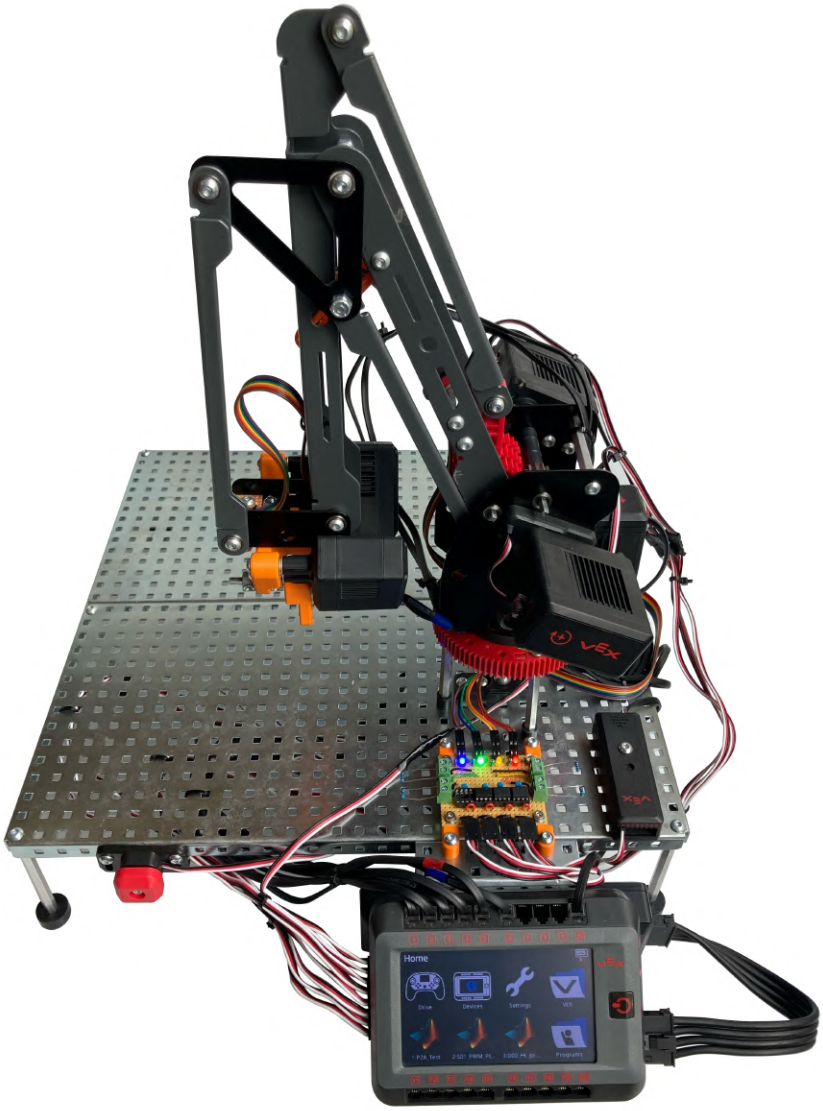
Simulink Library Documentation

rcl46

September 2022



Contents

1 Introduction

The experimental setup consists of the Vex Arm, Vex V5 Brain, a gripper and a circuit capable of reading 4 force sensors. This provides a platform for which to develop position control algorithms, perform pick and place proceedures and experiment with force control. The remainder of this document details the hardware, how to create Simulink models and finally the Simulink library to be used in this lab.

2 Vex V5 Brain Overview

The Vex V5 Brain is the interface between the computer and the robot arm. The models created in Simulink are uploaded to this via a micro USB cable and can be run by clicking them on the touch screen interface. The touch screen also allows the user to view the devices connected to each Smart Port and access a few basic settings.

Figure 1: Vex V5 Brain - the control centre for the robotic arm

Figure 1 shows the Smart Ports - these are used to connect the Smart Motors, the actuators used to control the arm. 20 of these ports are located on the edge as shown, with Port 21 located on the side (Figure 2). Whenever a Simulink block mask asks for a Smart Port number, this is what it is referring to.

For more simple purposes such as analog/digital inputs, the 3-Wire ports are used as shown in Figure 2 (left). Whenever a Simulink block mask asks for a 3-Wire port/analog input/digital input, this is what it is referring to.

The other side of the Vex V5 Brain (Figure 2 right) has, from left to right: the battery connector, the USB connector, an SD card slot and Smart Port 21. The USB cable is used to upload programs, and the SD card allows for data logging which is explained further in this document.

Figure 2: 3-Wire ports

Figure 3: Power, USB and SD card connectors

Figure 4: Vex Battery

The arm is powered via a Li-Ion 1100mAh battery which must be connected to the corresponding port on the Vex V5 Brain using the provided cable. There is a button on the battery which gives an indication of remaining charge, but if it is connected to the Vex V5 Brain the exact percentage can be read on screen. If possible, it is best not to charge the battery while it is in use and instead have 2 batteries in rotation.

3 Vex V5 Arm

The Vex Arm is constructed from 4 members and therefore has 4 degrees of freedom - the angle of each joint. To keep notation consistent, each joint has been assigned a letter as shown in Figure 5b.

(a) The Vex V5 Arm

(b) Arm joint labels

Figure 5

The rotation of joints A through D are each controlled by a separate motor, which in turn allows rotation of the 4 members. The pose of the arm can then be specified by only 4 angles.

In order to measure the current pose, potentiometers are coupled to each motor's axle* and connected to the Vex V5 Brain. The value read from each can then be scaled in order to calculate the current joint angles in radians.

Traditionally in robotic arms, angles are measured with respect to the previous member and so two different angle conventions are used as shown in Figure 6. All blocks in the Simulink library will expect angles in the 'Motor' definition but utilities to convert between the two are provided (*The potentiometer at joint A is coupled instead to the base rather than the motor axle).

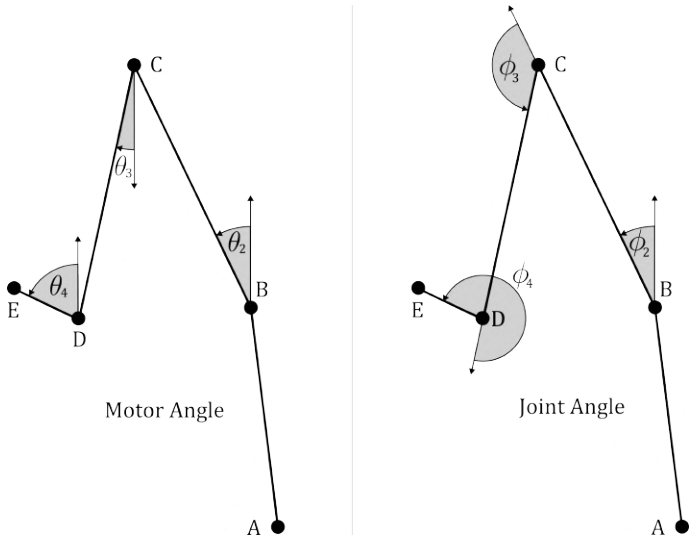


Figure 6: Angle definitions with respect to: motors (left); joints (right)

4 Vex Arm Home Explanation

Robotic arms have a 'home' position which is a fixed reference point for the arm. This point can then be used to measure distances and angles, and in this lab is also used to initialise the arm each time a user program is executed.

In order to specify the home position, each of the 4 potentiometer readings must be recorded while the arm is in the home position. A jig to put the arm in the correct position is used, but the home values needed should be included in the lab handout.

5 Gripper Design

In order to interact with the environment, a modular gripper has been designed and 3D printed. This is mounted to the member 'DE'.

The design features two of the Vex Smart Motors which are coupled directly to pincers - i.e. the Smart Motor's rotation is used to control each pincer. This then allows two methods of control:

1. Position - each motor has an internal encoder which can be used to track the current rotation, and therefore used as a PID reference.

2. Torque - the current motor torque can be read and used as a PID reference.

In order to use PID controllers, first the angles and directions need to be defined. Figure 7 shows the direction of rotation when a positive voltage is applied to each motor.

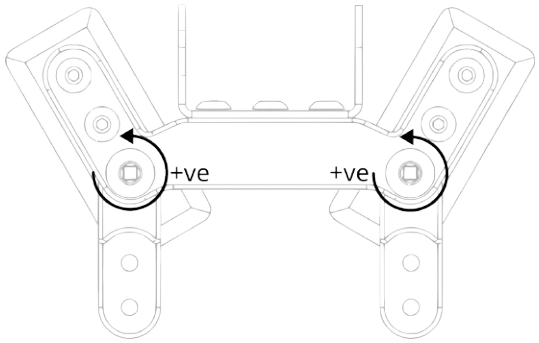


Figure 7: Vex Smart Motor positive directions

The angles for each motor are then defined as in Figure 8. Instead of just controlling each gripper independently, the average position θ_{AVG} can also be used as a control reference:

$$\theta_{AVG} = \frac{\theta_L + (-\theta_R)}{2} \tag{1}$$

Note that the angle θ_R will be negative when read from the Smart Motor due to the sign convention.

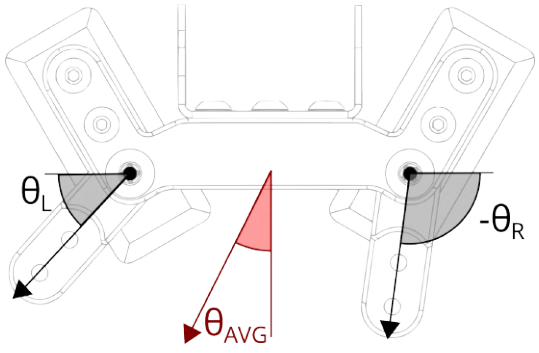
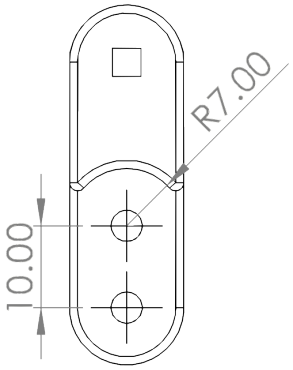


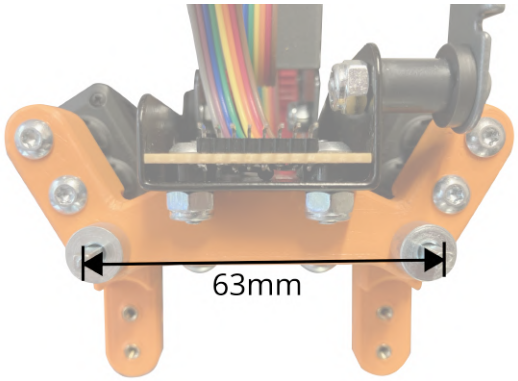
Figure 8: Gripper angle definitions

6 Gripper Dimensions

The gripper design consists of two main parts: the base which is used to mount the motors and connect the system to the Vex arm, and the two pincer mounts. Each pincer mount has $2 \times M3$ threaded holes to allow the pincer design to be easily changed. The dimensions for the pincer mounts are shown in Figure 9a which can be used to model and 3D print custom designs.



(a) Pincer mounting hole dimensions



(b) Gripper pivot spacing

The motor shaft spacing is also shown in Figure 9b. This is the point at which each gripper arm rotates about and so can be used to calculate applied forces from the motor torque measurement.

7 Force Sensors

Although the Vex Smart Motor torque can be read directly, additional force sensing circuitry has been made to improve low force readings and thus increasing sensitivity. The sensors chosen are the Ohmite FSR (datasheet link) in 2 sizes: $\varnothing 5.6mm$ and $\varnothing 14.7mm$.

Figure 10: Ohmite FSR (Force Sensitive Resistors)

These have a resistance that has a linear log-log relationship with the applied force - when the force increases their resistance decreases:

$$\log(F) = m\log(R) + \log(K) \tag{2}$$

The resistance can not be directly measured using the Vex Brain and so the FSRs are connected with another resistor in a potential divider configuration. This then allows the output analog voltage to be read by the Vex Brain's 3-Wire Port.

8 Force Sensor Circuit and Function

The potential divider used to read the sensor's resistance will be affected by the input impedance of the Vex Brain's ADC and so an opamp (TLV2371P) buffer is added to improve this due to its high input impedance and low output impedance. In addition to this, each circuit contains a power indicator LED to provide quick feedback that the circuit is connected correctly to the Vex Brain.

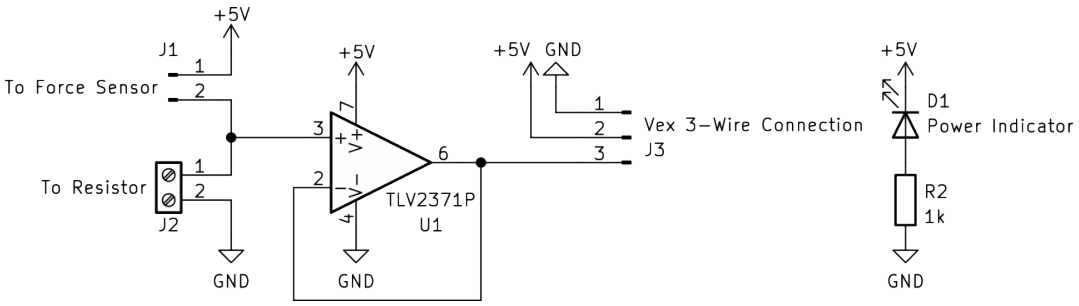
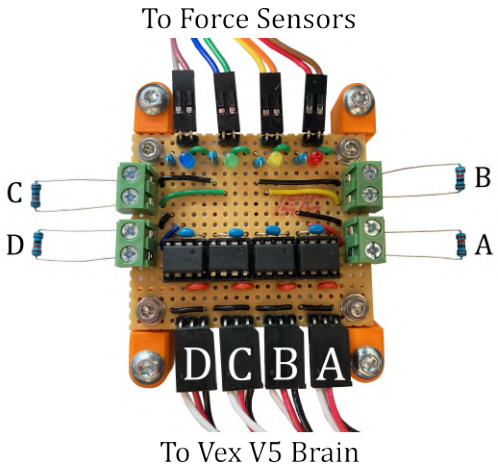
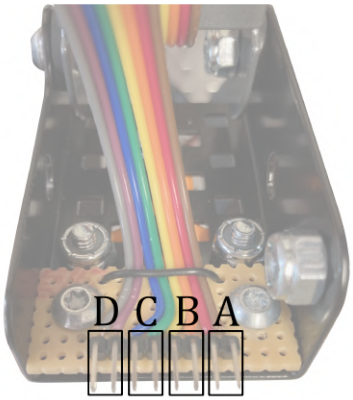


Figure 11: Schematic for one force sensor (4 identical circuts used)

The actual circuit is located next to the Vex V5 Brain. A ribbon cable passes up the Vex Arm to a series of pin headers to which the force sensors can be connected. The other side of the board connects cables to the 3-Wire ports in order to measure the voltage, and ultimately calculate the resistance of the force sensor. Figure 12a shows the complete vero board circuit - the 4 terminal blocks (2 either side) allow for different potential divider resistances to be added. This allows the sensitivity range to be adjusted, though the resistance needs to be input in the corresponding Simulink block (Force Sensor Read).



