Documentation for NeuroBoard

Author: Ben Antonellis

Before showing any documentation, this is the minimum bare example to use this API. Make sure the switch on the board is set to "CONTROL", not "EMG".

```
#include "NeuroBoard.hpp"
NeuroBoard board; // Create NeuroBoard object.
void setup() {
    board.startMeasurements();
}
void loop() {
    int sample = board.getNewSample(); // Get newest sample from the board.
}
```

Index

Common Usage Examples

Setting Button Triggers

If you want to use the red and/or white buttons present on the board, below are a few ways to enable buttons. When you set a function to one of the buttons, each time you press that button the function will be called. The only exception to this is when you call "enableButtonLongPress". This requires that you HOLD DOWN the button for the amount of milliseconds you pass to the function.

```
#include "NeuroBoard.hpp"

NeuroBoard board;

void setup() {

    // Required to start receiving samples from the board //
    board.startMeasurements();

    // Set a trigger for a regular various button presses (RED or WHITE)

    // So when we press the white button, the code present within the function will be called.
    // In this case, "White Button Pressed!" will be printed to the Serial Monitor.
    board.enableButtonPress(WHITE_BTN, []() {
        Serial.println("White Button Pressed!");
    });

    // Same as above, but since we attached this function to the red button, the code will only execute
    // when we press the red button.
    // Same as above, "Red Button Pressed!" will be printed to the Serial Monitor.
    board.enableButtonPress(RED_BTN, []() {
        Serial.println("Red Button Pressed!");
    });

    // Here, if we hold down the white button for 1000 milliseconds (1 second), the code in the function
    // will be called. So, after holding down the button, "White Button Held For 1 Second!" will be printed
    // to the Serial Monitor.
    board.enableButtonLongPress(WHITE_BTN, 1000, []() {
        Serial.println("White Button Held For 1 Second!");
    });

    // Same as above, but for the red button. After holding the red button down for 1 second, we expect
    // "Red Button Held For 1 Second!" to be printed to the Serial Monitor.
    board.enableButtonLongPress(RED_BTN, 1000, []() {
        Serial.println("Red Button Held For 1 Second!");
    });

}

void loop() {

    // Required if any button/envelopeTrigger/servo motor is enabled
    board.handleInputs();

    // loop code here

}
```

Using Envelope Triggers

Below are a couple ways to set an envelope trigger. As described below, the code passed to the function is only executed once the samples received by the board reach the user specified threshold.

```
#include "NeuroBoard.hpp"

NeuroBoard board;

void setup() {

    // Required to start receiving samples from the board //
    board.startMeasurements();

    // Once the incoming sample reaches above set threshold (700 in this case), the
    // function passed is called. Then, once the samples reached 9/10th of the passed
    // threshold (630 in this case), the function will be allowed to call again.

    // Once the threshold is met, the relay is turned on. It is only turned off when
    // the incoming samples are below the second threshold.

    board.setTriggerOnEnvelope(700, []() {
        Serial.println("Threshold Reached!");
    });

    // You can also set your own second threshold. The following code executes the function once
    // 600 is reached. Once the samples drop to 400 or below, only then will the functon be able
    // to be called again once hitting 600.

    board.setTriggerOnEnvelope(600, 400, []() {
        Serial.println("Threshold Reached!");
    });

}

void loop() {

    // Required if any button/envelopeTrigger/servo motor is enabled
    board.handleInputs();

    // loop code here

}
```

It should be noted that when using the setTriggerOnEnvelope function, the relay pin is used automatically. This means once the threshold you pass in gets reached by the samples being collected from the board, the relay pin will be turned on and off.

