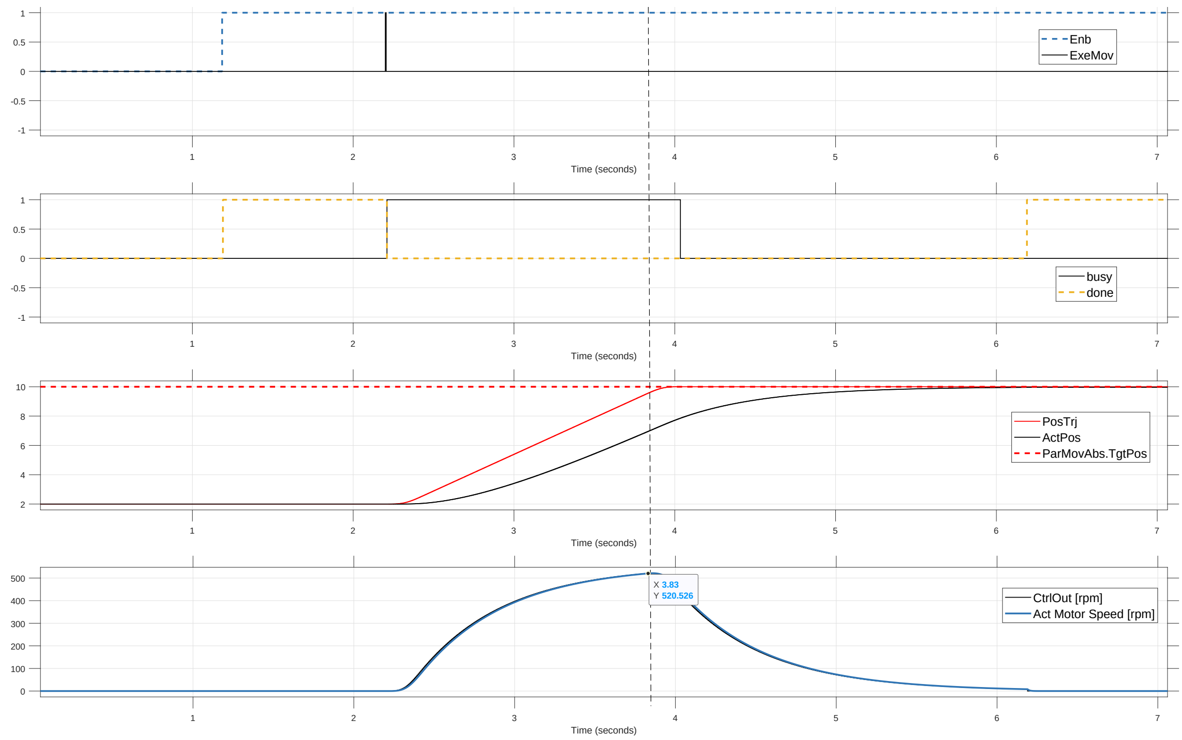


Figure 20: Kp increased to 200, ready for next step! ($K_p = 200$, $K_i = 0.0$, $GainFwdSpdCtrl = 0.0$, $GainBwdSpdCtrl = 0.0$, Window $= 0.2$mm, $TuneTime = 0.3$s)

8. Determine the Positive Feedforward Parameter
   - Execute a Positive Movement: Run a positive movement of the linear axis.
   - Measure Motor Speed: At a point where the actual position and the position trajectory (*PosTrj*) are parallel, read the motor speed.
   - Calculate Gain: Divide the motor speed by the set speed value (*MaxSpdSetPnt* of the structure **ParMovAbs**). The result is the value you should enter into *GainFwdSpdCtrl* of the structure **CtrlPar**. In

the example shown in figure 21. The *MaxSpdSetPnt* was set to $5\frac{\text{mm}}{\text{s}}$. The motor speed is 520.52rpm. The value for *GainFwdSpdCtrl* is then calculated by