(a) Force sensor vero board layout



(b) Pin header at end of arm for connecting force sensors

9 Simulink Overview

Simulink is a block diagram environment which can be used to simulate and design systems without having to write code (MATLAB Documentation). The purpose of this is to quickly write and iterate models for systems constructed from Vex parts. Blocks are placed in the model where they can be configured and connected together with 'wires', and the model is ultimately compiled into C++ and uploaded to the Vex V5 Brain. This section details the process of creating and uploading a model, configuration of block parameters within the 'mask', and finally an overview of the Simulink library designed for the Vex hardware.

9.1 Creating and Uploading Models

Figure 13 shows a template model provided to simplify the model making process. This file has all the correct hardware settings and contains some basic blocks to get started - the arm and gripper position to home the arm initially (as mentioned in Section 4), and an enabled subsystem in which a custom program can be made.

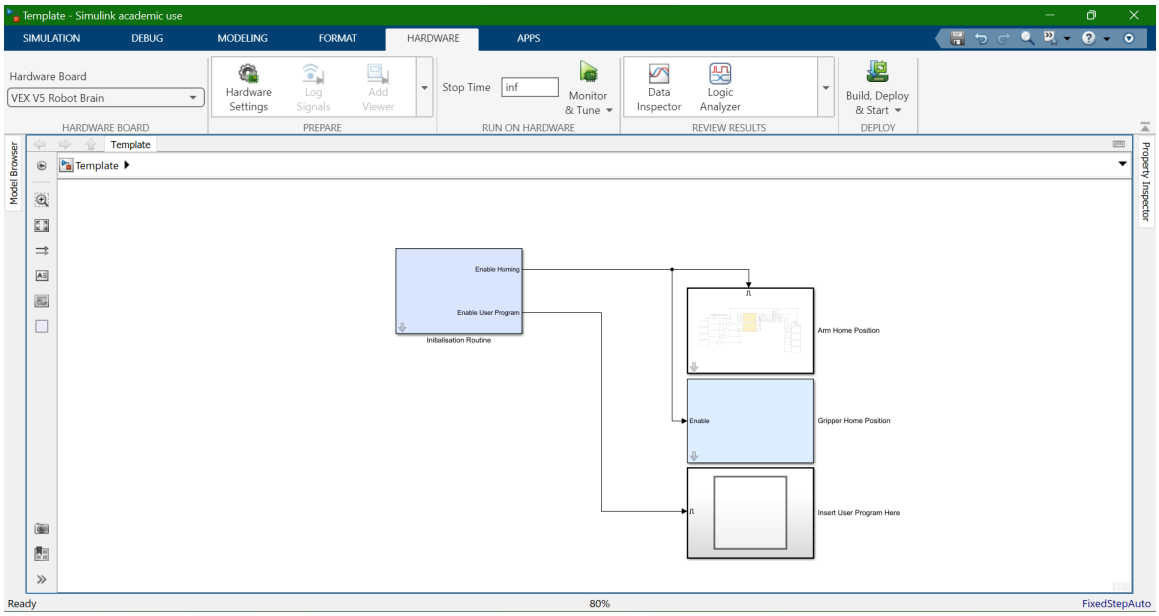


Figure 13: Template model in Simulink

Double clicking on the enabled subsystem (labelled 'Insert User Program Here') will open it - this is where all model creating is to be done. To start inserting blocks click on the 'Library Browser' as shown in Figure 15a which will open the library browser popup. A custom library has been made specifically for this lab labelled '4M25 Lab BLocks' and should cover all functionality needed.

Blocks are inserted into the model by dragging them from the library browser. Double clicking the block will then bring up the mask which provides a brief explanation and a series of parameters that need to be input. To connect blocks together click and drag from any of the ports to begin creating a wire which can then be snapped to another block.

Double-clicking on a block once it has been placed will bring up its mask. This allows parameters in the block's internal model to be changed easily without having to directly

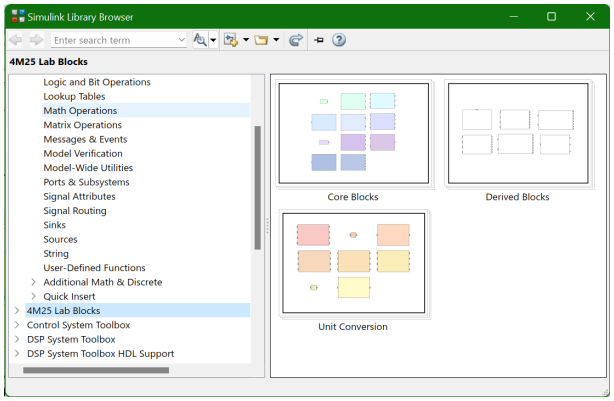
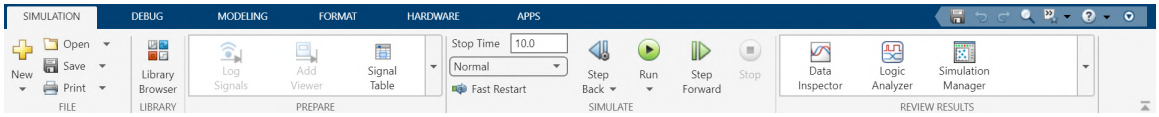
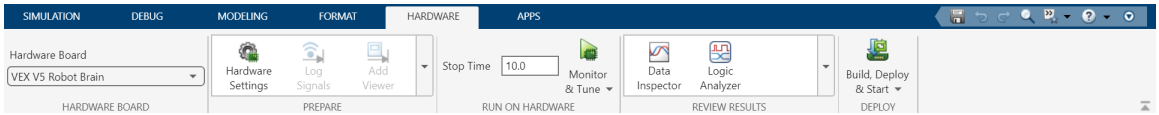


Figure 14: Simulink library popup window



(a) Simulink simulation tab



(b) Simulink hardware tab

Figure 15

interact with it. Usually this is something fixed like a PID gain or the Smart Port a device is connected to, though the parameters are explained in each block's description.

To upload the program to the Vex Brain you must access the 'Hardware' tab as shown in Figure 15b. The 'Build, Deploy & Start' option will compile the model, upload it to the Vex Brain and run it.

9.2 4M25 Simulink Library

A library in Simulink is a series of blocks with related function, contained in one location. Although Vex/MathWorks provide their own library, its use requires experience in Simulink and so the 4M25 library has been created to act as a higher level wrapper with no prior knowledge needed (aside from the course contents). It is split into 3 main areas as shown in Figure 16:

1. Core Blocks: These are the lowest level blocks which perform simple functions.
2. Derived Blocks: Constructed from Core Blocks and Unit Conversion, this section performs more complicated functions with a simple interface.
3. Unit Conversion: They are used to convert between different conventions, be it angle definitions or units.

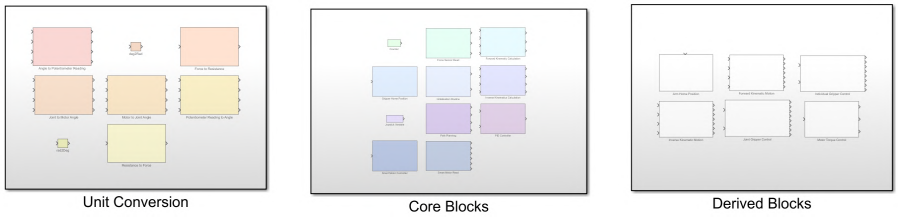
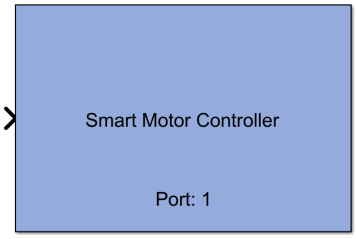


Figure 16: Library subsections

This section will show each of the blocks in question, explain the parameters present in their masks and give a brief overview of how their internal model is constructed.

9.3 Core Blocks

9.3.1 Smart Motor Controller



This block is used to control the Vex Smart Motors. These can be controlled in 2 methods:

1. Velocity: Input a velocity in RPM between -200 and 200.
2. Advanced: Input a voltage in mV between -12000 and 12000.

There are also 3 braking methods:

1. Coast: no braking method.
2. Brake: an internal brake is applied to the motor when stationary to try prevent movement.
3. Hold: a voltage is applied to the motor to fix its current position.

For the purposes of this lab, the setting should generally be set to 'Coast' as the PID controllers should be capable of holding the motor in place. The Smart Port which the Smart Motor is connected to must also be selected.

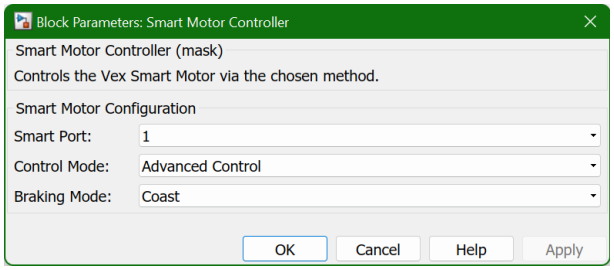


Figure 17: 'Smart Motor Controller' block mask

The model for this block consists only of a MATLAB System provided by MathWorks/Vex which allows for direct voltage control of the motor (this is not available with the standard Vex library).

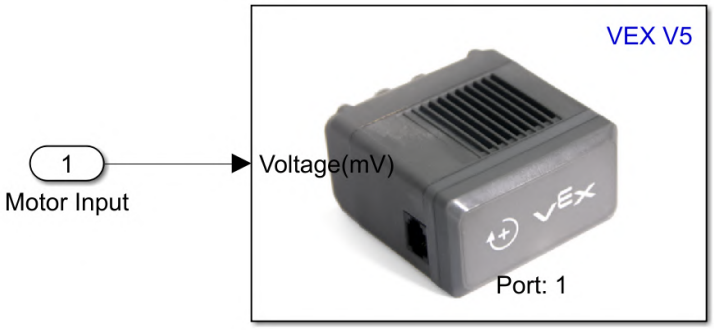
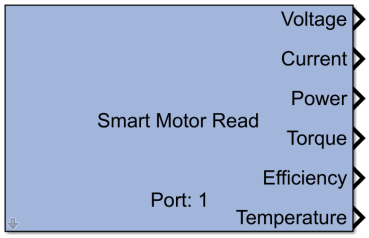


Figure 18: 'Smart Motor Controller' internal model

9.3.2 Smart Motor Read



This block reads several attributes relating to the Vex Smart Motors for use in control. Each output has units as follows:

- Voltage: mV
- Current: mA
- Power: ?
- Torque: Nm
- Efficiency: %
- Temperature: °C

The 'Sample Time' parameter dictates how often each of these values is read and is by default set to 0.01s. The Smart Port which the Smart Motor is connected to must also be selected.

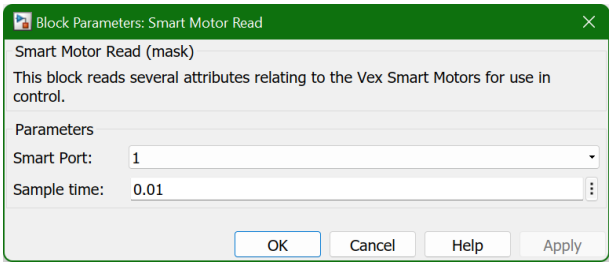


Figure 19: 'Smart Motor Controller' block mask

The model for this block consists only of a MATLAB System provided by MathWorks/Vex which allows for all motor parameters to be read (this is not available with the standard Vex library).

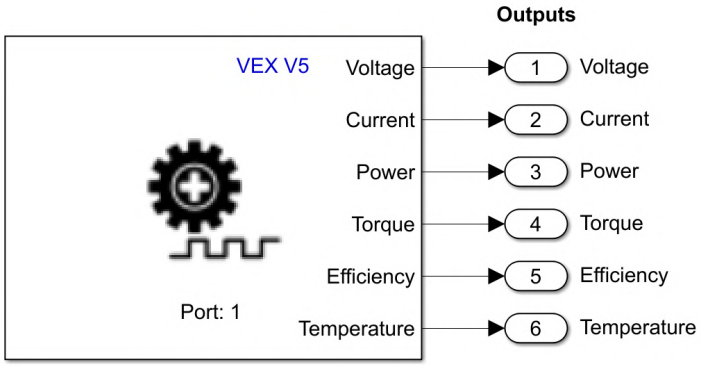
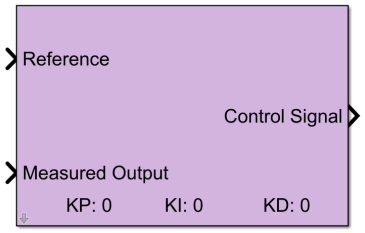


Figure 20: 'Smart Motor Read' internal model

9.3.3 PID Controller



The block is a general PID controller requiring an input reference and measured value. The mask has 2 sections:

1. The PID gains KP, KI and KD
2. The ability to limit the output of the PID controller

The PID gains are defined as expected. If the 'Limit Output' box is ticked, the PID block will have an output limited by the upper/lower limits specified. The 'Anti-windup Method' is used to prevent errors in the integrator while the system is saturated - the 3 methods are 'none', 'back-calculation' (using Kb) and 'clamping'.

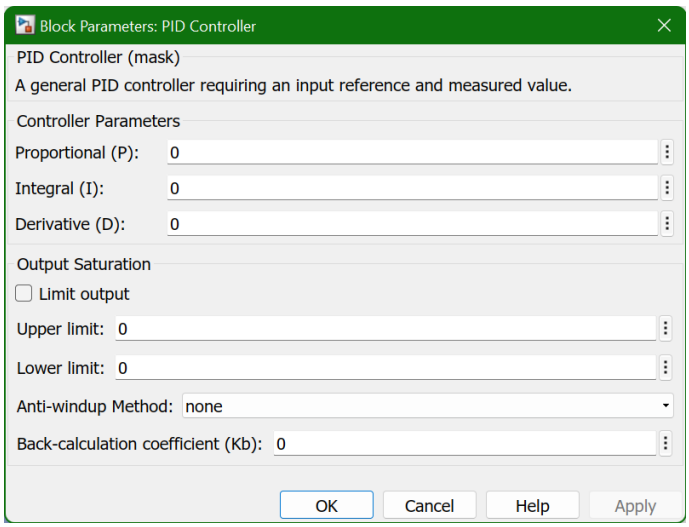


Figure 21: 'PID Controller' block mask

The internal model is constructed as follows:

(A) Calculate error by subtracting Measured Output from Reference

(B) PID controller which handles each of the proportional, integral and derivative calculations.

