Getting Started

To use the provided packages in your Bonsai environment, you will need to:

1. Install Bonsai
2. Go to `Package Manager`
3. Select NuGet.org as the `Package source`
4. Look for `AllenNeuralDynamics` for a list of all available packages written and maintained by AIND
5. Install the desired packages. Any dependencies will be automatically installed as well.
6. Some packages are still experimental, so you might need to tick `Include prerelease` to see them.

Version Control

A repository is represented in Bonsai by the `CreateRepository` operator. Several properties are exposed via this object, including the name and hash of the current commit.

In order to make it easier for users to evaluate the state of the local repository, an additional operator, `IsRepositoryClean`, is provided to check if the repository is in a clean state (i.e. are there any untracked or uncommitted changes).

Core - Logging

Examples

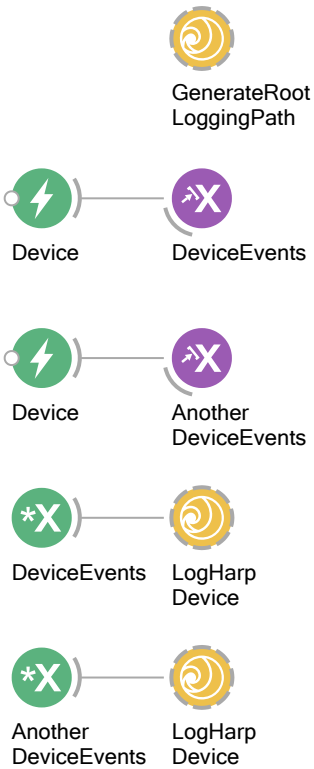
A simple logging pipeline can be assembled using the Core package.

The first thing that should be defined is the root where all data will be saved to. This can be done using the `GenerateRootLoggingPath`. This Node will instantiate a `Subject` with a unique path in the form of `<RootFolder>/<Subject>/<Date>`.

Harp data

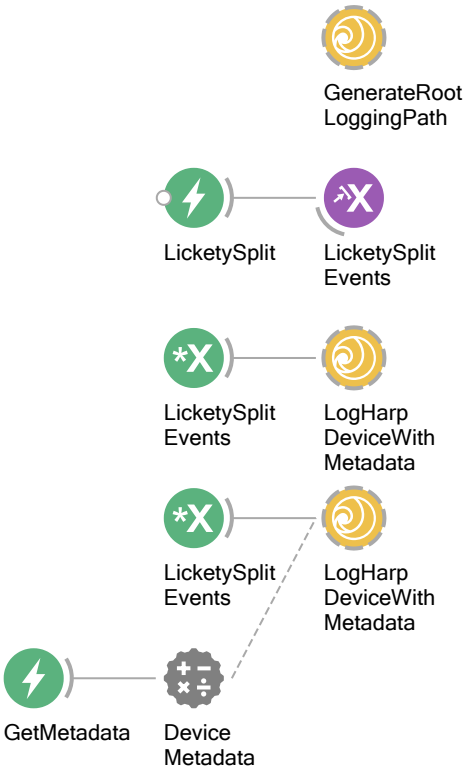
Once this `Subject` is created, other nodes can access to it. For instance, if one would like to save the data from a `Harp Device`:

Without metadata



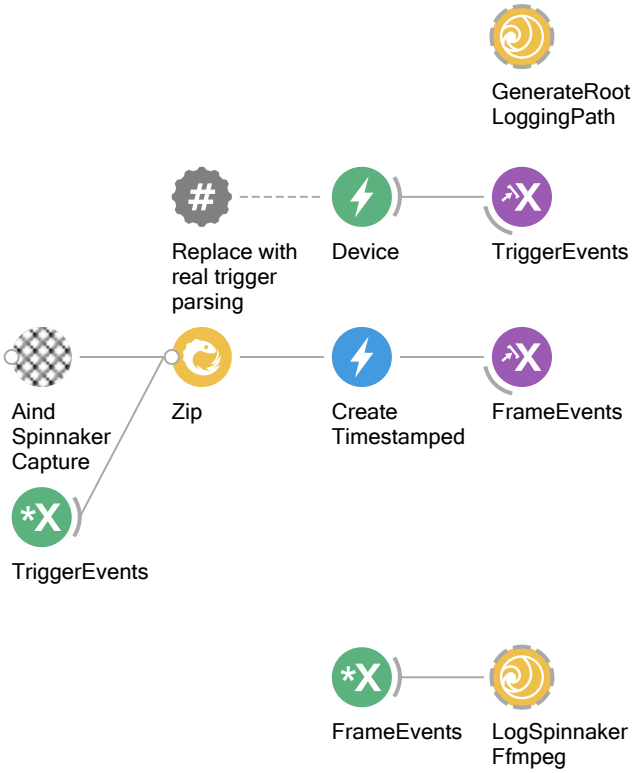
With Metadata

Each device can be saved with metadata by providing the `device.yml` file information to the operator. This string (i.e. the literal of the [`device.yml`]) can be passed manually or by using the `GetMetadata` node from the device-specific package. For example, to log data from a `LicketySplit` device:



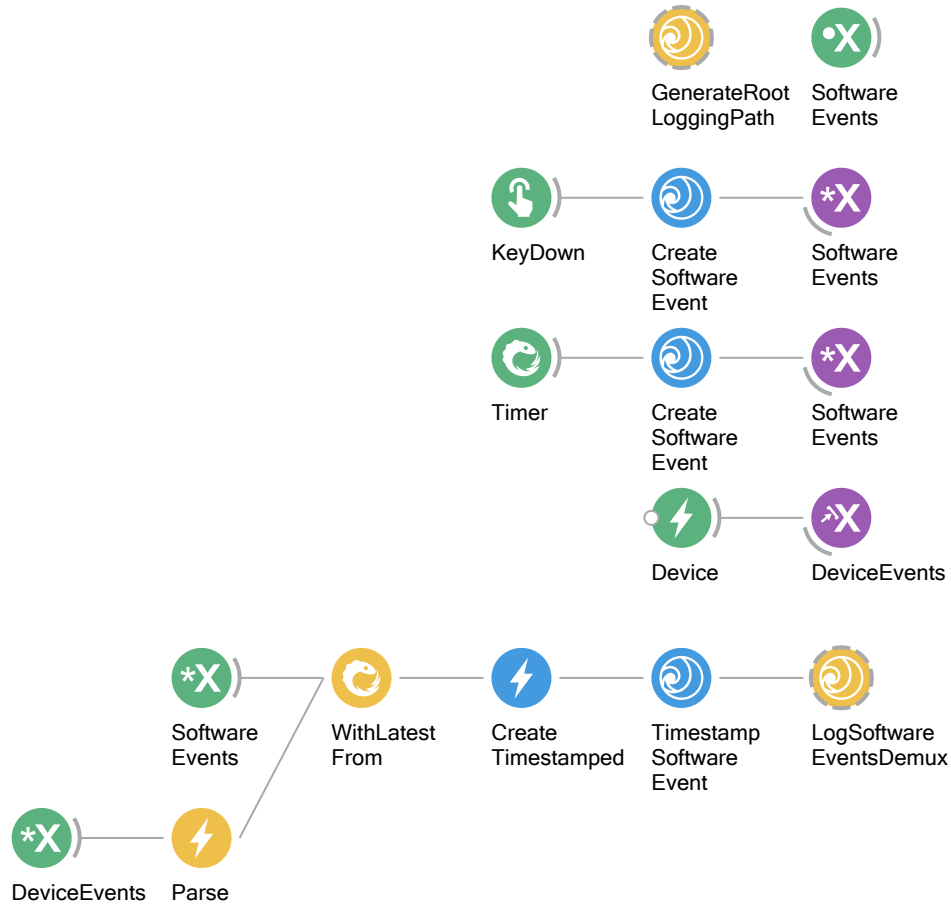
Spinnaker camera

Similarly, for a `Spinnaker Camera`:



Software events

For software events, one can use the `SoftwareEvent` class and the following pattern:



LicketySplitLickDetector

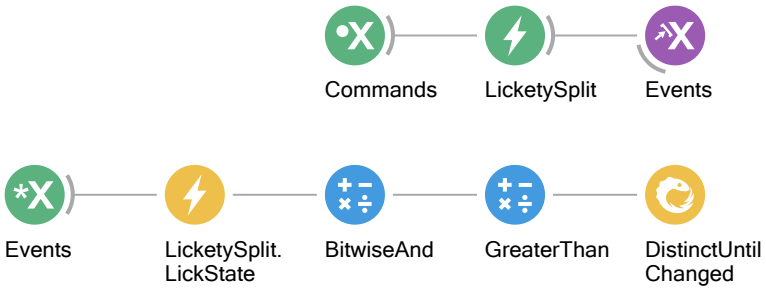
The repository of this device can be found here.

