Playing with Fusion

BDC-10001 CIM Motor/Controller – FRC User's Manual

Introduction

Venom combines a highly efficient motor controller, speed sensor, and CIM motor into a single, convenient package. It may be controlled through a servo style PWM interface (1 to 2ms pulse) or through a 1M bit CAN (Controller Area Network) interface.

CAN communication is abstracted through the PlayingWithFusionDriver library. This library provides FRC compatible C++, Java, and LabView APIs, as well as a C API which is intended for teams using unofficially supported languages such as Python.

Glossary

CAN	Controller Area Network
LSA	Optional Limit Switch and Analog breakout board for Venom. The LSA allows limit switches and an auxiliary analog input to be connected and read over CAN.
PWM	Pulse Width Modulation.
PWF	Playing With Fusion

Electrical connections

Venom has four electrical connections (not including the optional LSA breakout board). Power and ground (Red and Black) should be connected to the battery through the FRC power distribution board and protected by a 40 Amp fuse.

Warning! Do not reverse the polarity of the power connections. This will cause permanent damage to Venom

The green and yellow wires are dual purpose. The may be used for CAN communication or for basic PWM control. No configuration is necessary. Venom will automatically detect if it is driven by a PWM signal or is connected to a CAN bus.

When used for CAN, connect the green wire to CAN low on the roboRIO and yellow to CAN high. In general, CAN devices should be connected in a 'daisy chain'. The CAN bus should route directly between two devices, then between the next two, and so on. It is acceptable to 'T' into a CAN bus as long as the length of the 'T' (also known as the drop) is less than 11 inches. This length is based on the 1Mbit bitrate used by FRC.

Example of a CAN Bus using short drops to tie left and right motor pairs onto the main CAN trunk:

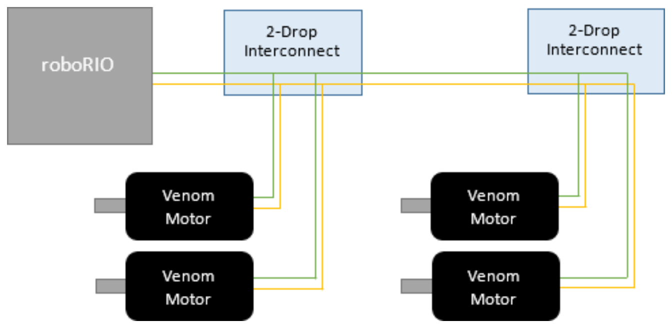


Figure 1 - CAN bus with multiple short drops

The bus length between the roboRIO and the first 2-drop interconnect and the length between the two 2-drop interconnects can be relatively long. If there are additional CAN devices on the bus they should be connected to the right of the second 2-drop interconnect. The length of that bus can also be relatively long. These sections of the bus follow the recommended daisy chain design.

The length of the drops between the 2-Drop interconnects and the Venom motors must be less than 11 inches to avoid CAN communication issues. Note that Venom motor's CAN pigtail is 6-7 inches long.

To use Venom in PWM mode, simply connect the green and yellow wires together and connect them to a PWM output of the roboRIO or another device. Like a hobby servo, the PWM signal should have a frequency around 50Hz, and the positive pulse width should vary between 1.0ms and 2.0ms. A 1.5ms pulse time results in a zero/neutral motor command. A 1.0ms and 2.0ms pulse time result in maximum motor speed in the forward or reverse direction.

Importing the FRC compatible software libraries

Online C++ and Java driver installation

The Venom control library (PlayingWithFusionDriver) can be imported into a robot project directly from the Visual Studio Code application. The following steps will download the latest version of the driver from the Playing With Fusion website.

- Click on the WPILib Command Pallet icon on the top right corner of the VS Code window.
- Select WPILib: Manage Vendor Libraries
- Then select Install new library (online) end enter the following URL:
    - https://www.playingwithfusion.com/frc/playingwithfusion2020.json
    - Or for the 2019 rio image: https://www.playingwithfusion.com/frc/playingwithfusion2019.json
- Press enter