$$\textit{GainFwdSpdCtrl} = \frac{520.52}{5} = 104.1$$

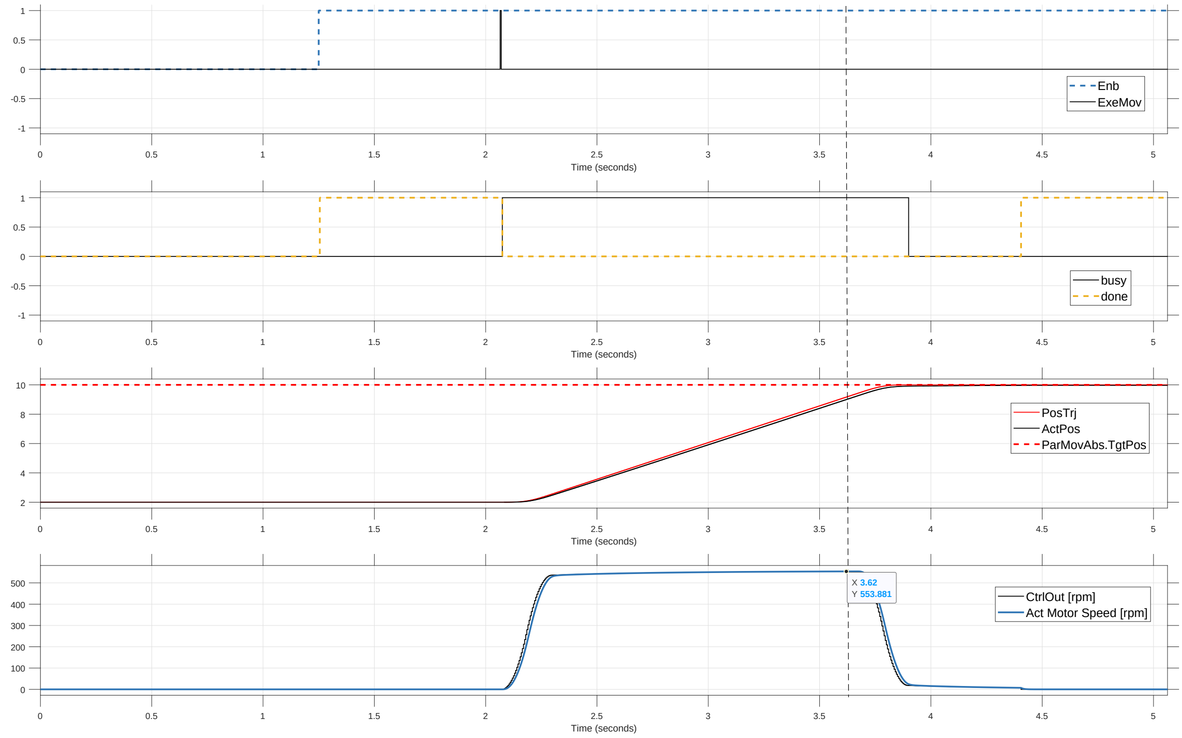


Figure 21: Adjusted Feedfordward parameters ($K_p = 200$, $K_i = 0.0$, *GainFwdSpdCtrl* $= 104.1$, *GainBwdSpdCtrl* $= 0.0$, Window $= 0.2$mm, *TuneTime* $= 0.3$s)

9. Determine the Negative Feedforward Parameter
   - Repeat Step 8: Perform the same process as in Step 8, but with a negative movement of the linear axis.
   - Calculate and Set Gain: Write the resulting value into *GainBwdSpdCtrl* of the strucutre **CtrlPar** .
10. Fine-Tune Feedforward Parameters (Optional)
    - Execute a Movement: Run a movement using the $K_p = 20$ value and the determined feedforward parameters.
    - Analyze the Motion: The movement should resemble the ideal trajectory shown in reference images (e.g., similar to figure 20).
    - Refine *GainFwdSpdCtrl* : Determine the positive feedforward gain again by observing the highest motor speed during the movement. Divide this by the set speed value (*MaxSpdSetPnt*) and update *GainFwdSpdCtrl* with this value. See figure 21.

$$\text{GainFwdSpdCtrl} = \frac{553.88}{5} = 110.7$$

    - Refine *GainBwdSpdCtrl* : Repeat the process with a negative movement to refine *GainBwdSpdCtrl* .