Using The Servo

Below are four ways you can setup the servo motor. If you want to end the servo motor, simply call "board.endServo();", and the servo motor will be disabled until re-enabled.

```
#include "NeuroBoard.hpp"

NeuroBoard board;

void setup() {

    // Required to start receiving samples from the board //
    board.startMeasurements();

    // **************************************** //
    // OPTION 1 //

    // You can start the servo motor by simplying typing this line in the setup function

    board.startServo();

    // **************************************** //
    // OPTION 2 //

    // Alternatively, we could set the servo motor based on a button pressed //

    board.enableButtonPress(WHITE_BTN, []() {
        board.startServo();
    });

    // **************************************** //
    // OPTION 3 //

    // We could also start the servo motor, and set sensitivity calls to the buttons //

    board.startServo();

    board.enableButtonPress(RED_BTN, []() {
        board.increaseSensitivity();
    });

    board.enableButtonPress(WHITE_BTN, []() {
        board.decreaseSensitivity();
    });

    // **************************************** //
    // OPTION 4 //

    // We could start the servo motor, and set the buttons to change the default servo motor mode //

    board.startServo();

    board.enableButtonPress(RED_BTN, []() {
        board.setServoDefaultPosition(OPEN_MODE);
    });

    board.enableButtonPress(WHITE_BTN, []() {
        board.setServoDefaultPosition(CLOSED_MODE);
    });

    // **************************************** //

}

void loop() {

    // Required if any button/envelopeTrigger/servo motor is enabled
    board.handleInputs();

    // If we want a visual representation of EMG strength, we can call the below function //

    board.displayEMGStrength();

    // And if we want to end the servo motor for any reason //

    board.endServo();

    // loop code here

}
```

NeuroBoard Functions

NeuroBoard::startMeasurements()

Samples data to the circular buffer, and calculates the envelope value all in the background. Sets up pins to the NeuroBoard works correctly. This function is required in the setup function in order for the board to work.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.startMeasurements();
}
```

NeuroBoard::startCommunication()

Enables communication between the Neuroduino and the Serial Port.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.startCommunication();
}
```

NeuroBoard::handleInputs()

Handles all buttons triggers, envelope triggers, and servo motor pings. If you want to set any button/envelope triggers or use any servo motor pings, this must be called in the "loop" function.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void loop() {
    board.handleInputs();
}
```

NeuroBoard::startServo()

Sets up the servo motor for use with the NeuroBoard. Keeps the servo motor enabled until the servo motor is explicitly disabled.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.startServo();
}
```

NeuroBoard::endServo()

Detaches the servo from the NeuroBoard, ending communication. This function must be called if you want to disabled the servo.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void loop() {
    if (/* some condition */) {
        // This isn't the only way to call this function, but it's an option.
        board.endServo();
    }
}
```

NeuroBoard::increaseSensitivity()

Increases the sensitivity for the servo motor. If already at the max sensitivty, this function will do nothing.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void loop() {
    board.increaseSensitivty();
}
```

NeuroBoard::decreaseSensitivity()

Decreases the sensitivity for the servo motor. If already at the minimum sensitivty, this function will do nothing.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void loop() {
    board.decreaseSensitivity();
}
```

NeuroBoard::setServoDefaultPosition()

Toggles the servo motor's default position, from open to closed.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    // You can pass either OPEN_MODE or CLOSED_MODE to the function.
    // OPEN_MODE sets the default position of the servo motor to open.
    // CLOSED_MODE sets the default position of the servo motor to closed.
    board.setServoDefaultPosition(OPEN_MODE);
}
```

NeuroBoard::getNewSample()

Returns the last measured sample the board received from the set channel.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.startMeasurements();
}
void loop() {
    int sample = board.getNewSample();
}
```

NeuroBoard::getSamples(int* arr[], const int& size)

Returns an array of samples, the size of which is specified by the user. You must also pass a reference to an array for the samples to get populated in.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.startMeasurements();
}
void loop() {
    int* samples;
    board.getSamples(&samples, 10); // Populates the samples array with 10 samples.
    delete[] samples; // Frees memory for reallocation.
}
```

NeuroBoard::getEnvelopeValue()