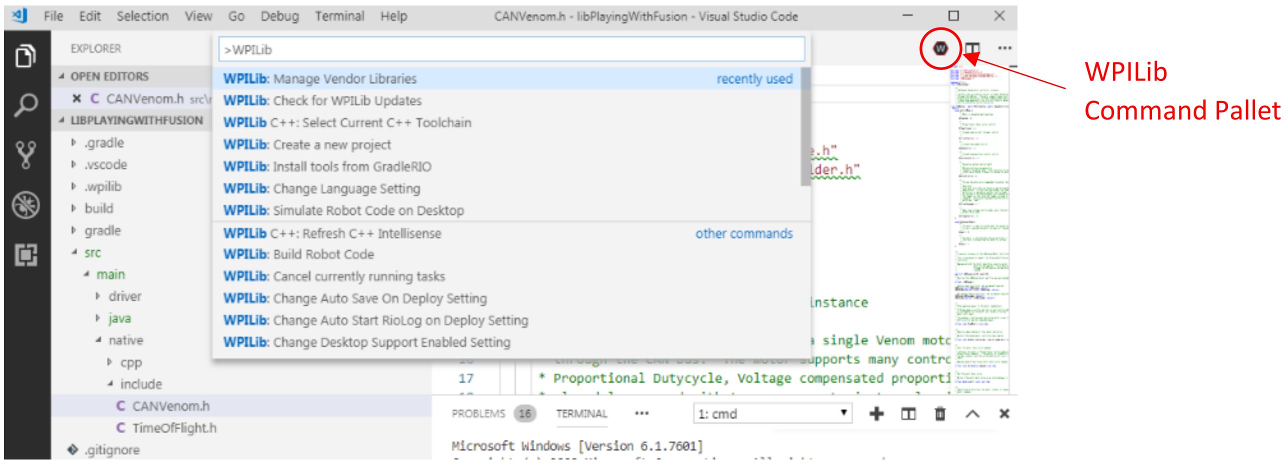

>WPILib
WPILib: Manage Vendor Libraries
recently used
WPILib: Check for WPILib Updates
WPILib C++: Select Current C++ Toolchain
WPILib: Create a new project
WPILib: Install tools from GradleRIO
WPILib: Change Language Setting
WPILib: Simulate Robot Code on Desktop
WPILib C++: Refresh C++ Intellisense
other commands
WPILib: Build Robot Code
WPILib: Cancel currently running tasks
WPILib: Change Auto Save On Deploy Setting
WPILib: Change Auto Start RioLog on Deploy Setting
WPILib: Change Desktop Support Enabled Setting
WPILib
Command Pallet

Figure 2 - WPILib Command Pallet in VS Code

If everything works, a folder named vendordeps will be created in the root directory of the project and will contain a file named playingwithfusion2020.json.

Offline C++ and Java driver installation

The Venom library may also be installed without an internet connection.   First, download PlayingWithFusionLibrary2020.zip from the Venom product page on the Playing With Fusion website.
https://www.playingwithfusion.com/productview.php?pdid=99&catid=1014

Once the driver files are available, preform the following steps:

- Unzip the PlayingWithFusionDriver zip file to C:\Users\Public\wpilib\2020\ under Windows or ~/wpilib/2020/ under Mac/Linux.
  - (Unzip to C:\Users\Public\frc2019\ or ~/frc2019/ if using the 2019 toolchain)
  - If asked to merge, select yes.
  - This zip file includes the driver libraries (so files), C++ header files, Java source as well as the JSON file which is used by the FRC build system to identify the Playing With Fusion driver.
- Create a robot project using Visual Studio Code (if one does not already exist)
- Click on the WPILib Command Pallet icon on the top right corner of the VS Code window.
- Select WPILib: Manage Vendor Libraries
- Then select Install new library (offline)
- Select PlayingWithFusion

If everything works a folder named vendordeps will be created in the root directory of the project and will contain a file named playingwithfusion.json.

LabVIEW Driver Installation

The Venom LabVIEW library may be installed without an internet connection. First, download libPlayingWithFusionDriver.so and playing_with_fusion_lib_for_frc-*.vip from the Venom product page on the Playing With Fusion website.
https://www.playingwithfusion.com/productview.php?pdid=99&catid=1014

Once the driver files are available, preform the following steps:

- Copy libPlayingWithFusionDriver.so into the /usr/local/frc/third-party/lib folder on the roboRIO
- Double click on the *.vip file. This will open the LabVIEW Package Manager
- Click the Install button

CAN Control Modes

Follow the Leader

This mode was created to support two or more motors that are mechanically coupled together. In these cases, one Venom is chosen as the leader. This is the motor which executes a motion profile or is placed in a closed-loop control mode (like speed control or position control). The other motors may then be placed in Follow the Leader mode. This will cause them to mirror the instantaneous output of the lead motor.

Follow the Leader mode is important because if multiple motors are coupled together and all are placed in a closed-loop mode, the PIDs is each motor may 'fight' each other and cause instabilities. When follow the leader is used, only one PID is active and all motors work as a team.

To enter Follow the Leader mode use the SetCommand API function on the lead motor as normal. Then use the SetCommand() function on the follower motors using CANVenom::ControlMode:: kFollowTheLeader (ControlMode.FollowTheLeader in Java) as the mode and the motor ID of the leader as the command. For example if the lead motor ID is 0, the following command (C++) will instruct the follower motor to mirror the leaders command:

```
leadMotor.SetCommand( ControlMode::kSpeedControl, 5000 );
followerMotor.SetCommand( ControlMode::kFollowTheLeader, 0 );
```

An alternate method to enter Follow the Leader Mode is to call the Follow() API function and pass a reference to the lead motor directly:

```
leadMotor.SetCommand( ControlMode::kSpeedControl, 5000 );
followerMotor.Follow( leadMotor );
```

Proportional (Coast & Braking)

In proportional mode, the SetCommand() API function commands the raw, open-loop, motor controller duty cycle (as a value between -1.0 and 1.0). This mode does not compensate for battery voltage, so the speed of an unloaded motor will be proportional to the command and the battery voltage.

$$DC = CommandedDutyCycle$$

Voltage Control

Similar to proportional mode, voltage control is also an open-loop control mode. In this mode the motor command is compensated for battery voltage.   If the motor is unloaded and the commanded voltage is 8 Volts, the motor speed will remain constant as the battery voltage varies, as long as the batter voltage is greater than the command (8 Volts in this example).

The voltage in voltage control mode refers to the voltage at the motor brushes.

In voltage control mode, the motor controller duty cycle is calculated using the following equation:

$$DC = \frac{CommandedVoltage}{MeasuredBatteryVoltage}$$

Speed Control

Speed Control is a closed-loop control mode. When a speed (in RPM) is specified using the SetCommand() function, a PID controller in Venom actively adjusts the motor controller output to achieve that speed.

If desired, Venom will slew rate limit the speed command to enforce motor acceleration and jerk limits. This effectively allows Venom to calculate true S-curves live without the need to use a motion profile.

If the jerk limit is set to zero, only acceleration is limited (equivalent to a trapezoidal curve). If both jerk and acceleration are zero the commanded speed is not slew rate limited.

An optional error deadband may be applied to the integral and derivative terms in the PID controller. With care, this deadband can be used to help prevent the PID from 'hunting' when the actual motor speed is near the commanded speed.

In Speed Control mode, the motor controller duty cycle is calculated using the following equations:

$$TargetSpeed = SCurve(CommandedSpeed, MaxV, MaxA, MaxJ)$$

$$RawError = TargetSpeed - MeasuredSpeed$$

$$Error = \begin{cases} RawError + Deadband & if\ RawError < -Deadband \\ 0 & if\ |RawError| \le Deadband \\ RawError - Deadband & if\ RawError > Deadband \end{cases}$$

$$DC = \frac{\left(k_p * RawError\right) + \left(k_i * \int Error\right) + \left(k_d * \frac{dError}{dt}\right) + \left(k_f * TargetSpeed\right) + b}{1024}$$

The following table lists PID gains and acceleration/jerk limits which may be used as a starting point when tuning Venom. These values were selected using an unloaded motor.

Constant	**Real World value**
Kf	0.184
B	0
Kp	0.195
Ki	0.010
Kd	0
Max V	5500 RPM
Max A	20,000 RPM/sec
Max J	31,250 RPM/sec^2
Deadband	0

Torque Control

Torque Control is a closed-loop control mode. In a brushed motor, motor torque is proportional to motor current. When a current (in Amps) is specified using the SetCommand() function, a PID controller in Venom actively adjusts the motor controller output to achieve that electrical current.

An optional error deadband may be applied to the integral and derivative terms in the PID controller. With care, this deadband can be used to help prevent the PID from 'hunting' when the actual motor current is near the commanded current.

Speed, acceleration and jerk limits are not applied in Torque Control mode.

In Torque Control mode, the motor controller duty cycle is calculated using the following equations:

$$RawError = CommandedCurrent - MeasuredCurrent$$

$$Error = \begin{cases} RawError + Deadband & if\ RawError < -Deadband \\ 0 & if\ |RawError| \leq Deadband \\ RawError - Deadband & if\ RawError > Deadband \end{cases}$$

$$DC = \frac{\left(k_p * RawError\right) + \left(k_i * \int Error\right) + \left(k_d * \frac{dError}{dt}\right) + \left(k_f * Command\right) + b}{1024}$$

The following table list PID gains which may be used as a starting point when tuning Venom. These values were selected using an unloaded motor.

Constant	**Real World value**
Kf	0
B	0
Kp	0.195
Ki	0.010
Kd	0
Max V	N/A
Max A	N/A
Max J	N/A
Deadband	0

Position Control

Position Control is a closed-loop control mode. When a position (in motor revolutions) is specified using the SetCommand() function, a PID controller in Venom actively adjusts the motor controller output to achieve that position.

If desired, Venom will slew rate limit the position command to enforce motor speed and acceleration limits. This effectively allows Venom to calculate trapezoidal motion paths live without the need to use a motion profile. The jerk limit is not used in position control mode.

If the acceleration limit is set to zero, only the maximum speed is limited. If both speed and acceleration limits are zero the commanded position is not slew rate limited.

An optional error deadband may be applied to the integral and derivative terms in the PID controller. With care, this deadband can be used to help prevent the PID from 'hunting' when the actual motor position is near the commanded position.

In Position Control mode, the motor controller duty cycle (-1.0 to 1.0) is calculated using the following equations:

$$(TargetPosition, TargetSpeed) = TrapezoidCurve(CommandedPosition, MaxV, MaxA)$$

$$RawError = TargetPosition - MeasuredPosition$$

$$Error = \begin{cases} RawError + Deadband & if\ RawError < -Deadband \\ 0 & if\ |RawError| \leq Deadband \\ RawError - Deadband & if\ RawError > Deadband \end{cases}$$

$$DC = \frac{(k_p * RawError) + (k_i * \int Error) + \left(k_d * \frac{dError}{dt}\right) + \left(k_f * TargetSpeed\right) + b}{1024}$$

The following table list PID gains which may be used as a starting point when tuning Venom. These values were selected using an unloaded motor.

Constant	Real World value
Kf	0.184
B	0
Kp	1.5
Ki	0
Kd	0
Max V	5500 RPM
Max A	20,000 RPM/sec
Max J	N/A
Deadband	0

Motion Profile

Motion profiles allow Venom to follow pre-calculated paths without tying up CPU resources on the roboRIO. The paths are made up of points which specify the motor speed and position (in rotations) at specific times. Up to 300 points may be loaded into Venom at a time.

Once the motion profile is executed, Venom will interpolate between each point, adjusting its output 100 times a second to achieve the instantaneous target. Additional points may be added as Venom executes a path which allows Venom to execute a motion path indefinitely. Once Venom reaches the end of a path or it runs out of points to execute, Venom will hold the last commanded position until a new motion profile is initiated.

While executing a motion profile, the motor controller duty cycle (-1.0 to 1.0) is calculated using the following equations:

$$(TargetPosition, TargetSpeed) = CurrentProfilePoint(time)$$

$$RawError = TargetPosition - MeasuredPosition$$

$$Error = \begin{cases} RawError + Deadband & if\ RawError < -Deadband \\ 0 & if\ |RawError| \leq Deadband \\ RawError - Deadband & if\ RawError > Deadband \end{cases}$$

$$DC = \frac{\left(k_p * RawError\right) + \left(k_i * \int Error\right) + \left(k_d * \frac{dError}{dt}\right) + \left(k_f * TargetSpeed\right) + b}{1024}$$

The following table list PID gains which may be used as a starting point when tuning Venom. These values were selected using an unloaded motor.

Constant	Real World value
Kf	0.184
B	0
Kp	1.5
Ki	0
Kd	0
Max V	5500 RPM
Max A	20,000 RPM/sec
Max J	N/A
Deadband	0

In order to execute a motion profile:

1. Place Venom in 'Disabled' mode, or any control mode other than kMotionProfile
   a. For example, SetCommand(ControlMode::kDisabled, 0)
2. Call the ResetPosition() API to reset the motor position to 0 revolutions
3. Call the ClearMotionProfilePoints() API to erase all stored motion profile points from Venom
4. Load up to 300 points into Venom using the AddMotionProfilePoint() API.

   a. Call the GetNumAvaliableMotionProfilePoints() API at any time to determine the number of addition motion profile points that may be sent to Venom. Note, that the available point buffer count is only updated every 10 ms, so this function may return an incorrect count if points were sent within the last 10 ms.
   b. Best practice is to call GetNumAvaliableMotionProfilePoints() every 20ms (or each iteration of a periodic task) to determine the number of free points in Venom. If the result is greater than 10 points, send 10 more points using AddMotionProfilePoint() then wait for the next iteration of the periodic function.
5. Use the CompleteMotionProfilePath() to specify the final point in a motion profile path and instruct Venom to hold the final commanded position.
6. Call the ExecutePath() API to begin following the motion profile.

PID Tuning Tips

Kp, Ki, and Kd are traditional PID gains. Kp is the proportion gain and is generally has the largest values. It attempts to correct motor output based on the instantaneous error.

Kf and B work together to form the feed forward term. This is sometimes referred to as the open loop command. These terms do not rely on any measured motor state and are based solely on the command specified by the SetCommand() function. Kf is especially helpful when in speed control mode because the speed of a brushed DC motor is proportional to the voltage at the motor brushes. The voltage at the motor brushes is simply the motor controller duty cycle multiplied by the motor supply (the battery) voltage. Venom rotates at least 5400 RPM at full command with a 12V supply voltage. Given the control mode equations from earlier sections, Kf may be calculated to achieve 5400 RPM with a maximum motor duty cycle (DC = 1.0):

$$DC = \frac{k_f * TargetSpeed}{1024}$$

Solving for Kf:

$$k_f = \frac{DC * 1024}{TargetSpeed} = \frac{1.0 * 1024}{5400} = 0.190$$

The calculated value of 0.190 is very near the value of 1.84 specified for Kf in the gain tables specified for each control mode.

The B term applies a fixed offset to the motor command. Generally this term should be zero, but there are cases where it may be helpful. For example if Venom is used as part of a lift or hoist, the motor command needed to balance the force of gravity may be specified by B.

When tuning any PID for the first time consider:

- Starting with the gains listed in this document for the desired control mode
- Keep Kd, B, and Deadband equal to zero
- Start with a Ki value around 10 times smaller than Kp.

Thermal and Over Current Protection

Venom employs thermal and overcurrent protection to protect the motor as well as the embedded controller. When the internal temperature is greater than 90°C or if the average motor current is greater than 40 Amps, Venom will begin to derate by internally limiting the maximum controller duty cycle.

The derates are proportional to how far the temperature or current limits are exceeded. In other words, Venom will not suddenly stop or coast if it becomes too hot. Instead, it will gradually reduce its output to prevent it from heating further.

Venom may be stalled indefinitely under full load without risk of damage. The over temperature and current limits will work to keep the motor temperature below 100°C. Although the limits cannot be disabled, they are rarely encountered.

Error Codes and Diagnostics

If Venom detects one or more fault conditions (over temperate, incorrectly formatted CAN message, etc.) it will report an error using the GetActiveFaults() API as well as flashing the rear LED in a distinctive pattern. Both GetActiveFaults and the LED report active fault conditions which zero or limit motor output at the current instant in time.

To diagnose brief or intermittent faults the GetLatchedFaults() and ClearLatchedFaults() APIs are provided as well. GetLatchedFaults() remembers any fault which becomes active after the last time ClearLatchedFaults() is called. It is especially helpful to diagnose brownout or harness problems which cause Venom to reboot.

In general, a robot should call ClearLatchedFaults() near the start of its program, then periodically call GetLatchedFaults(). If the VenomReset fault is ever set, it is likely that the motor lost power at some point since the start of the robot program.

LED Pattern	Description
🟥🟥🟥🟥	Venom is initializing. This state should last less than 40ms after power up.
🟥🟥🟩🟩	The 'Identify Device' feature is active. This pattern in used to locate a particular Playing With Fusion device when multiple are installed on a robot. See the Motor Configuration section for more information
🟢 🟩🟩🟩	Venom is initialized and in PWM mode. Waiting for a valid 1.0 to 2.0 ms PWM pulse.
🟩🟩🟩🟩	Venom successfully entered a valid CAN or PWM control mode. No Faults are active and motion may be commanded
🟢 🟢 🟩🟩	Venom is initialized and detected a valid CAN bus
🟥🟩🟩🟩	CAN communication fault. Check harness connections and but termination
🟥🟥🟩🟩	Missing heartbeat in CAN control mode. Ensure device ID matches device ID used by CANVenom class. See the Motor Configuration section for more information and instructions to change/verify the device ID
🟥🟩🟥🟩	Lead motor heartbeat is missing while in Follow The Leader mode.

🟥🟥🟥🟩	The lead motor ID is same as the motor ID. One Venom cannot follow itself. Ensure the leader and follower have different IDs
🟥🟩🟩🟥	An invalid control mode was specified by the roboRIO. This should not occur when using PlayingWithFusionDriver. Contact PWF Technical support.
🟥🟥🟩🟥	Another Venom with the same device ID was detected on the CAN bus. All Venom device IDs must be unique
🟥🟩🟥🟥	The forward limit switch is enabled and is active
🟥🟥🟥🟥	The reverse limit switch is enabled and is active
🔴 🟩🟩🟩🟩🟩🟩	Motor temperature is too high
🔴 🔴 🟩🟩🟩🟩🟩	Average motor current is too high

Limit Switches

Optional forward and reverse switches may be used to prevent Venom from driving through mechanical end stops or other conditions which may otherwise damage a robot. The limit switches are connected using the optional Limit Switch and Analog (LSA) breakout board which connects to the back of Venom.

Limit switches are disabled at power up. They must be enabled using the EnableLimitSwitches() API. The forward and reverse limits may be enabled/disabled independently. None, one, or both may be enabled.

Once enabled, the limit switch inputs must be connected to GND to allow motion. The forward limit switch must be grounded to allow Venom to spin in the forward direction and the reverse switch must be grounded to allow motion in the reverse direction. Internal pullup resistors on each of the limit switch inputs will cause the switches to become active (inhibit motion) if they become disconnected.

Auxiliary Digital and Analog Inputs

If they are not used as limit switches, the two limit switch inputs on the LSA breakout board may be used as general purpose digital inputs. The state of each input may be read through the GetFwdLimitSwitchActive() and GetRevLimitSwitchActive() APIs. Each function will return false when the corresponding digital input is grounded and true if the voltage is greater than 3 Volts. Do not exceed 5.0 Volts on either digital input.

A single auxiliary analog input is also available through the LSA breakout board. This input is not used by Venom directly. It was intended to simplify robot wiring by allowing a sensor placed near a motor to be connected directly to Venom rather than routing the harness to the roboRIO or another controller. The analog input is capable of measuring 0 - 5.0 Volts and may be read using the GetAuxVoltage() API.

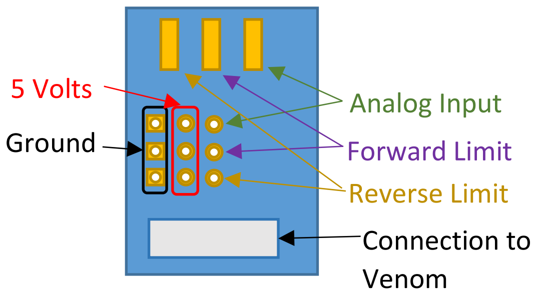


Figure 3 - LSA electrical connections

The LSA breakout connects to the rear of Venom through the accessory connector. This connector is normally hidden by a removable cover. To remove the cover and access the accessory connector, first remove the two small black Phillips screws retaining the rear plastic cover. Once the cover is off, gently remove the small square accessory cover. The cover should fall out with very little force. Finally, re-install the rear cover and the Philips screws.

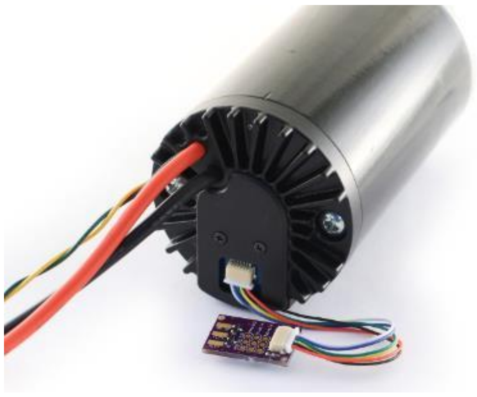


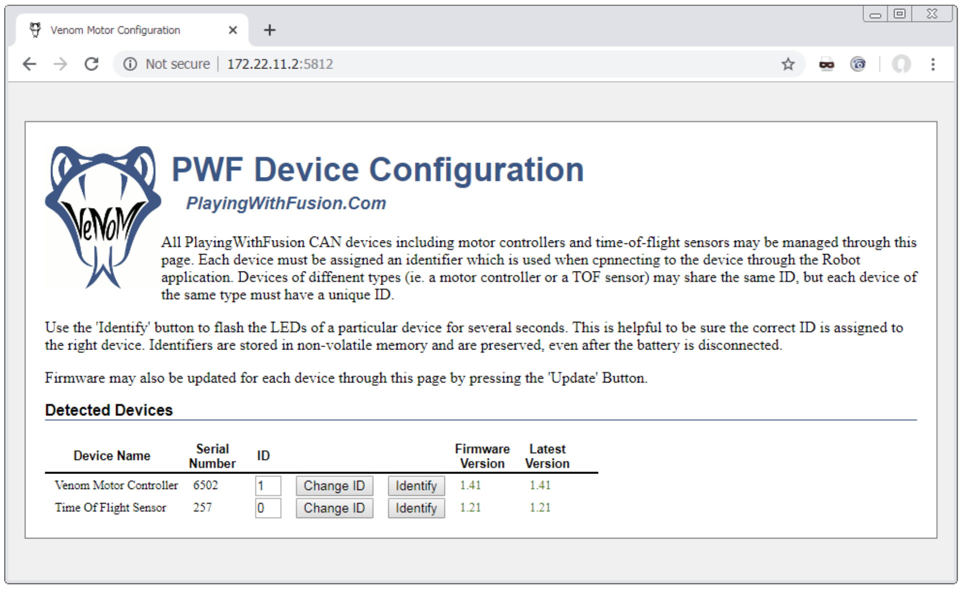
Figure 4 - Back of Venom with accessory cover removed and LSA connected

Motor Configuration

The PlayingWithFusionDriver library includes a web interface to manage Playing with Fusion devices

Each device must be assigned an ID that is unique for all other devices of the same type.   For example, only one motor controller may use ID 1.   It is perfectly acceptable to also configure a time of flight sensor as ID 1 because it is a different device type than the Venom motor.

The configuration interfaces may be accessed by typing in the IP address of the roboRIO into a web browser followed by :5812.



PWF Device Configuration

PlayingWithFusion.Com

All PlayingWithFusion CAN devices including motor controllers and time-of-flight sensors may be managed through this page. Each device must be assigned an identifier which is used when cpnnecting to the device through the Robot application. Devices of diffenent types (ie. a motor controller or a TOF sensor) may share the same ID, but each device of the same type must have a unique ID.

Use the 'Identify' button to flash the LEDs of a particular device for several seconds. This is helpful to be sure the correct ID is assigned to the right device. Identifiers are stored in non-volatile memory and are preserved, even after the battery is disconnected.

Firmware may also be updated for each device through this page by pressing the 'Update' Button.

Detected Devices

Device Name	Serial Number	ID			Firmware Version	Latest Version
Venom Motor Controller	6502	1	Change ID	Identify	1.41	1.41
Time Of Flight Sensor	257	0	Change ID	Identify	1.21	1.21

Figure 5 - PWF Device Configuration Page

Each detected Playing with Fusion device will be listed on the configuration page.   If a device is missing, check power to the device and verify the CAN bus connections.

Device IDs may be changed by entering the desired device ID into text box on the corresponding row then pressing the Change ID button. The change will take effect immediately and will be stored across power cycles.

The Identify button is a useful tool to determine which physical device is associated with each ID. The LED on a Venom motor will blink red and green for a short period when Identify is pressed.

If new firmware is available for any device, an Update button will appear in the right most column for that device. Firmware is included as part of the PlayingWithFusionDriver and cannot be downloaded separately from the Playing with Fusion website.

To update device firmware:

- Disable the robot using the driver station
- Ensure the robot battery is fully charged
- Press the Update button.

- Do not initiate another firmware update or remove power from the roboRIO until the update process is complete.
- A message will appear above the device list which will display whether or not the reprogramming attempt was successful.
- If you have difficulties reprogramming contact Playing With Fusion technical support at TechnicalSupport@PlayingWithFusion.com