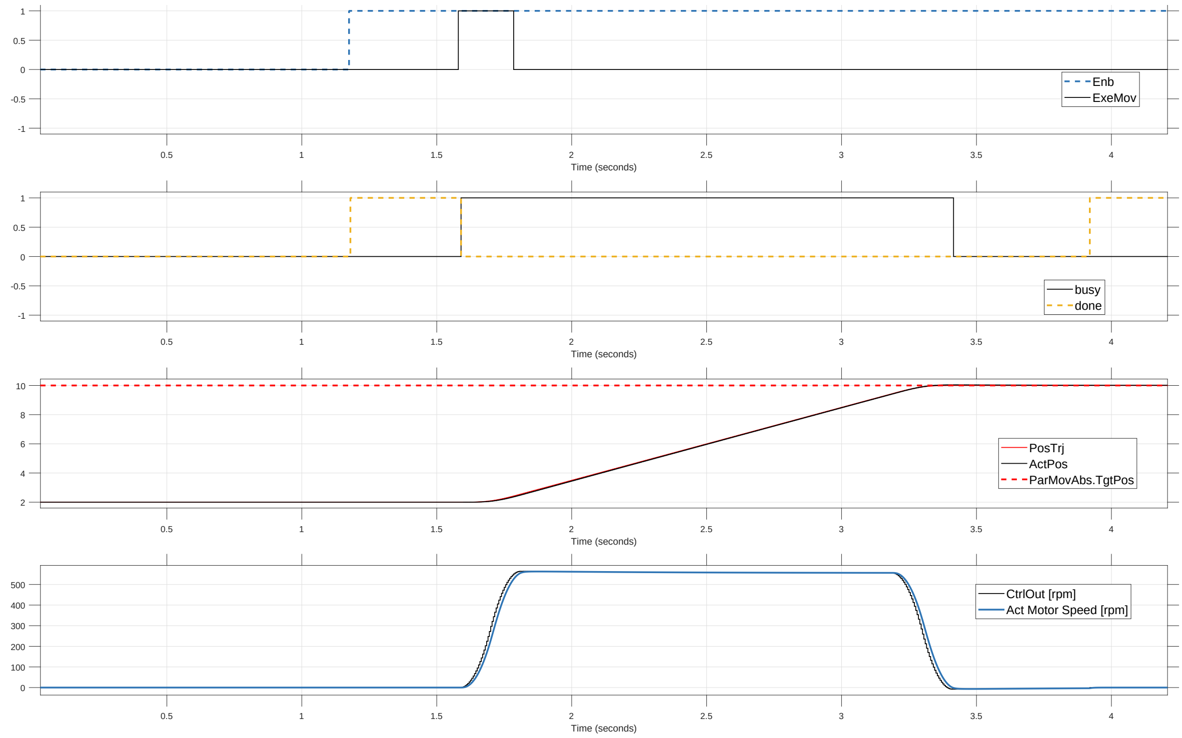


Figure 22: ($K_p = 200$, $K_i = 0.0$, *GainFwdSpdCtrl* $= 104.1$, *GainBwdSpdCtrl* $= 110.7$, Window $= 0.2$mm, *TuneTime* $= 0.3$s)

11. Adjust K_i Parameter
    - Monitor Signals: Observe the time between when the *Busy* signal is reset and when the *Done* signal is set.
    - Increase *Ki* : Gradually increase the *Ki* value and minimize the *TuneTime* to achieve the desired speed and responsiveness.
12. Finalize the Position Controller Setup
    - Optimal Configuration: Once the *Kp* , feedforward parameters, and *Ki* are tuned, the position controller is optimally set.
    - Run Movements in Modes 1, 2, and 3: You can now move the linear axis in Modes 1, 2, and 3 with any target positions, speeds, accelerations, and jerks that your mechanics and motor can handle, without needing to adjust the **CtrlPar** values again.
    - By following these steps, you ensure that your Technology Block is finely tuned for accurate and responsive control, enabling smooth and precise operation of the electric axis in various modes.