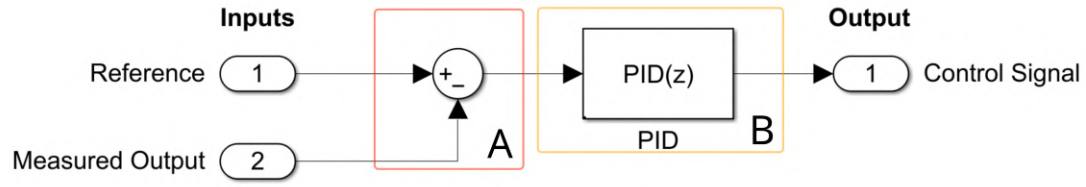
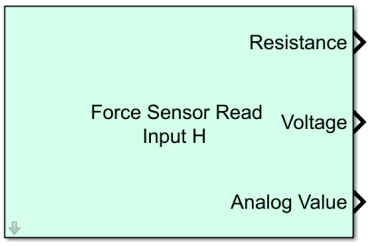


Figure 22: 'PID Controller' internal model

9.3.4 Force Sensor Read



Reads the analog input corresponding to the force sensor and outputs the analog reading, input voltage and sensor resistance.

The mask requires the 3-Wire Port that the sensor is connected to, the 'Sample Time' that readings are taken at and the 'Resistance' that is inserted in the circuit's terminal block.

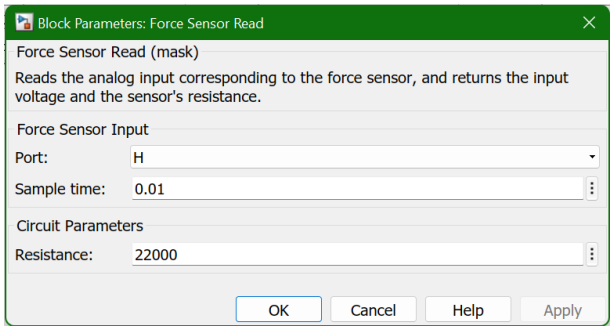


Figure 23: 'Force Sensor Read' block mask

The internal model is constructed as follows:

(A) Read analog input which the force sensing circuit is connected to.

(B) Convert analog input into a voltage by dividing by the max value (4095) and multiplying by the voltage source (5V).

(C) Calculate the force sensor's resistance using the inverse potential divider formula.

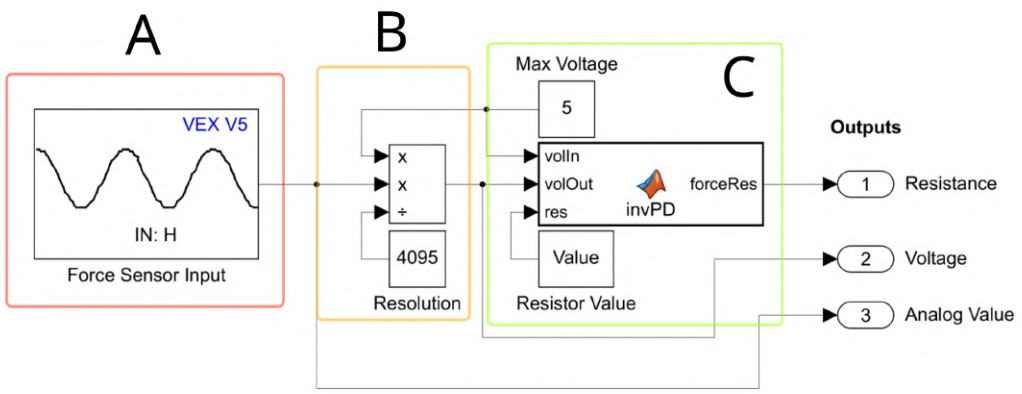
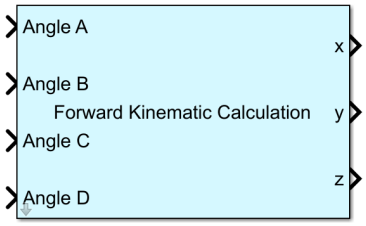
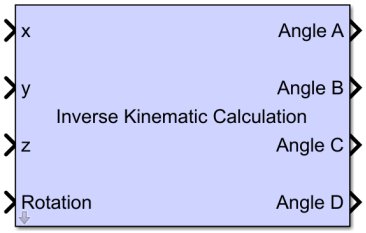


Figure 24: 'Force Sensor Read' internal model

9.3.5 Forward and Inverse Kinematic Calculation





The kinematics blocks allow conversion between pose (4 angles in radians) and Cartesian coordinates (mm). The masks for these blocks do not require any parameters as they are specific to the Vex Arm.

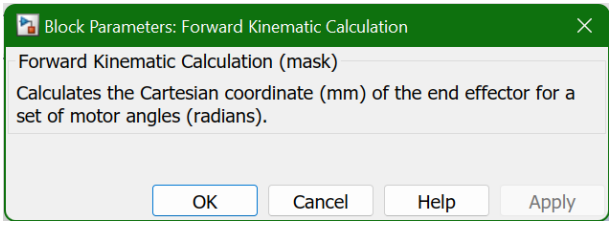


Figure 25: 'Forward Kinematic Calculation' block mask

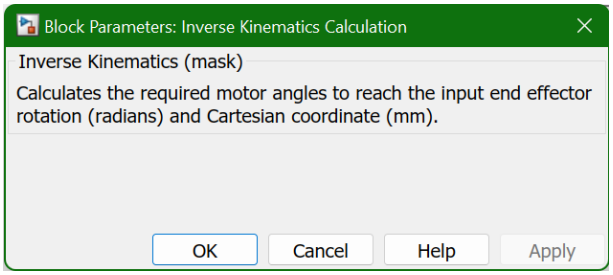
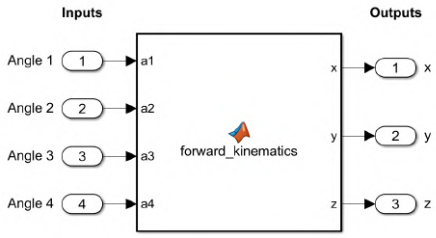
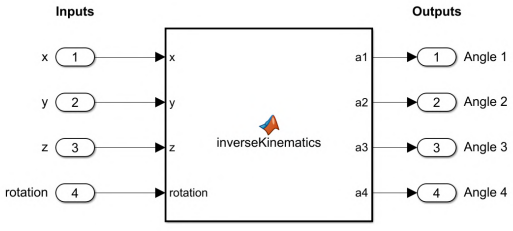


Figure 26: 'Inverse Kinematic Calculation' block mask

The kinematics models consist only of a MATLAB function with the inputs and outputs as shown on the block itself. The code for these is provided in the Appendix.

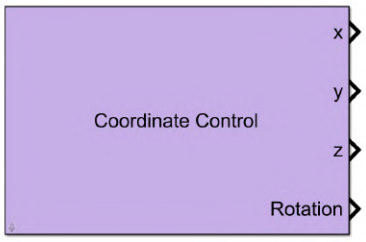


(a) 'Forward Kinematics Calculation' model



(b) 'Inverse Kinematics Calculation' model

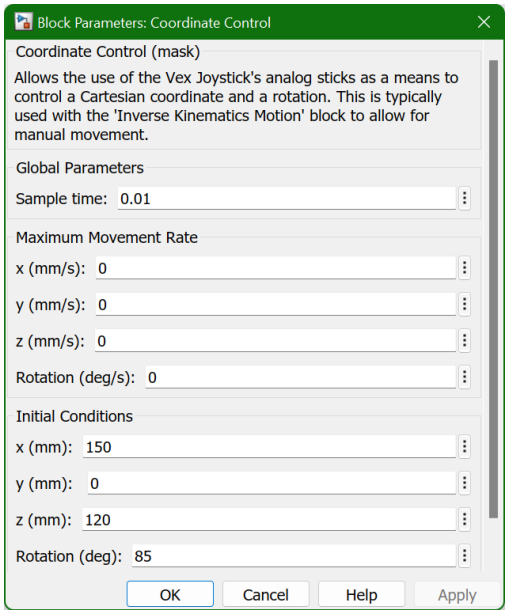
9.3.6 Coordinate Control



Allows the use of the Vex Joystick's analog sticks as a means to control a Cartesian coordinate and a rotation. This is typically used with the 'Inverse Kinematics Motion' block to allow for manual movement.

The mask has 3 main sections:

1. Global Parameters: the sample time dictates how often the Vex Joystick inputs are read.
2. Maximum Movement Rate: these values dictate how fast the target changes when each stick is in it maximum position.
3. Initial Conditions: when the program starts, this is the initial target the arm will try to reach.



(a) 'Coordinate Control' block mask

(b) 'Coordinate Control' joystick control scheme

Figure 28

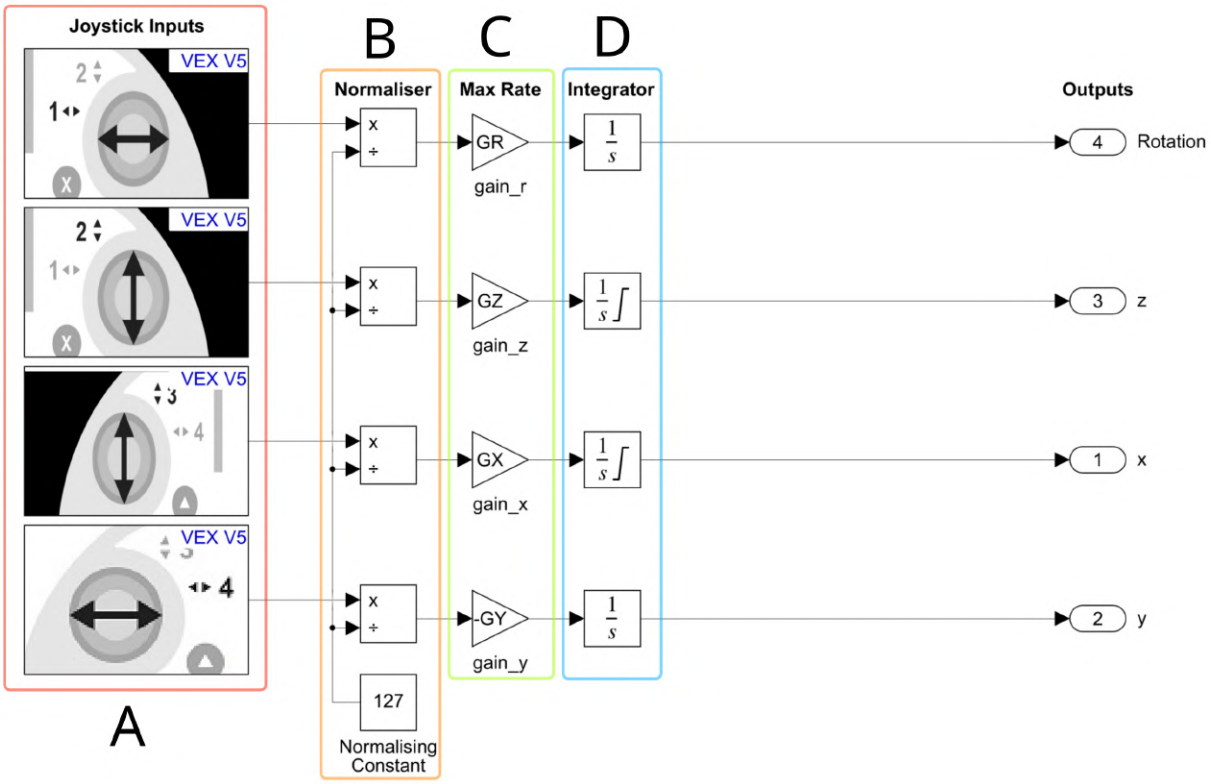
The internal model is constructed as follows:

(A) Reads the analog stick inputs from the Vex joystick.

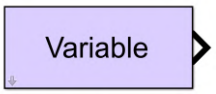
(B) The readings range from -128 to 127 so these are normalised by dividing by 127.

(C) The max rate corresponds to a normalised value of 1, so each normalised value is multiplied by the respective max rate.

(D) By integrating this over time, the target is stored.



9.3.7 Joystick Variable



Allows a variable to be incremented/decremented using two specified buttons on the Vex Joystick.
The mask has 2 sections:

1. Joystick Parameters: the 'Increment Button' and 'Decrement Button' define what joystick buttons are used to control the variable, with the 'Sample time' dictating how often their states are read.

2. Variable Parameters: The 'Rate' is how quickly the value changes when the buttons are pressed, the initial value is the value at program start and finally the limits can be specified.

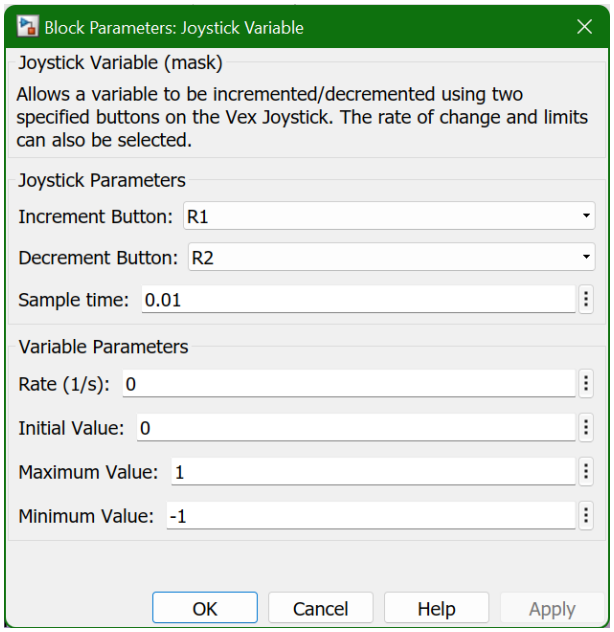


Figure 29: 'Joystick Variable' block mask

The internal model is constructed as follows:

(A) Reads the button inputs from the Vex joystick.

(B) Subtract the inputs such that the output is 1 for increment, -1 for decrement and 0 if either both or none are pressed.

(C) The output is multiplied by the max rate.

(D) By integrating this over time, the target is stored.