Returns the envelope value of the channel. For every incoming reading from the currently set channel, if the reading isn't larger than the current envelope value, the envelope value is subtracted by a value (default is 1, but the user can set the value themselves). This ensures we don't have an envelope value higher than necessary.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.startMeasurements();
}
void loop() {
    int envelopeValue = board.getEnvelopeValue();
}
```

NeuroBoard::setChannel(const uint8_t& channel)

Sets the current channel to listen on. The boundries for what channels could be set are described below in the code.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.setChannel(A0); // Sets channel to Analog 0 (minimun analog)
    board.setChannel(A5); // Sets channel to Analog 5 (maximum analog for Leonardo)
    board.setChannel(A7); // Sets channel to Analog 7 (maximum analog for Uno)
}
```

NeuroBoard::setDecayRate(const int& rate)

Sets the decay rate for the envelope trigger function. I.E, setDecayRate(5) will subtract the envelope value by 5 for every tick. The value must be positive, but if the user accidently passes a negative value, it will convert to a positive value and continue to work.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.setDecayRate(7); // Subtracts by 7 every tick.
}
```

NeuroBoard::enableButtonPress(const uint8_t& button, const int& interval, void (*callback)(void))

NeuroBoard::enableButtonPress(const uint8_t& button, void (*callback)(void))

Calls the passed function when the specified button is pressed.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void myFunction() {
    Serial.println("Button Pressed!");
}
void setup() {
    // Passing a defined function to the function.
    board.enableButtonPress(RED_BTN, myFunction);

    // You can also pass a lambda function.
    board.enableButtonPress(RED_BTN, []() {
        Serial.println("Button Pressed!");
    });

    // You can also specify an interval, which is a dealy between each button press
    // to accept input again. The value is in milliseconds.
    board.enableButtonPress(WHITE_BTN, 100, []() {
        Serial.println("Button Pressed!");
    });
}
```

NeuroBoard::enableButtonLongPress(const uint8_t& button, const int& milliseconds, void (*callback)(void))

Calls a function when the button has been held for a specific period of time, of which is passed by the user. The number passed is the amount of milliseconds the button needs to be held in order for the function to be executed.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.enableButtonLongPress(WHITE_BTN, 1000, []() {
        Serial.println("Button Held For 1 Second!");
    });
}
```

NeuroBoard::setTriggerOnEnvelope(const int& threshold, const int& secondFactor, void (*callback)(void))

NeuroBoard::setTriggerOnEnvelope(const int& threshold, void (*callback)(void))

Calls a function when the envelope value is greater than the passed threshold.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    // Once the envelope value reaches 700, the function passed is called. Then,
    // once the envelope value falls below 600, the function is able to be called
    // again once reaching 700.
    board.setTriggerOnEnvelope(700, 600, []() {
        Serial.println("Threshold Reached!");
    });

    // Alternatively, you can not pass a second threshold, and one will be calculated
    // for you. The second threshold is 9/10 of the first threshold. So if you pass
    // 1000, the second threshold will be 900, since 900 is 9/10 of 1000.
    board.setTriggerOnEnvelope(1000, []() {
        Serial.println("Threshold Reached!");
    });
}
```

NeuroBoard::displayEMGStrength()

Sets a flag to display the current strength of the readings using the LED bar. If called multiple times, it will cycle through being enabled and disabled.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void setup() {
    board.displayEMGStrength(); // Enabled the EMGStrength to be displayed.
}
```

NeuroBoard::writeLED(const int& led, const bool& state)

Turns on and off an LED, based on what state is passed. Numbers 0 - 7.

```
#include "NeuroBoard.hpp"
NeuroBoard board;
void loop() {
    board.writeLED(0, ON); // Turns ON the first LED on the bar.
    board.writeLED(0, OFF); // Turns OFF the first LED on the bar.

    // If you want to turn on multiple LEDs at a time, you can use a loop.
    // The following code turns on all LEDs on the bar.
    // MAX_LEDS is board specific. For Leonardo, its 8. For Uno, its 6.
    // ledPins is also board specific.
    // - For Leonardo, it's [0, 7].
    // - For Uno, it's [8, 13].
    for (int i = 0; i < MAX_LEDS; i++) {
        board.writeLED(ledPins[i], ON);
    }

    // And you can also turn them all off.
    for (int i = 0; i < MAX_LEDS; i++) {
        board.writeLED(ledPins[i], OFF);
    }
}
```