Examples

It is advisable to first get familiar with the general Bonsai interface for Harp devices. This documentation can be found at harp-tech.org.

Detecting licks

Licks detected by the board can be `Parse`d from register `LickState` using the following pattern:



`True` and `False` values will correspond to the onset and offset of lick events, respectively.

SniffDetector

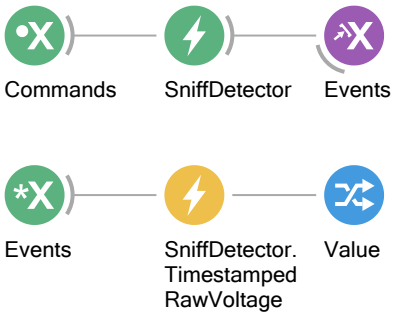
The repository of this device can be found here.

Examples

It is advisable to first get familiar with the general Bonsai interface for Harp devices. This documentation can be found at harp-tech.org.

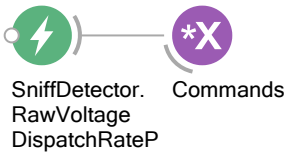
Parsing sensor data

The thermistor data ca be parsed using the following pattern



Setting the dispatch rate

The dispatch rate of the sensor data event can be set using the following pattern:



Treadmill

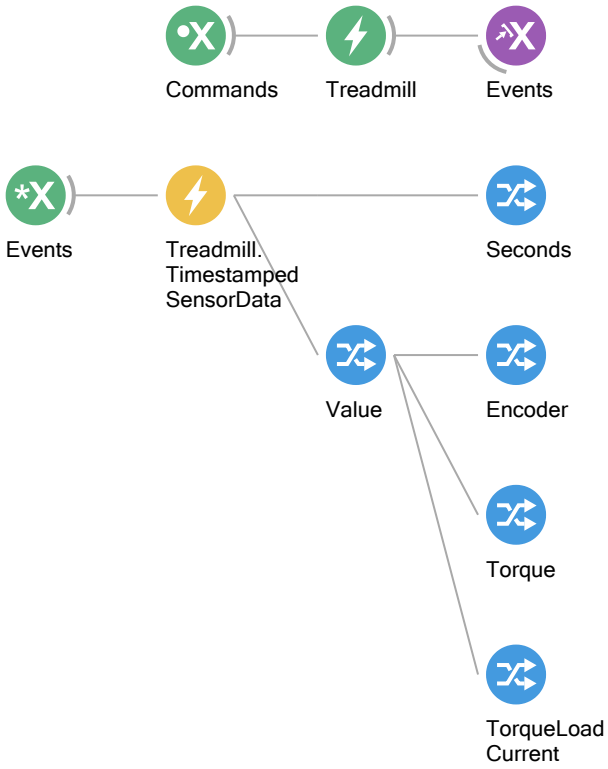
The repository of this device can be found here.

Examples

It is advisable to first get familiar with the general Bonsai interface for Harp devices. This documentation can be found at harp-tech.org.

Parsing sensor data

Device sensor data can be parsed from the individual sensor registers ([`Encoder`, `Torque` and `TorqueLoadCurrent`]). For simplicity, a single register with a packed data structure is also provided. The following pattern can be used to parse the treadmill sensor data:



Using the magnetic particle break

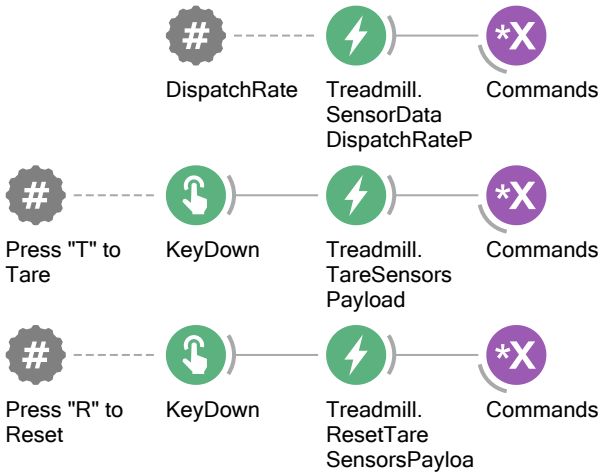
The treadmill is fitted with a magnetic particle break that can be used to control the torque of the wheel experiences up to a set point. The following pattern can be used to control the break. In this example we will use a sinusoidal pattern to set the value of the break, but any other numeric input can be used.