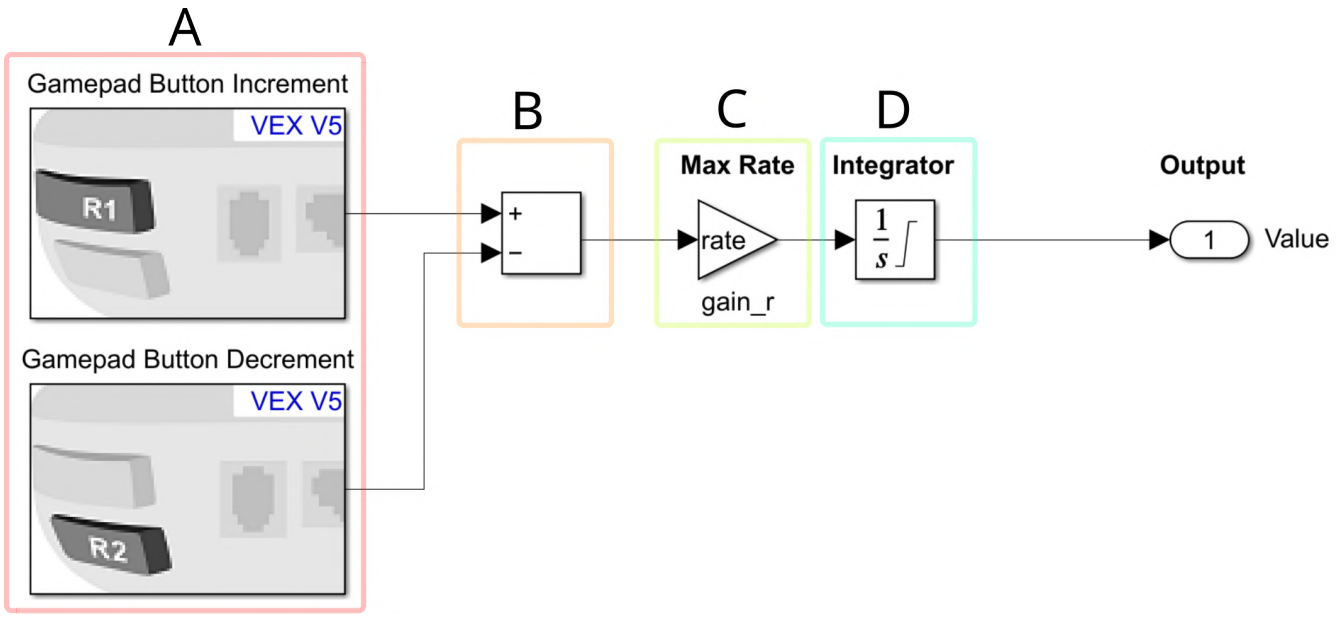
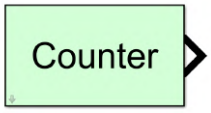


Figure 30: 'Joystick Variable' internal model

9.3.8 Counter



Outputs a count value which increments based on the specified cycle time, and resets at the specified limit. This is useful for cycling through position/torque targets.
The mask requires 3 parameters:

1. Sample Time: the sample time the model is running at (typically 0.01s).
2. Cycle Time: the time in seconds it takes for the count to increment by 1.
3. Number of Cycles: the maximum value before the counter resets.

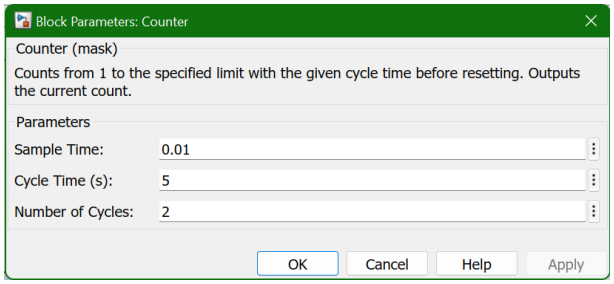


Figure 31: 'Counter' block mask

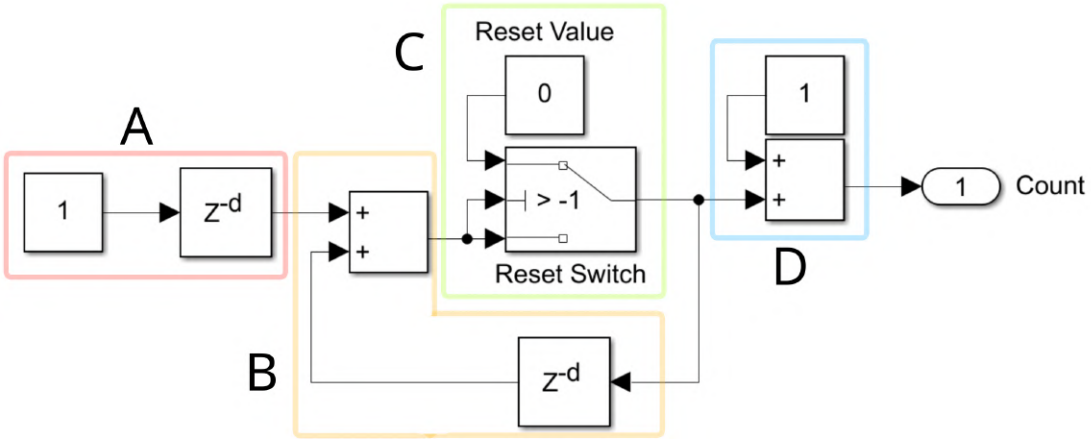
The internal model is constructed as follows:

(A) Input must be delayed so that the counter correctly begins at 0.

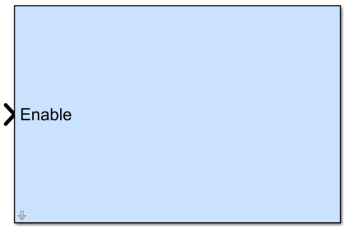
(B) Increment each cycle time (shown as -d).

(C) Switch output to 0 when the limit value is reached, resetting the counter.

(D) The counter begins at 0, and so 1 is added to match the MATLAB numbering convention.



9.3.9 Gripper Home Position



Given an enable input, moves the gripper to the home position and resets the motor encoders.
The mask has 2 sections:

1. Global Parameters: 'Sample Time' dictates how often the motor encoders are read.
2. Smart Motor Ports: the Smart Ports that the left and right gripper motors are connected to.

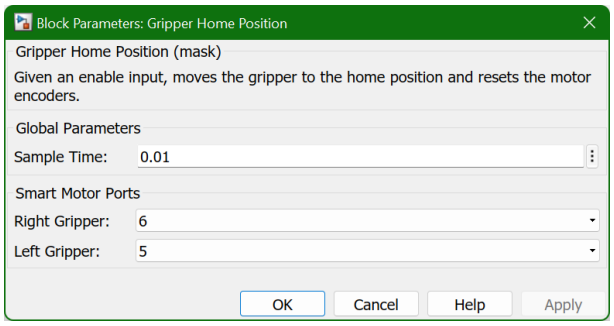


Figure 32: 'Gripper Home' block mask

The internal model is constructed as follows:

(A) 4.5 seconds after the homing procedure begins, reset the gripper motor encoders as they have reached the home position.

(B) Read the motor encoder positions (in degrees).

(C) If the position hasn't changed by at least 2 degrees in the last 0.3 seconds, assume the gripper has stopped and reached the home position.

(D) When the home position is reached, output a constant 1.

(E) Set motor velocities to 5 until the home position signal from (D) is received, then set the velocity to 0.

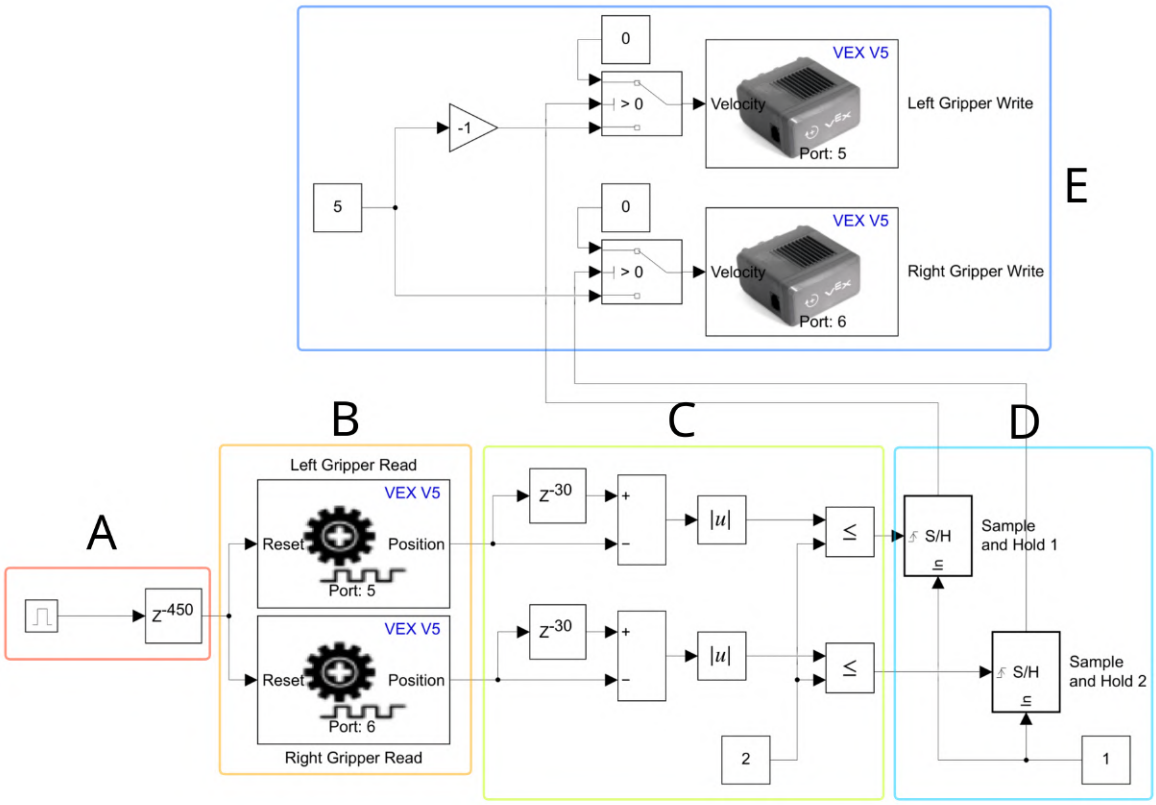
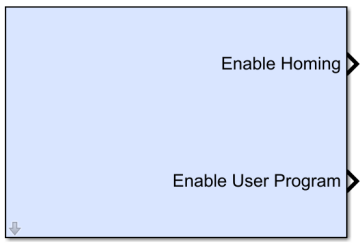
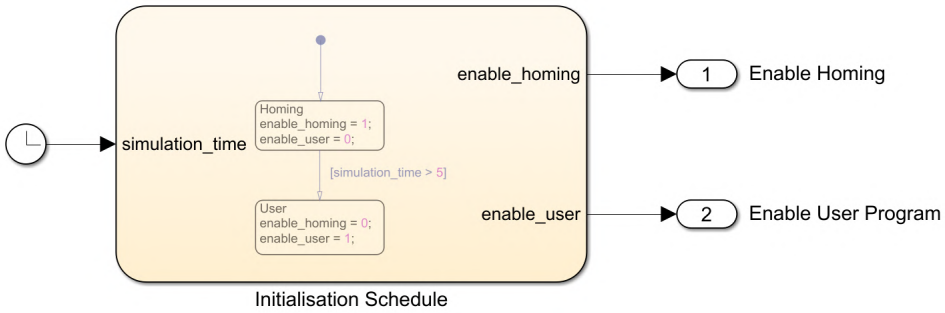


Figure 33: 'Gripper Home' internal model

9.3.10 Initialisation Routine



Used to help home the Vex Arm prior to beginning the User Program. The 'Enable Homing' output is high for the first 5 seconds before switching the 'Enable User Program' output to high.

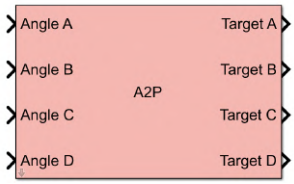


(A) On program start move to 'Homing' state and set `enable_homing` to 1, `enable_user` to 0.

(B) After `simulation_time` exceeds 5 seconds move to 'User' state and set `enable_homing` to 0, `enable_user` to 1.

9.4 Unit Conversion

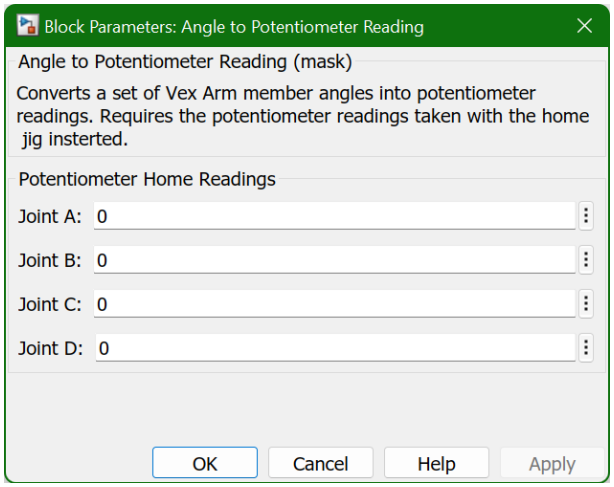
9.4.1 Angle to Potentiometer



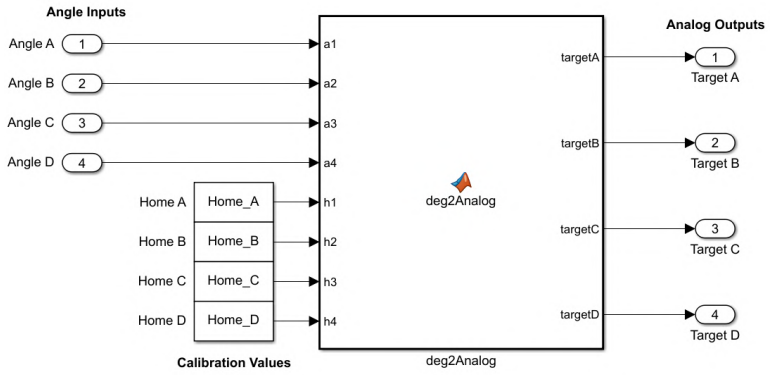
Converts a set of Vex Arm member angles into potentiometer readings using the following:

1. Subtract home angles (radians) from desired position (radians).
2. Scale by the potentiometer's bits/radian ratio and the joint gear ratios.
3. Subtract this from the home potentiometer reading.

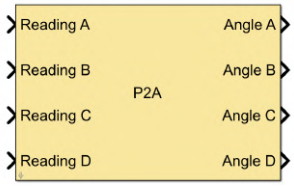
The mask inputs are home potentiometer readings for each of the 4 joints.



The internal model shown below is largely a MATLAB function (`deg2Analog`) with inputs and outputs. The inputs `Home_X` are passed into the function as the arm home positions, the values of which are edited in the block mask.



9.4.2 Potentiometer to Angle



Converts a set of potentiometer readings into Vex Arm member angles using the following:

1. Subtract reading from home positon (radians).
2. Scale by the potentiometer's bits/radian ratio and the joint gear ratios.
3. Add to the home angle reading.

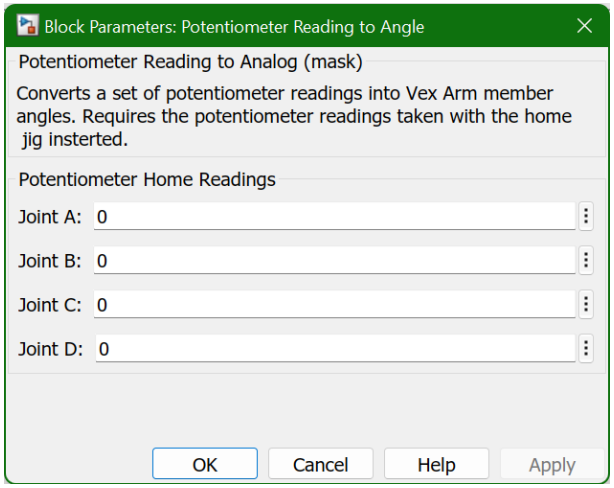
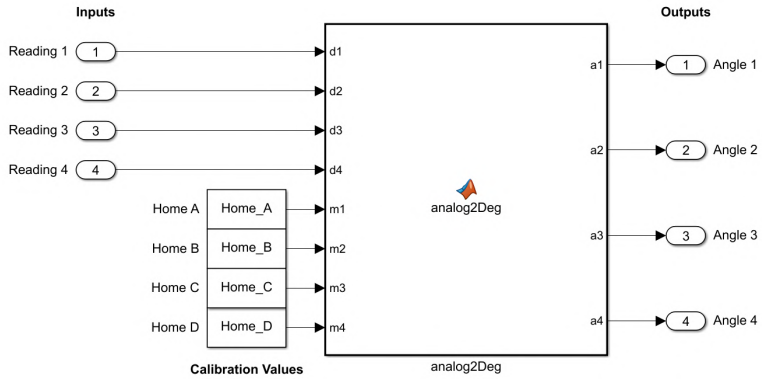
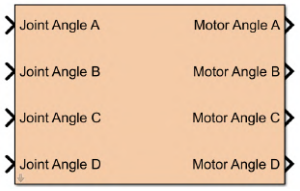


Figure 34: 'Potentiometer to Angle' block mask

The internal model shown below is largely a MATLAB function (`deg2Analog`) with inputs and outputs. The inputs `Home_X` are passed into the function as the arm home positions, the values of which are edited in the block mask.



9.4.3 Joint to Motor Angle



The potentiometer readings used to calculate the Vex Arm's current position are located at the base and are coupled to the shaft of each motor. This block is used to obtain these motor angles from the relative joint angles (see Figure 6).
The mask for this block does not require any parameters as it is specific to the Vex Arm.

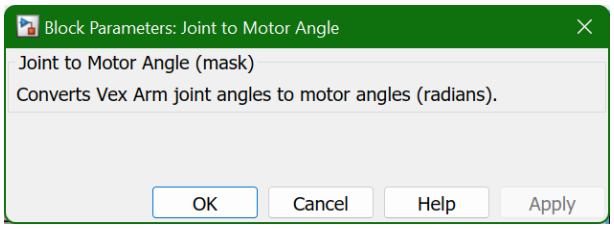


Figure 35: 'Joint to Motor' block mask

The internal model shown below is a MATLAB function (`jointToMotor`) with inputs and outputs. The code for this is supplied in the Appendix.

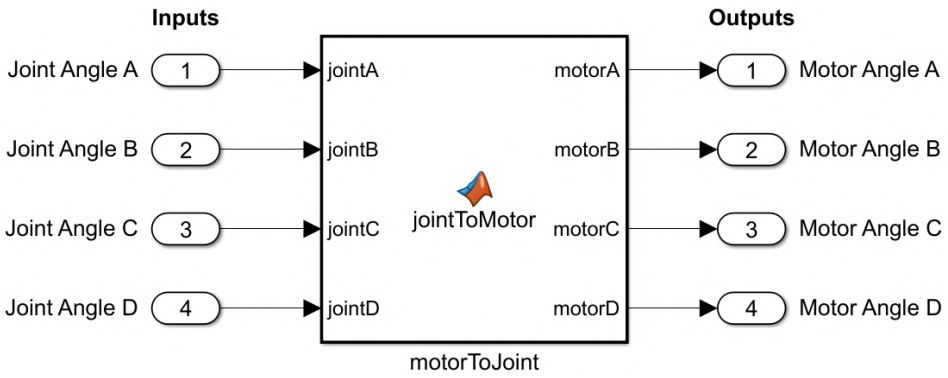
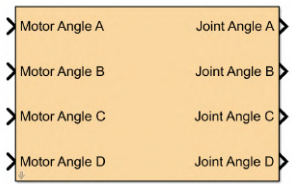


Figure 36: 'Joint to Motor' internal model

9.4.4 Motor to Joint Angle



The potentiometer readings used to calculate the Vex Arm's current position are located at the base and are coupled to the shaft of each motor. To instead have the angle of each joint relative to the previous, they must be converted using this block (see Figure 6).
The mask for this block does not require any parameters as it is specific to the Vex Arm.

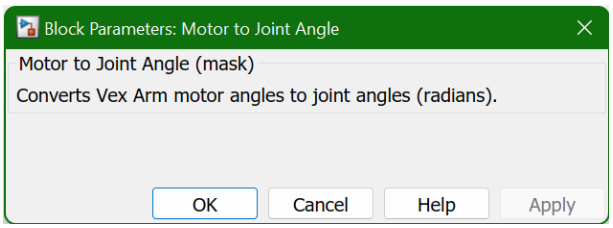


Figure 37: 'Motor to Joint' block mask

The internal model shown below is a MATLAB function (`motorToArm`) with inputs and outputs. The code for this is supplied in the Appendix.

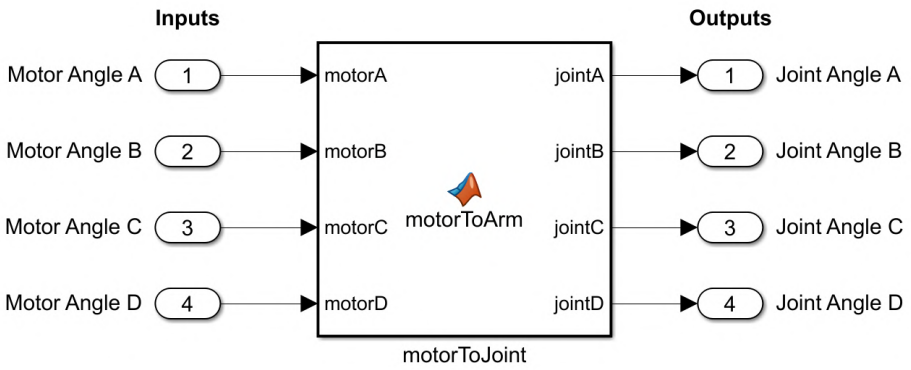


Figure 38: 'Motor to Joint' internal model

9.4.5 Degree/Radian Conversion

The above blocks convert either degrees to radians (left) or radians to degrees (right). These can be useful if degrees are more intuitive to work with, as all blocks expect an input in radians and typically output radians.
The masks for these blocks require no input as they are a simple unit conversion.

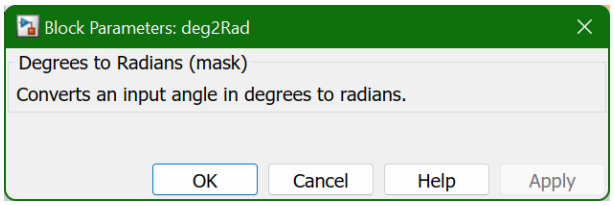


Figure 39: 'Degrees to Radians' block mask

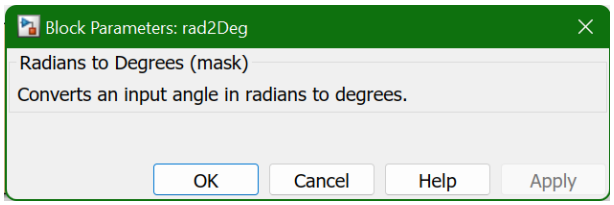


Figure 40: 'Radians to Degrees' block mask

The internal models are shown below. They consist of a division by the current unit and multiplication of the desired unit.

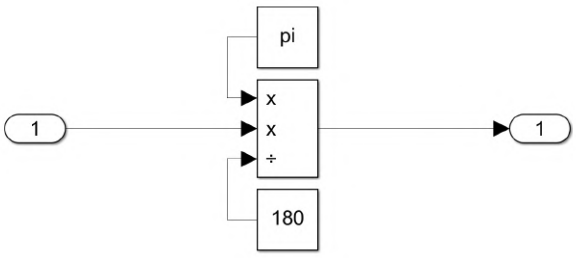


Figure 41: Degrees to radians model

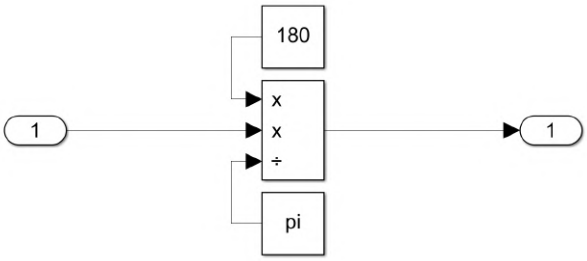
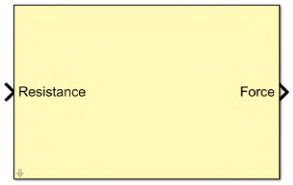


Figure 42: Radians to degrees model

9.4.6 Resistance to Force



Converts the resistance of a force sensor into a force. The Ohmite force sensitive resistors have a linear log-log relationship between Force (N) and Resistance (Ω):

$$\log(F) = m\log(R) + \log(K) \tag{3}$$

$$F = KR^m \tag{4}$$

The mask requires only the gradient (m) and constant (K) parameters.

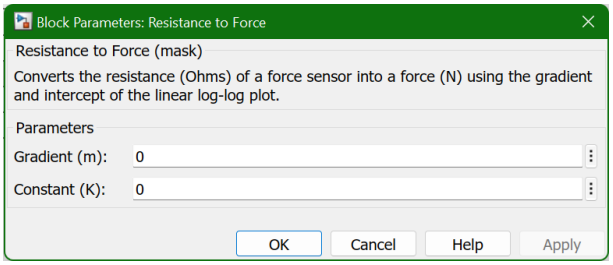


Figure 43: 'Resistance to Force' block mask

The internal model is constructed as follows:

(A) Calculate the term R^m.

(B) Multiply this by the constant K.

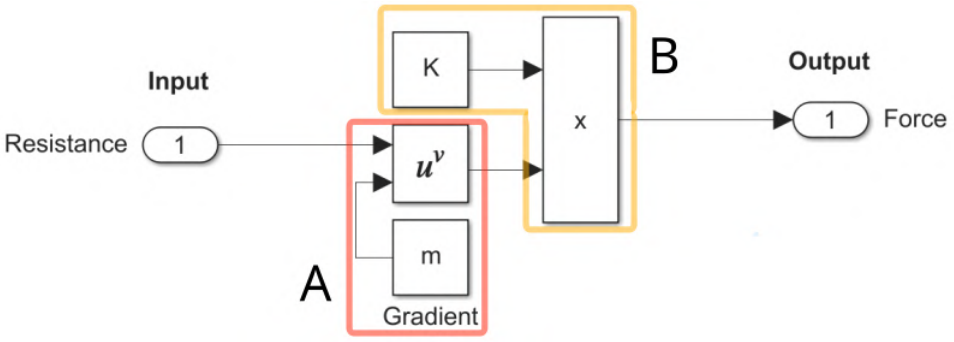
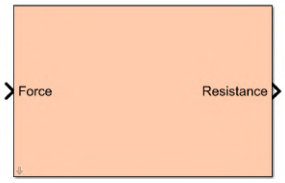


Figure 44: 'Resistance to Force' internal model

9.4.7 Force to Resistance



Converts the resistance of a force sensor into a force. The Ohmite force sensitive resistors have a linear log-log relationship between Force (N) and Resistance (Ω):

$$\log(F) = m\log(R) + \log(K) \tag{5}$$

$$R = \left(\frac{F}{K}\right)^{\frac{1}{m}} \tag{6}$$

The mask requires only the gradient (m) and constant (K) parameters.

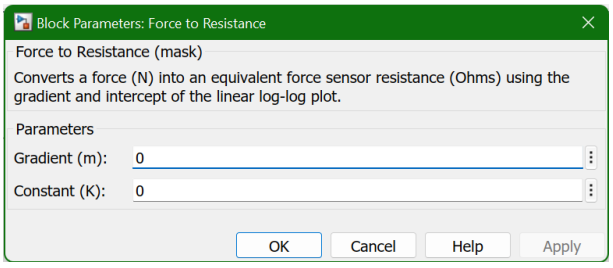


Figure 45: 'Force to Resistance' block mask

The internal model is constructed as follows:

(A) Calculate $\frac{F}{K}$

(B) Calculate $\frac{1}{m}$

(C) Calculate $\left(\frac{F}{K}\right)^{\frac{1}{m}}$

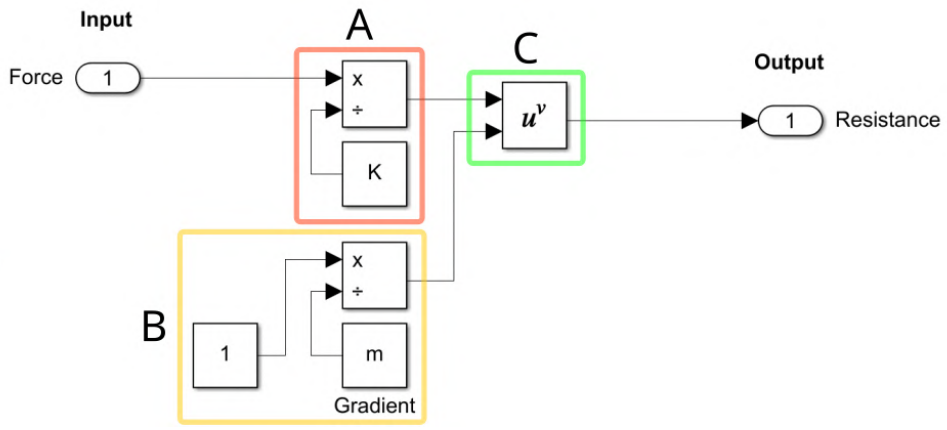
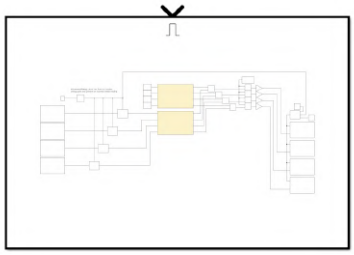


Figure 46: 'Force to Resistance' internal model

9.5 Derived Blocks

9.5.1 Arm Home Position



When the enable input is '1', the block moves each of the 4 Vex Arm joints into the home position as specified in the block mask. When the enable input is '0' the block is inactive. The mask requires the 'Potentiometer Home Readings' which sets the position the Vex Arm will go to in the first 5 seconds of execution. The 'Sample Time' sets how often the potentiometer values are read during the homing procedure.

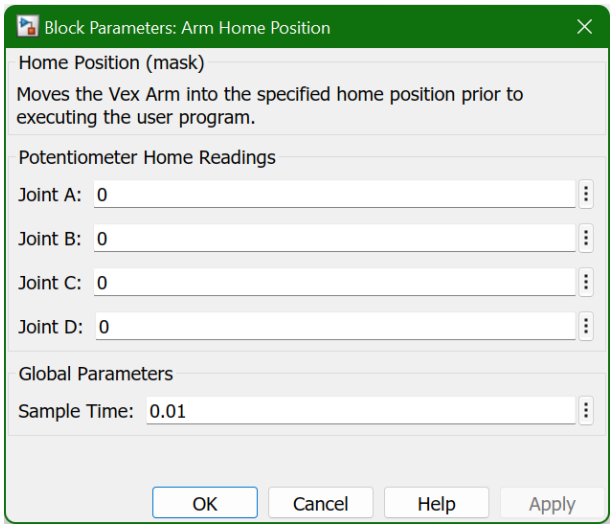


Figure 47: 'Arm Home Position' block mask

(A) Read potentiometer position for each of the 4 Vex Arm joints.

(B) Sample initial readings

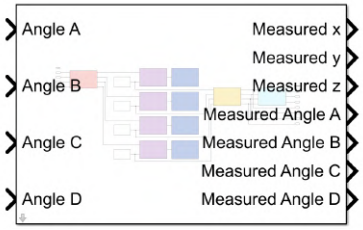
(C) Use initial potentiometer readings to calculate the current arm angles, and the home angles.

(D) Calculate the angle difference between the current position and home position.

(E) Convert the angle into encoder count (out of 1800), scale each reading by the gear ratio (7:1, 3:1, 3:1, 3:1) and finally divide by 2 as the Vex motor encoder assumes a Smart Motor gear ratio of 36:1, but 18:1 is used on the arm.

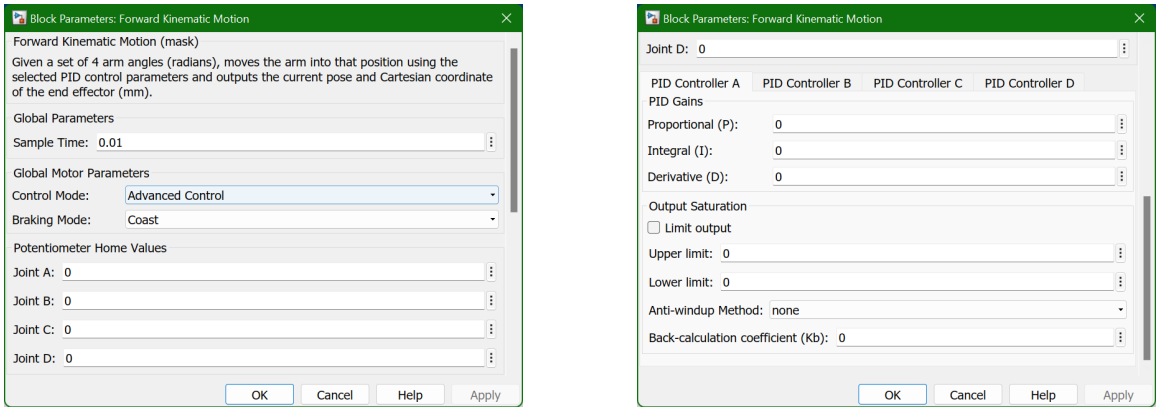
(F) Pass the previous values into each Smart Motor as the target position and switch the velocity to 30 RPM.

9.5.2 Forward Kinematic Motion



Given a set of 4 motor angles (radians), moves the arm into that position using the selected PID control parameters and outputs the current pose and Cartesian coordinate of the end effector (mm).
The mask contains 5 sections:

1. Global Parameters: the 'Sample Time' at which the potentiometers positions are read.
2. Global Motor Parameters: the control and braking mode of the Smart Motors.
3. Potentiometer Home Values: as defined in Section 4.
4. PID Gains: KP, KI and KD for the PID controllers.
5. Output Saturation: PID limits as defined in Section 9.3.3.



(a) (b)

Figure 48: 'Forward Kinematic Motion' block mask

The internal model is constructed as follows:

(A) Converts the angles (radians) into the corresponding potentiometer readings.

(B) Each joint has a PID controller which uses the previous values as a reference, and the analog readings as the measured value.

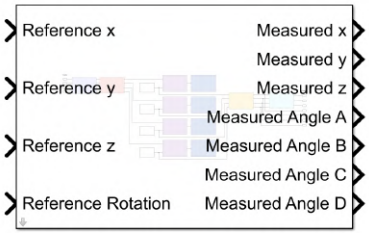
(C) The PID output is used to drive each Smart Motor.

(D) The current analog readings are converted into angles to determine the Vex Arm's current pose.

(E) The pose is then used to calculate the forward kinematics which results in the arm's current Cartesian coordinate and end effector rotation.

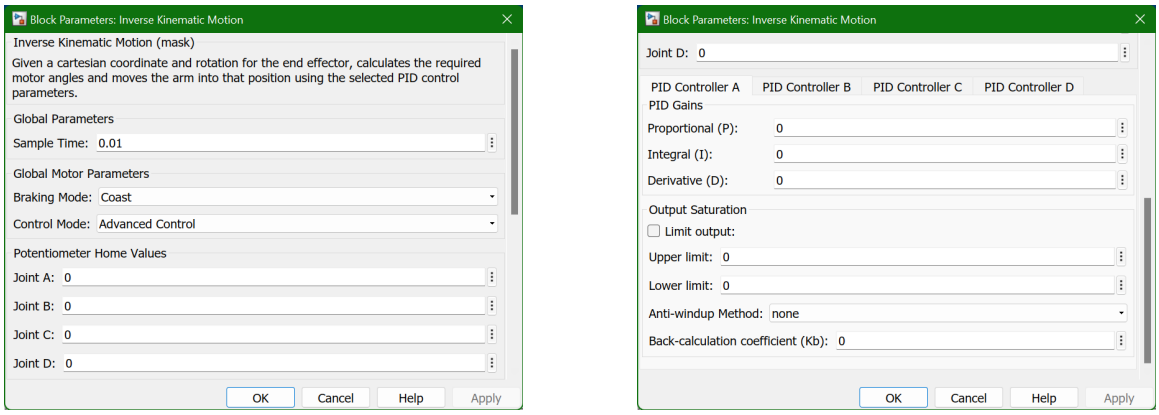
Figure 49: 'Forward Kinematic Motion' internal model

9.5.3 Inverse Kinematic Motion



Given a cartesian coordinate (mm) and rotation (radians) for the end effector, calculates the required motor angles and moves the arm into that position using the selected PID control parameters.
The mask contains 5 sections:

1. Global Parameters: the 'Sample Time' at which the potentiometers positions are read.
2. Global Motor Parameters: the control and braking mode of the Smart Motors.
3. Potentiometer Home Values: as defined in Section 4.
4. PID Gains: KP, KI and KD for the PID controllers.
5. Output Saturation: PID limits as defined in Section 9.3.3.



(a) (b)

Figure 50: 'Inverse Kinematic Motion' block mask

The internal model is constructed as follows:

(A) Calculates the required motor angles to reach the input coordinate and end effector rotation.

(B) Converts the angles (radians) into the corresponding potentiometer readings.

(C) Each joint has a PID controller which uses the previous values as a reference, and the analog readings as the measured value.

(D) The PID output is used to drive each Smart Motor.

(E) The current analog readings are converted into angles to determine the Vex Arm's current pose.

(F) The pose is then used to calculate the forward kinematics which results in the arm's current Cartesian coordinate and end effector rotation.

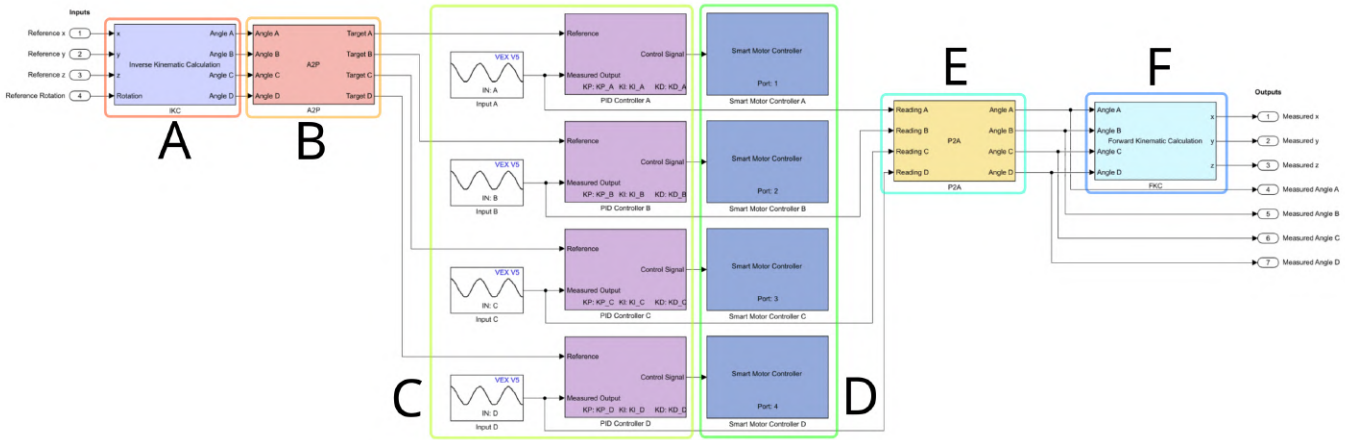
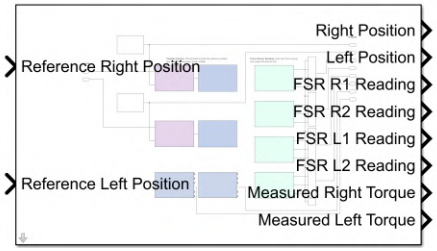


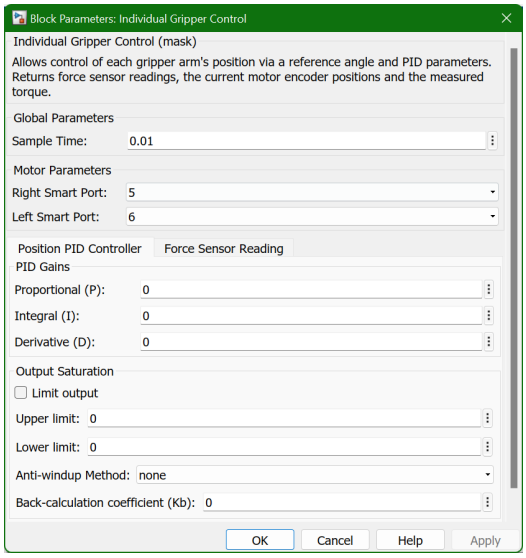
Figure 51: 'Inverse Kinematic Motion' internal model

9.5.4 Individual Gripper Control

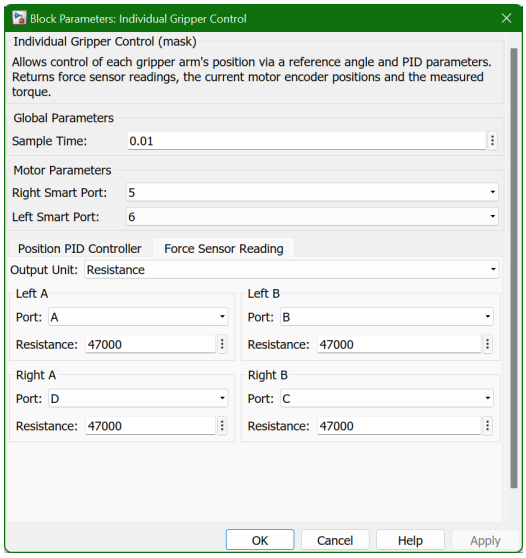


Allows control of each gripper arm's position via a reference angle and PID parameters. Returns force sensor readings, the current motor encoder positions and the measured torque. The mask contains 5 sections:

1. Global Parameters: the 'Sample Time' at which the motor encoder positions are read.
2. Motor Parameters: the Smart Ports correspodning to the right and left motors.
3. PID Gains: KP, KI and KD for the PID controllers.
4. Output Saturation: PID limits as defined in Section 9.3.3.
5. Force Sensor Reading: the output unit (resistance, voltage or analog reading), the 3-Wire port that each sensor is connected to and finally the resistance in each circuit's terminal block (see Section 8).



(a)



(b)

Figure 52: 'Individual Gripper Control' block mask

The internal model is constructed as follows:

(A) Each gripper arm has a PID controller using the provided input as a reference and the motor encoder value as the measurement.

(B) The PID output is passed to the Smart Motor controllers

(C) The torque for each motor is also measured to provide a useful output when gripping an object.

(D) Each of the 4 force sensors is read. Rather than output the resistance, voltage and analog reading the user can specify which, hence the switches.

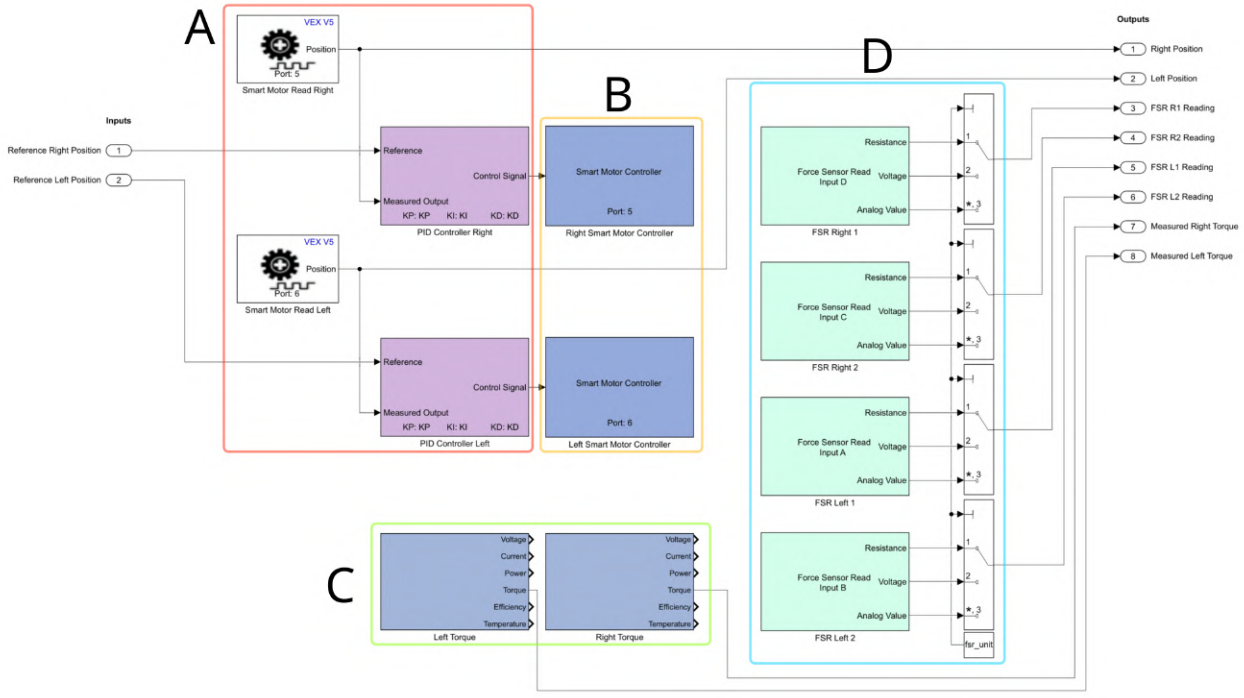
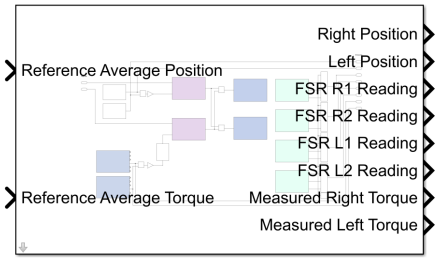


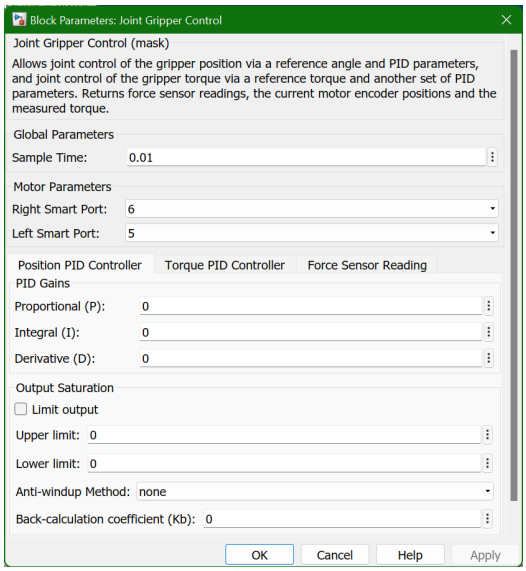
Figure 53: 'Individual Gripper Control' internal model

9.5.5 Joint Gripper Control

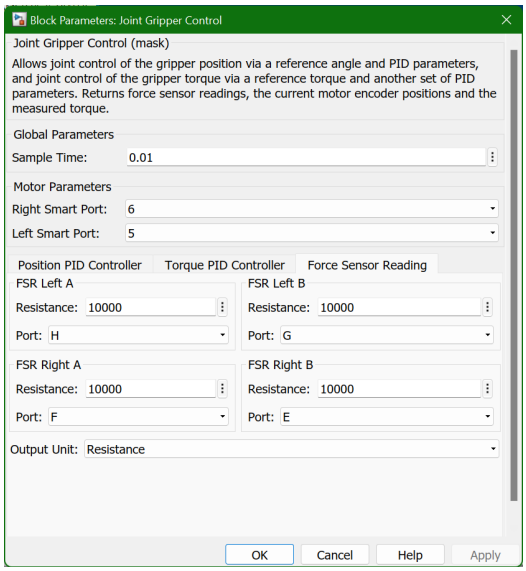


Allows control of the average gripper arm position and average gripper torque through two PID controllers. Returns force sensor readings, the current motor encoder positions and the measured torque.
The mask contains 5 sections:

1. Global Parameters: the 'Sample Time' at which the motor encoder positions are read.
2. Motor Parameters: the Smart Ports correspodning to the right and left motors.
3. Position PID Controller: KP, KI and KD for the PID controllers, and PID limits as defined in Section 9.3.3.
4. Torque PID Controller: same parameters as above.
5. Force Sensor Reading: the output unit (resistance, voltage or analog reading), the 3-Wire port that each sensor is connected to and finally the resistance in each circuit's terminal block (see Section 8).



(a)



(b)

Figure 54: 'Joint Gripper Control' block mask

The internal model is constructed as follows:

(A) Read each arm's position (degrees).

(B) Read each arm's torque (Nm).

(C) Calculate the average position and average torque for the PID reference.

(D) The torque reading is relatively noisy, so a low pass filter is applied.

(E) PID controllers are then used for position and torque.

(F) The PID outputs for position and torque are summed (torque subtracted from the right motor due to the sign convention).

(G) The PID outputs are then fed into the Smart Motors.

(H) Each of the 4 force sensors is read. Rather than output the resistance, voltage and analog reading the user can specify which, hence the switches.

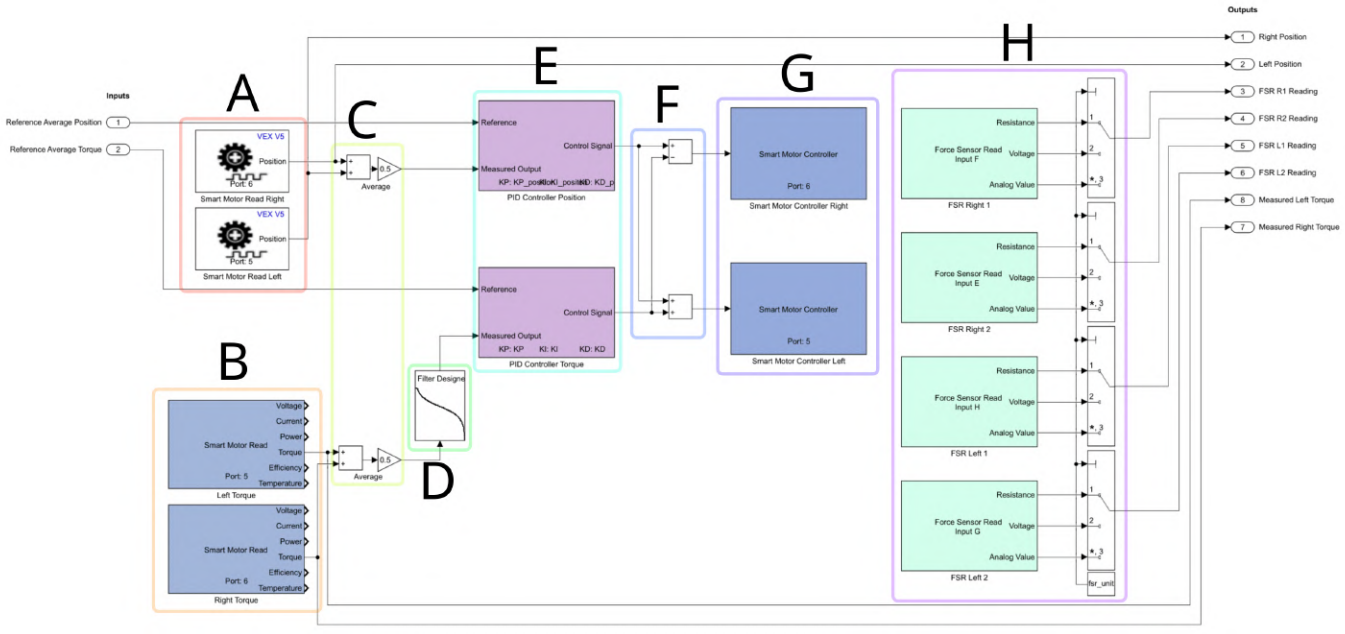
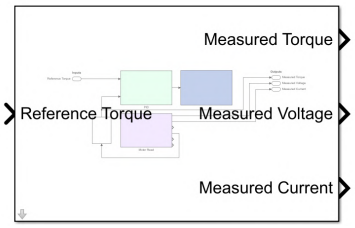


Figure 55: 'Joint Gripper Control' internal model

9.5.6 Motor Torque Control



Uses a PID controller to match the reference torque (Nm) of the specified Smart Motor. The PID controller output is the voltage sent to the Smart Motor.
The mask has 3 sections:

1. Smart Motor Parameters: the Smart Port the motor is connected to, and 'Sample Time' which is how often the Smart Motor torque is read.
2. PID Controller: the PID gains KP, KI and KD.
3. Output Saturation: the ability to limit the output of the PID controller as defined in Section 9.3.3.

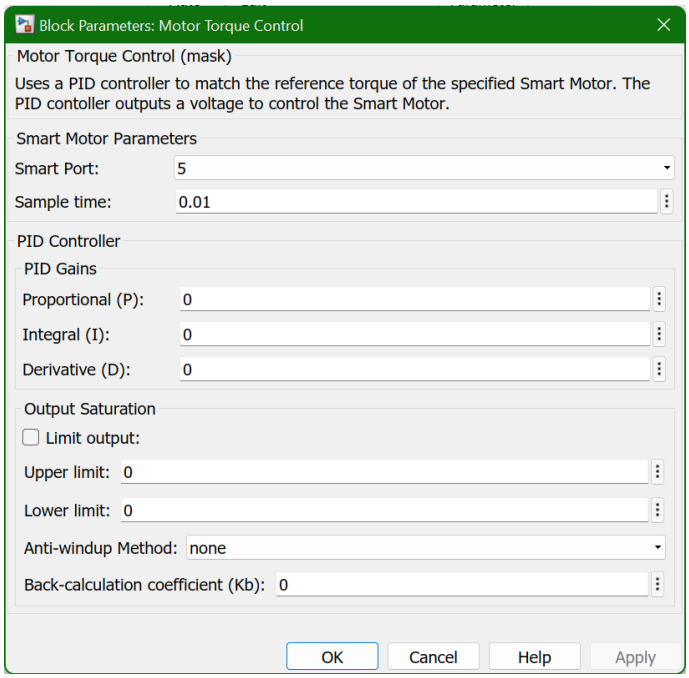


Figure 56: 'Motor Torque Control' block mask

The internal model is constructed as follows:

(A) The current Smart Motor torque (Nm) is read.

(B) The sensor is noisy and so the torque data is passed through a low pass filter.

(C) The filtered torque is used as the measured output for a PID controller; the reference is an input to the block.

(D) The PID output is used as a voltage for the Smart Motor Controller.

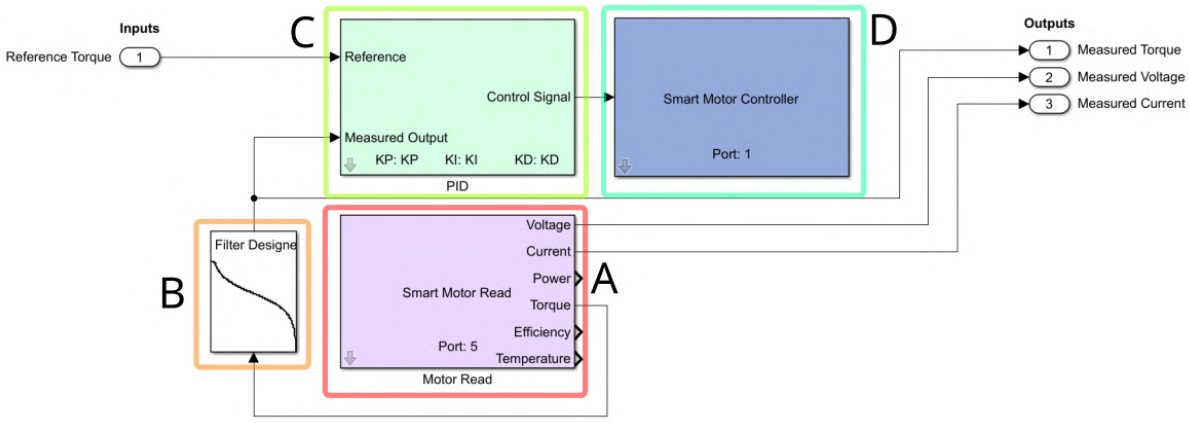


Figure 57: 'Motor Torque Control' internal model

10 Using Monitor & Tune

Simulink provides functionality to log data and change variables while the program is running (MATLAB Documentation). Rather than use the 'Build, Deploy & Start' option as in Section 9.1, instead click 'Monitor & Tune' setting the 'Simulation Time' to inf (see Figure 15b). This then runs the model on the Vex Brain but keeps a serial communication open with which to read data and update variables.

By left-clicking on any signal (see Figure 58), an option shows to 'Log Selected Signal' which will allow it to be read inside the 'Data Inspector'. Left-clicking the logging symbol and clicking the 'Properties' option allows the signal name to be changed to easily identify it within the 'Data Inspector' window.

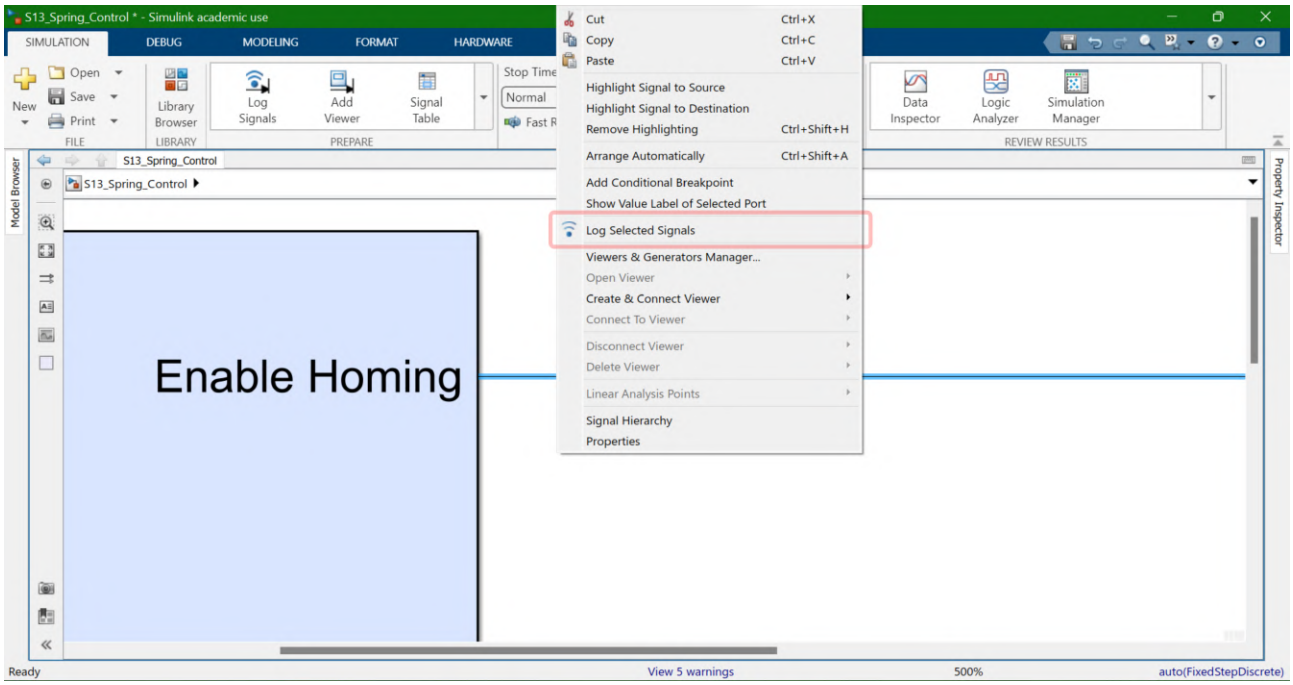


Figure 58: Signal logging popup

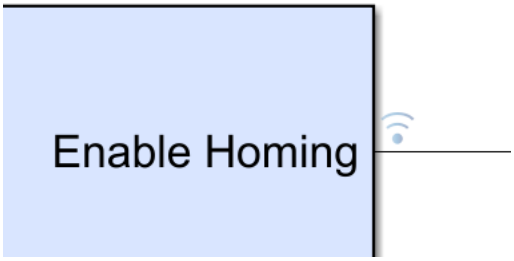


Figure 59: A symbol will show on any signal marked for data logging

While 'Monitor & Tune' is active, block masks can also be edited. This is convenienct

when tuning PID controllers as the parameters can be changed while smultaneously viewing performance in the 'Data Inspector' window.

An example of logging data is shown below in Figure 60. The left hand side contains a list of all currently logged signals which can be displayed by clicking the tick boxes; these are then visible in the graph on the right hand side.

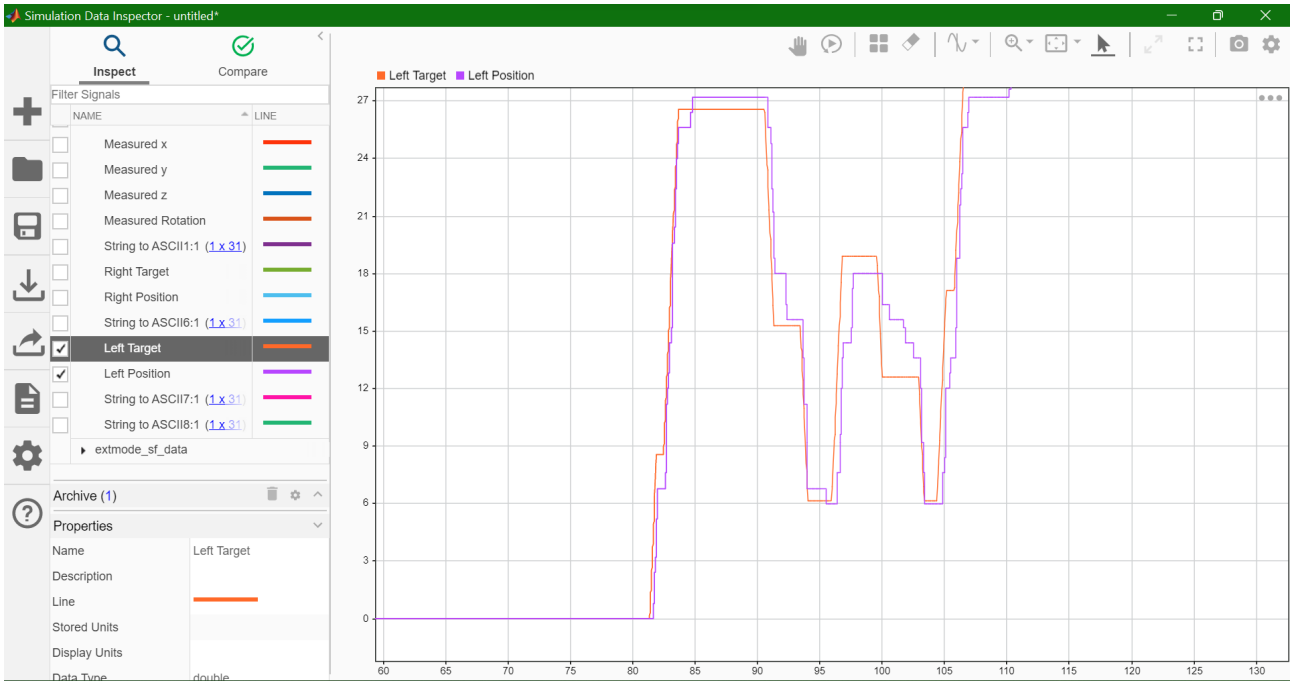


Figure 60: Data Inspector window with two signals logged: 'Left Target' and 'Left Position'

11 Appendix

Angle to Potentiometer

```
function [targetA,targetB,targetC,targetD] = deg2Analog(a1,a2,a3,a4,h1,h2,h3,h4)

    % Vex Brain has a 10 bit ADC == 4096 resolution
    % Potentiometer has 333 degrees of rotation
    scale = ((4096 / 333)*360) / (2*pi);

    % Home angles
    conv = 2*pi / 360;
    ang1 = 0;
    ang2 = 20*conv;
    ang3 = 3.07*conv;
    ang4 = 93.07*conv;

    % Subtract home angles from desired position
    t1 = a1 - ang1;
    t2 = a2 - ang2;
    t3 = a3 - ang3;
    t4 = a4 - ang4;

    % Scale angles, including gear ratios
    targetA = h1 + t1*scale;
    targetB = h2 - t2*scale*3;
    targetC = h3 - t3*scale*3;
    targetD = h4 - t4*scale*3;
```

Angle to Potentiometer

```
function [a1,a2,a3,a4] = analog2Deg(d1,d2,d3,d4,m1,m2,m3,m4)

    % Vex Brain has a 10 bit ADC == 4096 resolution
    % Potentiometer has 333 degrees of rotation
    scale = ((4096 / 333)*360) / (2*pi);

    % Home angles
    conv = 2*pi / 360;
    ang1 = 0;
    ang2 = 20*conv;
    ang3 = 3.07*conv;
    ang4 = 93.07*conv;

    % Find angles relative to home position
    t1 = (d1 - m1) / scale;
    t2 = (m2 - d2) / (3*scale);
    t3 = (m3 - d3) / (3*scale);
    t4 = (m4 - d4) / (3*scale);

    % Compute actual motor angles
    a1 = t1 + ang1;
    a2 = t2 + ang2;
    a3 = t3 + ang3;
```

```
24          a4 = t4 + ang4;
```

Joint to Motor Angle

```
1   function [motorA,motorB,motorC,motorD] = jointToMotor(jointA,jointB,jointC,jointD)
2
3       % Joints A and B are the same in both reference frames
4       motorA = jointA;
5       motorB = jointB;
6       % Joints C and D have angles dependant on previous members
7       motorC = pi - (jointB + jointC);
8       motorD = 2*pi - jointD - (pi - (jointC + jointB));
```

Motor to Joint Angle

```
1   function [jointA,jointB,jointC,jointD] = motorToArm(motorA,motorB,motorC,motorD)
2
3       % Joints A and B are the same in both reference frames
4       jointA = motorA;
5       jointB = motorB;
6       % Joints C and D have angles dependant on previous members
7       jointC = pi - (motorB + motorC);
8       jointD = pi - (motorC - motorD);
```

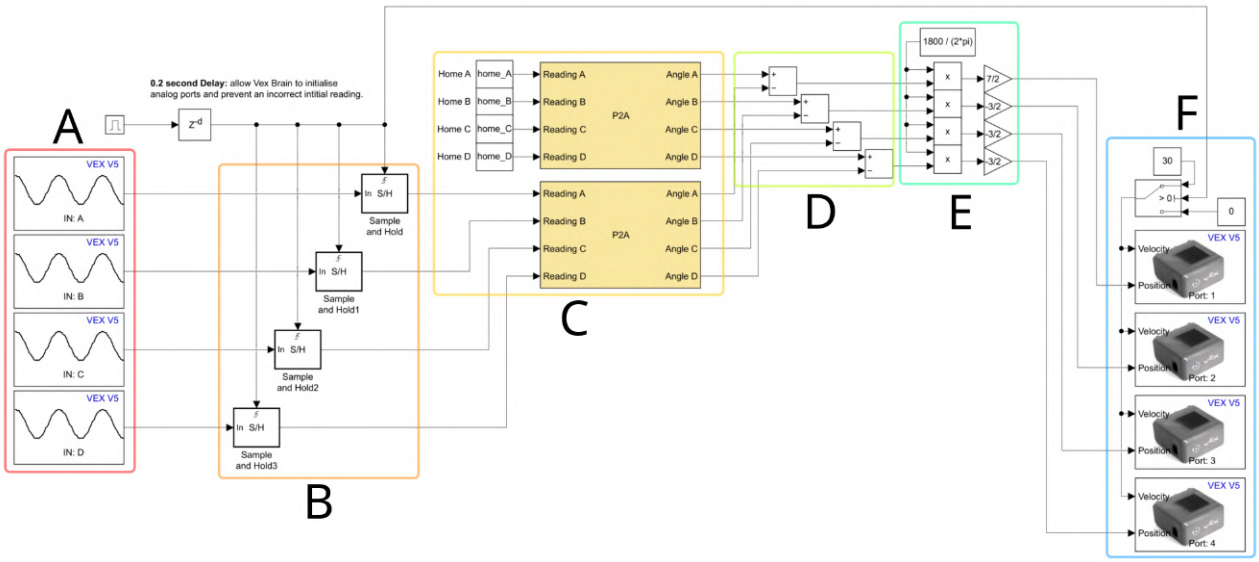
Forward Kinematic Calculation Code

```
1   function [x,y,z] = forward_kinematics(a1, a2, a3, a4)
2
3       % Define geometric constants
4       conv = 25.4;        % Convert from inches to mm
5       AR = 0.7224*conv;
6       AZ = (5.296+1.375)*conv;
7       BC = 7*conv;
8       CD = 7*conv;
9       DE = 2*conv;
10
11      % Implement derived forward dynamics equations
12      x = (AR + BC*sin(a2) + CD*sin(a3) + DE*sin(a4))*cos(a1);
13      y = (AR + BC*sin(a2) + CD*sin(a3) + DE*sin(a4))*sin(a1);
14      z = (AZ + BC*cos(a2) - CD*cos(a3) + DE*cos(a4));
```

Inverse Kinematic Calculation Code

```
1   function [a1,a2,a3,a4]= inverseKinematics(x,y,z,rotation)
2
3       % Geometric constants
4       conv = 25.4;
5       L_1 = 7*conv;
6       L_2 = 7*conv;
7       L_3 = 2*conv;
8       AR = 0.7224*conv;
9       AZ = (5.296+1.375)*conv;
10
11      % Handle end effector position/rotation
12      x_adj = x - L_3*sin(rotation)*cos(atan(y/x));
13      y_adj = y - L_3*sin(rotation)*sin(atan(y/x));
14      z_adj = z - L_3*cos(rotation);
```

```
15          t4 = rotation;
16
17          % Fix base angle to reduce unknowns
18          t1 = atan(y_adj/x_adj);
19          XE = sqrt(x_adj^2 + y_adj^2) - AR;
20          YE = z_adj - AZ;
21
22          % Derived using MATLAB Symbolic Maths
23          t2 = 2*atan((2*L_1*XE - (- L_1^4 + 2*L_1^2*L_2^2 + 2*L_1^2*XE^2 + ...
24          2*L_1^2*YE^2 - L_2^4 + 2*L_2^2*XE^2 + 2*L_2^2*YE^2 - XE^4 - ...
25          2*XE^2*YE^2 - YE^4)^(1/2))/(L_1^2 + 2*L_1*YE - L_2^2 + XE^2 + YE^2));
26
27          % As above
28          t3 = 2*atan((2*L_2*XE - ((- L_1^2 + 2*L_1*L_2 - L_2^2 + XE^2 + YE^2) ...
29          *(L_1^2 + 2*L_1*L_2 + L_2^2 - XE^2 - YE^2))^(1/2))/ ...
30          (- L_1^2 + L_2^2 - 2*L_2*YE + XE^2 + YE^2));
31
32          % Above formulas can produce complex results, the following fixes that
33          a1 = NaN(size(t1));
34          a2 = NaN(size(t2));
35          a3 = NaN(size(t3));
36          a4 = NaN(size(t4));
37
38          tf_mat = imag(t1) == 0;
39          a1(tf_mat) = real(t1(tf_mat));
40
41          tf_mat = imag(t2) == 0;
42          a2(tf_mat) = real(t2(tf_mat));
43
44          tf_mat = imag(t3) == 0;
45          a3(tf_mat) = real(t3(tf_mat));
46
47          tf_mat = imag(t4) == 0;
48          a4(tf_mat) = real(t4(tf_mat));
```


A
B
C
D
E
F
0.2 second Delay: allow Vex Brain to initialise analog ports and prevent an incorrect intitial reading.
z^-d
VEX V5
IN: A
IN: B
IN: C
IN: D
In S/H
Sample and Hold
Sample and Hold1
Sample and Hold2
Sample and Hold3
Home A
Home B
Home C
Home D
home_A
home_B
home_C
home_D
Reading A
Reading B
Reading C
Reading D
Angle A
Angle B
Angle C
Angle D
P2A
1800 / (2*pi)
x
7/2
3/2
30
> 0
0
Velocity
Position
Port: 1
Port: 2
Port: 3
Port: 4