Configuring additional settings

Additional settings are available for the treadmill. These include the ability to:

- Set the dispatch rate of the sensor data event;
- Tare the value of all or individual sensors;
- Reset the Tare function of all or individual sensors;

The following example shows how to achieve all the above, respectively:



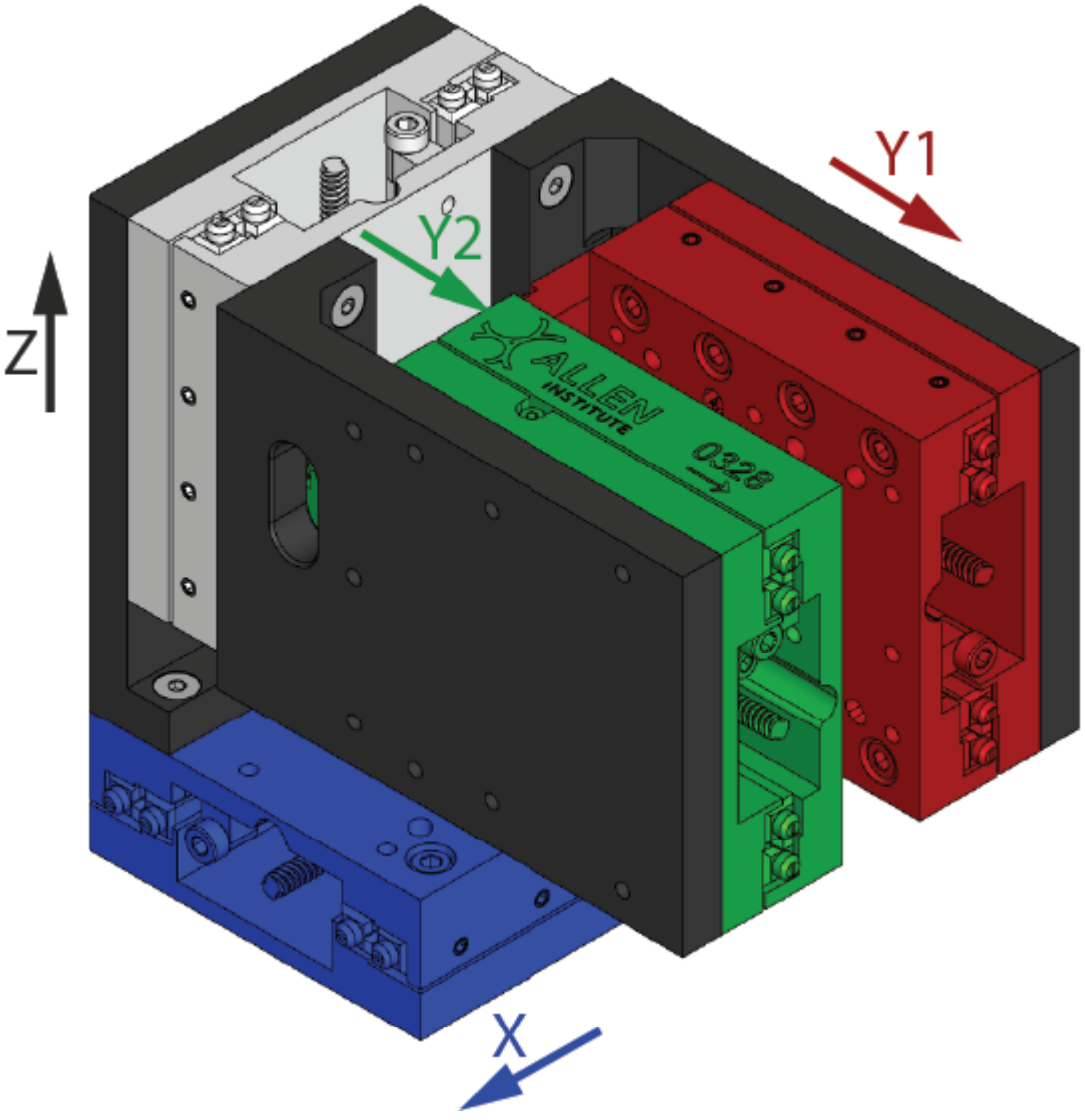
AIND Manipulator
Assembly instructions

The manipulator is composed of two main components:

- The physical manipulator composed of multiple axis that makes up a configuration (e.g. a 4 axis manipulator)
- A controller board that interfaces with the manipulator and the computer

Physical manipulator

The physical manipulator is expected to be assembled as per the following figure. The color of the axis is indicative of the color of the cable that should be connected to the corresponding port on the controller board.



Motor	Axis	Cable Color
0	X (lateral)	Blue
1	Y1 (forward, "port", motor's left mouse's right)	Red
2	Y2 (forward, "starboard", motor's right mouse's left)	Green
3	Z (vertical)	White/Silver

Each motor should also be wired to its respective cable following the color code below:

- Green (B+)
- Green-White (B-)
- Red (A+)
- Red-White (A-)

Finally, each end-of-travel switch follows the color code below:

- Blue (GND)
- White (Signal)
- Brown (+V)

Controller board

Bonsai Interface

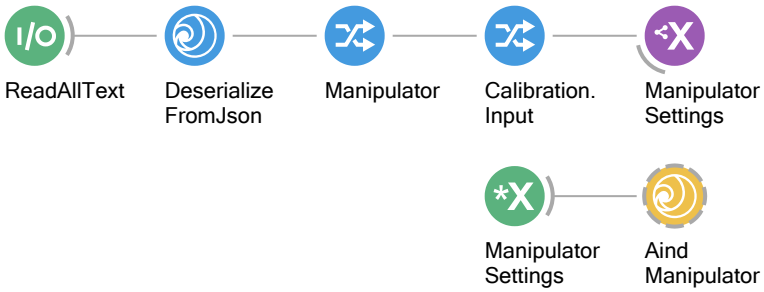
To interface with the motors, the user can choose to use the Harp package for the Harp.StepperDriver. For simplicity, we maintain our own wrapper with core functionality and user-interface. The user can install it from the Aind Nuget feed.

The interface with the manipulator is handled by a single operator: `AindManipulator`. This operator is responsible for the following:

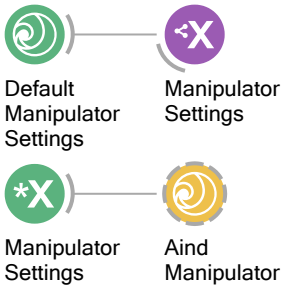
- Setting initial settings of the manipulator
- Implementing `StopMotors`, `HomeMotors`, `MoveTo` and `MoveBy` operations.
- Implementing a front-end interface for the user to interact with the manipulator.

Load a configuration

The operator expects settings to be passed via a `AindManipulatorCalibrationInput` class. This class can defined using the respective schema in `Aind.Behavior.Services` repository. An example on how to generate a configuration file can be found here. The Json file can be passed to the operator using the following pattern:



Alternatively, an operator is provided for quickly defining the settings in the workflow directly. This can be done using the following pattern:



Front-end interface

The operator also provides a front-end interface for the user to interact with the manipulator. This can be accessed by double-clicking the `AindManipulator` while the workflow is running.

Controlling the manipulator

The easiest way to control the manipulator without using the user-interface is to directly interact with the `StopMotors`, `HomeMotors`, `MoveTo` and `MoveBy` subjects.

If the user wants to control the manipulator using the Harp interface, `StepperMotorCommands` and `StepperMotorEvents` can be used to send and receive commands and events respectively.

Environment Sensor

The repository of this device can be found here.

Examples

It is advisable to first get familiar with the general Bonsai interface for Harp devices. This documentation can be found at harp-tech.org.

Parsing sensor data

Device sensor data can be parsed from the individual sensor registers ([`Pressure`, `Temperature`, `Humidity`]). For simplicity, a single register with a packed data structure is also provided. By default, this register will emit a period message with the data from all sensors. The following pattern can be used to parse the register: