Braille Glove

Generated by Doxygen 1.13.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

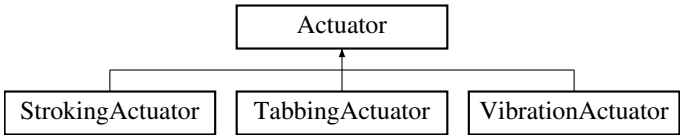
Class Documentation

4.1 Actuator Class Reference

Abstract class for implementing different types of actuators.

`#include <Actuator.h>`

Inheritance diagram for Actuator:



Public Member Functions

- Actuator (int pin, ActuatorType type)

  Constructor for the Actuator class.
- virtual void activate ()=0

  Pure virtual function to activate the actuator.
- virtual void deactivate ()=0

  Pure virtual function to deactivate the actuator.

Protected Attributes

- int **pin**

  GPIO pin number the actuator is connected to.
- ActuatorType **actuatorType**

  Type of actuator (e.g., vibration, stroking, tabbing).
- bool **turnedOn** = false

  Flag to check if the actuator is currently active.

4.1.1 Detailed Description

Abstract class for implementing different types of actuators.

This class provides a base for all actuator types, defining common properties and methods.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Actuator()

```
Actuator::Actuator (
            int pin,
            ActuatorType type)  [inline]
```

Constructor for the Actuator class.

Parameters

pin	The GPIO pin to which the actuator is connected.
type	The type of actuator.

4.1.3 Member Function Documentation

4.1.3.1 activate()

```
virtual void Actuator::activate ()  [pure virtual]
```

Pure virtual function to activate the actuator.

This function must be implemented by derived classes to define how the actuator should be activated.

Implemented in StrokingActuator, TabbingActuator, and VibrationActuator.

4.1.3.2 deactivate()

```
virtual void Actuator::deactivate ()  [pure virtual]
```

Pure virtual function to deactivate the actuator.

This function must be implemented by derived classes to define how the actuator should be deactivated.

Implemented in StrokingActuator, TabbingActuator, and VibrationActuator.

The documentation for this class was generated from the following file:

- src/ActuatorTypes/Actuator.h

4.2 ActuatorProcessingOrderMapper Class Reference

Reorders braille dot numbers based on finger sensitivity order.

`#include <ActuatorProcessingOrderMapper.h>`

Public Member Functions

- ActuatorProcessingOrderMapper ()
  Constructor for ActuatorProcessingOrderMapper.
- std::vector< int > reorderVectorBySensitivity (const std::vector< int > &values)
  Reorders a vector of braille chords based on sensitivity.

4.2.1 Detailed Description

Reorders braille dot numbers based on finger sensitivity order.

This class processes braille chords and reorders the actuation sequence based on predefined sensitivity levels of each dot.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ActuatorProcessingOrderMapper()

```
ActuatorProcessingOrderMapper::ActuatorProcessingOrderMapper ()
```

Constructor for ActuatorProcessingOrderMapper.

Initializes the sensitivity order mapping.

4.2.3 Member Function Documentation

4.2.3.1 reorderVectorBySensitivity()

```
std::vector< int > ActuatorProcessingOrderMapper::reorderVectorBySensitivity (
            const std::vector< int > & values)
```

Reorders a vector of braille chords based on sensitivity.

Each braille chord in the vector is restructured based on the predefined sensitivity order.

Parameters

values	A vector of braille chords encoded as integers.

Returns

A reordered vector with braille chords sorted by sensitivity.

The documentation for this class was generated from the following files:

- src/Mapper/ActuatorProcessingOrderMapper.h
- src/Mapper/ActuatorProcessingOrderMapper.cpp

4.3 BrailleMapper Class Reference

Maps characters to their corresponding Braille integer representations.

`#include <BrailleMapper.h>`

Public Member Functions

- **BrailleMapper** ()

  Constructs a BrailleMapper object and initializes mappings.
- int getBrailleHash (char letter) const

  Retrieves the Braille integer representation of a given letter.
- std::vector< int > stringToIntegerList (const String &input) const

  Converts a string into a list of Braille integer representations.

4.3.1 Detailed Description

Maps characters to their corresponding Braille integer representations.

This class provides functionality to convert individual characters and strings into Braille numerical representations based on English Tier One Braille.

4.3.2 Member Function Documentation

4.3.2.1 getBrailleHash()

```
int BrailleMapper::getBrailleHash (
            char letter) const
```

Retrieves the Braille integer representation of a given letter.

Parameters

letter	The character to be mapped.

Returns

The corresponding Braille integer representation.

4.3.2.2 stringToIntegerList()

```
std::vector< int > BrailleMapper::stringToIntegerList (
            const String & input) const
```

Converts a string into a list of Braille integer representations.

Given an input string (e.g., "hello"), this function returns a vector containing the corresponding Braille integer values for each character.

Parameters

input	The input string to convert.

Returns

A vector of integers representing the Braille values of the characters.

The documentation for this class was generated from the following files:

- src/Mapper/BrailleMapper.h
- src/Mapper/BrailleMapper.cpp

4.4 Controller Class Reference

Handles the initialization and execution of either the master or slave mode.

```
#include <Controller.h>
```

Public Member Functions

- Controller (bool isSlave)
  Constructor for the Controller class.
- void setup ()
  Sets up the master or slave mode based on the given conditions.
- void loop ()
  Runs the main execution loop for either the master or slave mode.

4.4.1 Detailed Description

Handles the initialization and execution of either the master or slave mode.

This class determines whether the system should operate as a master or a slave and initializes the appropriate class accordingly.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Controller()

```
Controller::Controller (
            bool isSlave)
```

Constructor for the Controller class.

Determines whether the device should act as a master or a slave.

Parameters

isSlave	A boolean flag indicating whether the device should run in slave mode.

4.4.3 Member Function Documentation

4.4.3.1 loop()

```
void Controller::loop ()
```

Runs the main execution loop for either the master or slave mode.

This function should be called continuously in the main program loop.

4.4.3.2 setup()

```
void Controller::setup ()
```

Sets up the master or slave mode based on the given conditions.

This function initializes the appropriate components depending on whether the device is in master or slave mode.

The documentation for this class was generated from the following files:

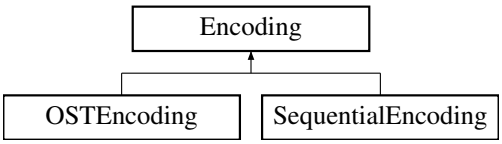
- src/Controller/Controller.h
- src/Controller/Controller.cpp

4.5 Encoding Class Reference

Base class for encoding operations.

```
#include <Encoding.h>
```

Inheritance diagram for Encoding:



Static Public Member Functions

- static void customDelay (unsigned long timeInMs)

  Custom delay function to provide a non-blocking delay.
- static bool validIndex (int number, Hand hand)

  Checks whether the given pin number is valid for the specified hand.

4.5.1 Detailed Description

Base class for encoding operations.

This class provides the basic functions for encoding used by its child classes.

4.5.2 Member Function Documentation

4.5.2.1 customDelay()

```
static void Encoding::customDelay (
            unsigned long timeInMs)  [inline], [static]
```

Custom delay function to provide a non-blocking delay.

This function allows for a non-blocking delay (unlike the blocking `delay()` function in Arduino), so the program can continue execution while waiting.

Parameters

timeInMs	The delay duration in milliseconds.

4.5.2.2 validIndex()

```
static bool Encoding::validIndex (
            int number,
            Hand hand)  [inline], [static]
```

Checks whether the given pin number is valid for the specified hand.

This function validates whether the pin number belongs to the correct hand (left or right) based on the actuator configuration.

Parameters

number	The pin number to be validated.
hand	The hand (left or right) for which the validation is being done.

Returns

True if the pin is valid for the hand; false otherwise.

The documentation for this class was generated from the following file:

- src/Models/EncodingScheme/Encoding.h

4.6 GloveModel Class Reference

Represents the glove model, simulating actuator behavior on a given hand.

`#include <GloveModel.h>`

Public Member Functions

- GloveModel (Hand hand, Actuator &actuator1, Actuator &actuator2, Actuator &actuator3)

  Constructs a GloveModel object with the given hand and actuators.
- void resetAllActuators ()

  Resets all actuators to their initial state.
- void executePatternAt (int index)

  Executes a pattern of actuations at the specified index in the pattern.
- void pauseBetweenLetters ()

  Pauses between letters during a sequence of actuations.
- void activateOnNumber (int number)

  Activates the stimuli on a given number representing a specific actuator pattern.
- void setPattern (std::vector< int > newValues)

  Sets a new pattern of actuations to be played.
- std::vector< int > getPattern ()

  Retrieves the current pattern of actuations.
- int getPatternLength ()

  Returns the length of the current pattern.
- void setChordMode (ChordingScheme chordMode)

  Sets the chording mode for the glove, either sequential or OST encoding.

4.6.1 Detailed Description

Represents the glove model, simulating actuator behavior on a given hand.

This class is used to simulate the actuations of actuators on a glove. It defines the pattern of actuations and the chording scheme (OST or sequential). The class also provides methods to control and reset actuators and manage the sequence of actuations.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 GloveModel()

```
GloveModel::GloveModel (
            Hand hand,
            Actuator & actuator1,
            Actuator & actuator2,
            Actuator & actuator3)  [inline]
```

Constructs a GloveModel object with the given hand and actuators.

Initializes the glove with actuators and assigns the specified hand (left or right).

Parameters

hand	The hand (left or right) of the glove.
actuator1	First actuator of the glove.
actuator2	Second actuator of the glove.
actuator3	Third actuator of the glove.

4.6.3 Member Function Documentation

4.6.3.1 activateOnNumber()

```
void GloveModel::activateOnNumber (
            int number)  [inline]
```

Activates the stimuli on a given number representing a specific actuator pattern.

The method activates and deactivates actuators based on the given number and the selected encoding scheme (OST or sequential).

Parameters

number	The number representing the actuator pattern to be activated using the specific stimuli.

4.6.3.2 executePatternAt()

```
void GloveModel::executePatternAt (
            int index)  [inline]
```

Executes a pattern of actuations at the specified index in the pattern.

Resets all actuators and vibrates based on the pattern at the given index.

Parameters

index	The index of the pattern to be executed.

4.6.3.3 getPattern()

```
std::vector< int > GloveModel::getPattern ()  [inline]
```

Retrieves the current pattern of actuations.

This method returns the current pattern of actuations that is set.

Returns

A vector containing the current pattern of actuations.

4.6.3.4 getPatternLength()

```
int GloveModel::getPatternLength ()  [inline]
```

Returns the length of the current pattern.

Returns

The length of the current pattern.

4.6.3.5 pauseBetweenLetters()

```
void GloveModel::pauseBetweenLetters ()  [inline]
```

Pauses between letters during a sequence of actuations.

This function applies a custom delay and resets all actuators after the delay.

4.6.3.6 resetAllActuators()

```
void GloveModel::resetAllActuators ()  [inline]
```

Resets all actuators to their initial state.

This method deactivates all actuators, ensuring that they start in the same condition.

4.6.3.7 setChordMode()

```
void GloveModel::setChordMode (
            ChordingScheme chordMode)  [inline]
```

Sets the chording mode for the glove, either sequential or OST encoding.

This method selects the chording mode (either sequential or OST) for the actuations.

Parameters

chordMode	The desired chording scheme (SEQUENTIAL_ENCODING or OST_ENCODING).

4.6.3.8 setPattern()

```
void GloveModel::setPattern (
            std::vector< int > newValues)  [inline]
```

Sets a new pattern of actuations to be played.

This method allows the user to define a new pattern of actuations.

Parameters

newValues	A vector containing the new pattern of actuations.

The documentation for this class was generated from the following file:

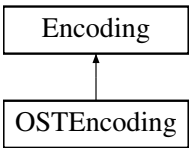
- src/Models/GloveModel.h

4.7 OSTEncoding Class Reference

Handles the OST encoding scheme for actuators.

`#include <OSTEncoding.h>`

Inheritance diagram for OSTEncoding:



Static Public Member Functions

- static void handle (int number, Actuator **actuators, Hand hand)
  Activates an actuator based on the OST encoding sequence.

Static Public Member Functions inherited from Encoding

- static void customDelay (unsigned long timeInMs)
  Custom delay function to provide a non-blocking delay.
- static bool validIndex (int number, Hand hand)
  Checks whether the given pin number is valid for the specified hand.

4.7.1 Detailed Description

Handles the OST encoding scheme for actuators.

This class defines the OST encoding scheme and how the actuators should be activated based on that scheme.

4.7.2 Member Function Documentation

4.7.2.1 handle()

```
static void OSTEncoding::handle (
            int number,
            Actuator ** actuators,
            Hand hand)  [inline], [static]
```

Activates an actuator based on the OST encoding sequence.

This function handles the activation of actuators based on the OST encoding scheme. The activation sequence is determined by the index number and the hand (left or right). The appropriate actuator is activated according to the given index, and a custom delay is applied.

Parameters

number	The index number representing the actuator to be activated.
actuators	The array of actuator pointers to be used.
hand	The hand (left or right) to which the actuator belongs.

The documentation for this class was generated from the following file:

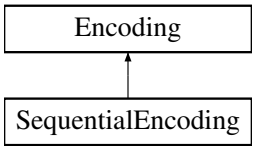
- src/Models/EncodingScheme/OSTEncoding.h

4.8 SequentialEncoding Class Reference

Handles the sequential encoding scheme for actuators.

`#include <SequentialEncoding.h>`

Inheritance diagram for SequentialEncoding:



Static Public Member Functions

- static void handle (int number, Actuator **actuators, Hand hand)

  Activates and deactivates an actuator based on the sequential encoding scheme.

Static Public Member Functions inherited from Encoding

- static void customDelay (unsigned long timeInMs)

  Custom delay function to provide a non-blocking delay.
- static bool validIndex (int number, Hand hand)

  Checks whether the given pin number is valid for the specified hand.

4.8.1 Detailed Description

Handles the sequential encoding scheme for actuators.

This class defines the sequential encoding scheme and how the actuators should be activated in sequence.

4.8.2 Member Function Documentation

4.8.2.1 handle()

```
static void SequentialEncoding::handle (
            int number,
            Actuator ** actuators,
            Hand hand)  [inline], [static]
```

Activates and deactivates an actuator based on the sequential encoding scheme.

This function handles the sequential encoding scheme by activating the actuator corresponding to the given pin number and hand (left or right). After activation, the actuator is deactivated after a specified duration. A delay is applied both after activation and deactivation.

Parameters

number	The index number representing the actuator to be activated.
actuators	The array of actuator pointers to be used.
hand	The hand (left or right) to which the actuator belongs.

The documentation for this class was generated from the following file:

- src/Models/EncodingScheme/SequentialEncoding.h

4.9 SingeltonGloveSettings Class Reference

Singleton class that defines all the different glove settings.

`#include <SingeltonGloveSettings.h>`

Static Public Member Functions

- static SingeltonGloveSettings & getInstance ()

  Gets the singleton instance of the SingeltonGloveSettings class.

Public Attributes

- const int **OST_OFFSET** = 10

  The offset between activations in the OST encoding.
- const int **DURATION** = 200

  Duration of actuator activation in milliseconds.
- const int **PAUSE** = 2000

  Pause time between two characters in milliseconds.
- const int **NUM_ACTUATORS** = 3

  The number of actuators used in the glove.
- const int **AUDIO_STIMULI_OFFSET** = 100

  The offset between audio stimuli and tactile stimuli in milliseconds.
- const int **SEQ_OFFSET** = 1000

  The offset between characters in the Sequential encoding scheme.
- const int **studyOstRepititions** = 126

  Number of repetitions for the OST encoding in the study.
- const int **studySeqRepititions** = 44

  Number of repetitions for the Sequential encoding in the study.

4.9.1 Detailed Description

Singleton class that defines all the different glove settings.

This class follows the Singleton pattern, ensuring that only one instance of the glove settings exists. It contains various constants that define the parameters for actuator timing, offsets, and repetitions related to the glove's operation.

4.9.2 Member Function Documentation

4.9.2.1 getInstance()

```
static SingeltonGloveSettings & SingeltonGloveSettings::getInstance ()  [inline], [static]
```

Gets the singleton instance of the SingeltonGloveSettings class.

Returns

A reference to the single instance of SingeltonGloveSettings.

The documentation for this class was generated from the following file:

- src/Settings/SingeltonGloveSettings.h

4.10 SingeltonWifiConnector Class Reference

Singleton class for Wi-Fi settings.

```
#include <SingeltonWifiSettings.h>
```

Static Public Member Functions

- static SingeltonWifiConnector & getInstance ()
  Gets the singleton instance of the SingeltonWifiConnector class.

Public Attributes

- const char * **MASTER_SSID** = "MV-Glove"
  The SSID name for the master Wi-Fi network.
- const char * **SLAVE_SSID** = "VS-Glove"
  The SSID name for the slave Wi-Fi network.
- const uint8_t SLAVE_MAC [6] = {0x48, 0x55, 0x19, 0xF6, 0xC9, 0xB3}
  The MAC address of the slave device for the Wi-Fi connection.

4.10.1 Detailed Description

Singleton class for Wi-Fi settings.

This class follows the Singleton pattern to ensure only one instance exists for managing Wi-Fi settings, including SSID names and MAC address for master and slave devices in the system.

4.10.2 Member Function Documentation

4.10.2.1 getInstance()

```
static SingeltonWifiConnector & SingeltonWifiConnector::getInstance ()  [inline], [static]
```

Gets the singleton instance of the SingeltonWifiConnector class.

Returns

A reference to the single instance of SingeltonWifiConnector.

4.10.3 Member Data Documentation

4.10.3.1 SLAVE_MAC

```
const uint8_t SingeltonWifiConnector::SLAVE_MAC[6] = {0x48, 0x55, 0x19, 0xF6, 0xC9, 0xB3}
```

The MAC address of the slave device for the Wi-Fi connection.

The MAC address is used to identify the slave device in the network.

The documentation for this class was generated from the following file:

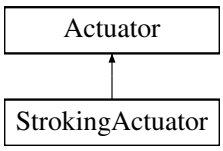
- src/Settings/SingeltonWifiSettings.h

4.11 StrokingActuator Class Reference

A class representing a stroking actuator.

```
#include <StrokingActuator.h>
```

Inheritance diagram for StrokingActuator:



Public Member Functions

- StrokingActuator (int pin)
  Constructor for StrokingActuator.
- void activate () override
  Activates the stroking actuator.
- void deactivate () override
  Deactivates the stroking actuator.

Public Member Functions inherited from Actuator

- Actuator (int pin, ActuatorType type)

  Constructor for the Actuator class.

Additional Inherited Members

Protected Attributes inherited from Actuator

- int **pin**

  GPIO pin number the actuator is connected to.
- ActuatorType **actuatorType**

  Type of actuator (e.g., vibration, stroking, tabbing).
- bool **turnedOn** = false

  Flag to check if the actuator is currently active.

4.11.1 Detailed Description

A class representing a stroking actuator.

This actuator is designed to create a stroking sensation using a servo motor.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 StrokingActuator()

```
StrokingActuator::StrokingActuator (
            int pin)  [inline]
```

Constructor for StrokingActuator.

Initializes the stroking actuator by attaching the servo to the specified pin and setting it to the initial position (0 degrees).

Parameters

pin	The GPIO pin to which the actuator is connected.

4.11.3 Member Function Documentation

4.11.3.1 activate()

```
void StrokingActuator::activate ()  [inline], [override], [virtual]
```

Activates the stroking actuator.

Moves the servo to 180 degrees to simulate a stroking motion.

Implements Actuator.

4.11.3.2 deactivate()

```
void StrokingActuator::deactivate ()  [inline], [override], [virtual]
```

Deactivates the stroking actuator.

Moves the servo back to 0 degrees.

Implements Actuator.

The documentation for this class was generated from the following file:

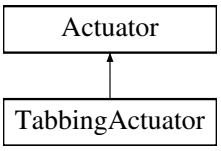
- src/ActuatorTypes/StrokingActuator.h

4.12 TabbingActuator Class Reference

A class representing a tabbing actuator.

```
#include <TabbingActuator.h>
```

Inheritance diagram for TabbingActuator:



Public Member Functions

- TabbingActuator (int pin)
  Constructor for TabbingActuator.
- void activate () override
  Activates the tabbing actuator.
- void deactivate () override
  Deactivates the tabbing actuator.

Public Member Functions inherited from Actuator

- Actuator (int pin, ActuatorType type)
  Constructor for the Actuator class.

Additional Inherited Members

Protected Attributes inherited from Actuator

- int **pin**
  GPIO pin number the actuator is connected to.
- ActuatorType **actuatorType**
  Type of actuator (e.g., vibration, stroking, tabbing).
- bool **turnedOn** = false
  Flag to check if the actuator is currently active.

4.12.1 Detailed Description

A class representing a tabbing actuator.

This actuator is designed to create a tapping or tabbing sensation using a servo motor.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 TabbingActuator()

```
TabbingActuator::TabbingActuator (
            int pin)  [inline]
```

Constructor for TabbingActuator.

Initializes the tabbing actuator by attaching the servo to the specified pin and setting it to the initial position (180 degrees).

Parameters

pin	The GPIO pin to which the actuator is connected.

4.12.3 Member Function Documentation

4.12.3.1 activate()

```
void TabbingActuator::activate ()  [inline], [override], [virtual]
```

Activates the tabbing actuator.

Moves the servo to 90 degrees to simulate a tabbing motion.

Implements Actuator.

4.12.3.2 deactivate()

```
void TabbingActuator::deactivate ()  [inline], [override], [virtual]
```

Deactivates the tabbing actuator.

Moves the servo back to 180 degrees.

Implements Actuator.

The documentation for this class was generated from the following file:

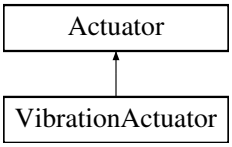
- src/ActuatorTypes/TabbingActuator.h

4.13 VibrationActuator Class Reference

A class representing a vibration actuator.

`#include <VibrationActuator.h>`

Inheritance diagram for VibrationActuator:



Public Member Functions

- VibrationActuator (int pin)
  Constructor for VibrationActuator.
- void activate () override
  Activates the vibration actuator.
- void deactivate () override
  Deactivates the vibration actuator.

Public Member Functions inherited from Actuator

- Actuator (int pin, ActuatorType type)
  Constructor for the Actuator class.

Additional Inherited Members

Protected Attributes inherited from Actuator

- int **pin**
  GPIO pin number the actuator is connected to.
- ActuatorType **actuatorType**
  Type of actuator (e.g., vibration, stroking, tabbing).
- bool **turnedOn** = false
  Flag to check if the actuator is currently active.

4.13.1 Detailed Description

A class representing a vibration actuator.

This actuator uses a digital output pin to control a vibration motor.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 VibrationActuator()

```
VibrationActuator::VibrationActuator (
            int pin)  [inline]
```

Constructor for VibrationActuator.

Initializes the vibration actuator by setting the specified pin as an output and turning off the vibration motor initially.

Parameters

pin	The GPIO pin to which the actuator is connected.

4.13.3 Member Function Documentation

4.13.3.1 activate()

```
void VibrationActuator::activate ()  [inline], [override], [virtual]
```

Activates the vibration actuator.

Turns on the vibration motor if it is not already on.

Implements Actuator.

4.13.3.2 deactivate()

```
void VibrationActuator::deactivate ()  [inline], [override], [virtual]
```

Deactivates the vibration actuator.

Turns off the vibration motor if it is currently on.

Implements Actuator.

The documentation for this class was generated from the following file:

- src/ActuatorTypes/VibrationActuator.h

4.14 WifiMaster Class Reference

< Include for TCP server

```
#include <WifiMaster.h>
```

Public Member Functions

- WifiMaster (GloveModel gloveModel)
  Constructs a WifiMaster object with a given glove model.
- void setup ()
  Standard setup function for the Wi-Fi master.
- void loop ()
  Standard loop function for the Wi-Fi master.
- void sendVectorToSlave (const std::vector< int > &reorderedValues, const ChordingScheme status, int repeat)
  Sends a vector to the slave device, including a repeat count for longer patterns.
- void sendVectorToSlave (const std::vector< int > &reorderedValues, const ChordingScheme status)
  Sends a vector to the slave device for a short pattern (no repeat).

4.14.1 Detailed Description

< Include for TCP server

This class handles the communication and Wi-Fi functionality for the master device, including website setup, response handling, and communication between the master and slave devices.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 WifiMaster()

```
WifiMaster::WifiMaster (
            GloveModel gloveModel)
```

Constructs a WifiMaster object with a given glove model.

Parameters

gloveModel	The glove model to be used for actuator control and pattern execution.

4.14.3 Member Function Documentation

4.14.3.1 loop()

```
void WifiMaster::loop ()
```

Standard loop function for the Wi-Fi master.

Continuously handles communication and controls actuators.

4.14.3.2 sendVectorToSlave() [1/2]

```
void WifiMaster::sendVectorToSlave (
            const std::vector< int > & reorderedValues,
            const ChordingScheme status)
```

Sends a vector to the slave device for a short pattern (no repeat).

Parameters

reorderedValues	The vector of reordered values representing the pattern.
status	The current chording scheme status.

4.14.3.3 sendVectorToSlave() [2/2]

```
void WifiMaster::sendVectorToSlave (
            const std::vector< int > & reorderedValues,
            const ChordingScheme status,
            int repeat)
```

Sends a vector to the slave device, including a repeat count for longer patterns.

Parameters

reorderedValues	The vector of reordered values representing the pattern.
status	The current chording scheme status.
repeat	The number of times to repeat the pattern if it is long.

4.14.3.4 setup()

```
void WifiMaster::setup ()
```

Standard setup function for the Wi-Fi master.

Sets up Wi-Fi, ESP-NOW, and the web server.

The documentation for this class was generated from the following files:

- src/Master/WifiMaster.h
- src/Master/WifiMaster.cpp

4.15 WifiSlave Class Reference

< Include for TCP server

```
#include <WifiSlave.h>
```

Public Member Functions

- WifiSlave (GloveModel gloveModel)
  Constructs a WifiSlave object with a given glove model.
- void setup ()
  Standard setup function for the Wi-Fi slave.
- void loop ()
  Standard loop function for the Wi-Fi slave.
- void processMessage (const uint8_t *mac, const uint8_t *buf, size_t count)
  Processes the received message.

Static Public Member Functions

- static void onReceiveCallback (const uint8_t *mac, const uint8_t *buf, size_t count, void *arg)
  Callback function for handling received data.

4.15.1 Detailed Description

< Include for TCP server

This class defines the behavior of a WiFi slave device, including receiving data via Wi-Fi and processing it to control actuators.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 WifiSlave()

```
WifiSlave::WifiSlave (
            GloveModel gloveModel)
```

Constructs a WifiSlave object with a given glove model.

Parameters

gloveModel	The glove model to be used for actuator control and pattern execution.

4.15.3 Member Function Documentation

4.15.3.1 loop()

```
void WifiSlave::loop ()
```

Standard loop function for the Wi-Fi slave.

Continuously handles incoming data and controls the actuators accordingly.

4.15.3.2 onReceiveCallback()

```
void WifiSlave::onReceiveCallback (
            const uint8_t * mac,
            const uint8_t * buf,
            size_t count,
            void * arg)  [static]
```

Callback function for handling received data.

This function is called when data is received via Wi-Fi. It identifies the type of data and processes it accordingly.

Parameters

mac	The MAC address of the sender.
buf	The received data buffer.
count	The number of bytes received.
arg	Additional argument passed to the callback (not used here).

4.15.3.3 processMessage()

```
void WifiSlave::processMessage (
            const uint8_t * mac,
            const uint8_t * buf,
            size_t count)
```

Processes the received message.

This function processes the message and takes the appropriate actions based on the content of the message.

Parameters

mac	The MAC address of the sender.
buf	The received data buffer.
count	The number of bytes received.

4.15.3.4 setup()

```
void WifiSlave::setup ()
```

Standard setup function for the Wi-Fi slave.

Initializes necessary components for Wi-Fi communication and the glove model.

The documentation for this class was generated from the following files:

- src/Slave/WifiSlave.h
- src/Slave/WifiSlave.cpp

Chapter 5

File Documentation

5.1 Actuator.h

```
00001 #ifndef ACTUATOR_H
00002 #define ACTUATOR_H
00003
00004 #include "ActuatorType.h"
00005
00006 #ifdef UNIT_TEST
00007     #include "../test/Mocks/Servo_Mock.h"
00008 #else
00009     #include <Servo.h>
00010     #include <Arduino.h>
00011 #endif
00012
00013
00020 class Actuator {
00021   protected:
00022     int pin;
00023     ActuatorType actuatorType;
00024     bool turnedOn = false;
00025
00026   public:
00032     Actuator(int pin, ActuatorType type) : pin(pin), actuatorType(type) {}
00033
00040     virtual void activate() = 0;
00041
00048     virtual void deactivate() = 0;
00049 };
00050
00051 #endif // ACTUATOR_H
```

5.2 ActuatorType.h

```
00001 #ifndef ACTUATOR_TYPE_H
00002 #define ACTUATOR_TYPE_H
00003
00010 enum ActuatorType {
00011     Vibration,
00012     Tabbing,
00013     Stroking
00014 };
00015
00016 #endif // ACTUATOR_TYPE_H
```

5.3 StrokingActuator.h

```
00001 #ifndef STROKING_ACTUATOR_H
00002 #define STROKING_ACTUATOR_H
00003
00004 #include "Actuator.h"
```

```
00005
00012 class StrokingActuator : public Actuator {
00013   private:
00014     Servo servo;
00015
00016   public:
00025     StrokingActuator(int pin) : Actuator(pin, Stroking) {
00026         servo.attach(pin);
00027         servo.write(0);
00028         turnedOn = false;
00029     }
00030
00036     void activate() override {
00037         turnedOn = true;
00038         servo.write(180);
00039     }
00040
00046     void deactivate() override {
00047         turnedOn = false;
00048         servo.write(0);
00049     }
00050 };
00051
00052 #endif // STROKING_ACTUATOR_H
```

5.4 TabbingActuator.h

```
00001 #ifndef TABBING_ACTUATOR_H
00002 #define TABBING_ACTUATOR_H
00003
00004 #include "Actuator.h"
00005
00012 class TabbingActuator : public Actuator {
00013   private:
00014     Servo servo;
00015
00016   public:
00025     TabbingActuator(int pin) : Actuator(pin, Stroking) {
00026         servo.attach(pin);
00027         servo.write(180);
00028         turnedOn = false;
00029     }
00030
00036     void activate() override {
00037         turnedOn = true;
00038         servo.write(90);
00039     }
00040
00046     void deactivate() override {
00047         turnedOn = false;
00048         servo.write(180);
00049     }
00050 };
00051
00052 #endif // TABBING_ACTUATOR_H
```

5.5 VibrationActuator.h

```
00001 #ifndef VIBRATION_ACTUATOR_H
00002 #define VIBRATION_ACTUATOR_H
00003
00004 #include "Actuator.h"
00005
00012 class VibrationActuator : public Actuator {
00013   public:
00022     VibrationActuator(int pin) : Actuator(pin, Vibration) {
00023         pinMode(pin, OUTPUT);
00024         digitalWrite(pin, LOW);
00025         turnedOn = false;
00026     }
00027
00033     void activate() override {
00034         if (!turnedOn) {
00035             turnedOn = true;
00036             digitalWrite(pin, HIGH);
00037         }
00038     }
00039
```

```
00045     void deactivate() override {
00046         if (turnedOn) {
00047             turnedOn = false;
00048             digitalWrite(pin, LOW);
00049         }
00050     }
00051 };
00052
00053 #endif // VIBRATION_ACTUATOR_H
```

5.6 Controller.h

```
00001 #ifndef CONTROLLER_H
00002 #define CONTROLLER_H
00003
00004 #ifdef UNIT_TEST
00005     #include "../test/Mocks/ESP8266WiFi_Mock.h"
00006     #include "../test/Mocks/MockWiFiUDP.h"
00007     #include "../test/Mocks/new_Arduino_Mock.h"
00008     #include "../test/Mocks/ESPNOW_Mock.h"
00009     #include "../test/Mocks/ESP_Mock.h"
00010 #else
00011     #include <ESP8266WiFi.h>
00012     #include <ESP8266WebServer.h>
00013     #include <WiFiUdp.h>
00014 #endif
00015
00016 #include <vector>
00017 #include "../Models/GloveModel.h"
00018 #include "../ActuatorTypes/VibrationActuator.h"
00019 #include "../Mapper/ActuatorProcessingOrderMapper.h"
00020 #include "../Mapper/BrailleMapper.h"
00021 #include "../Models/HandEnum.h"
00022 #include "../Master/WifiMaster.h"
00023 #include "../Slave/WifiSlave.h"
00024
00032 class Controller {
00033 public:
00041     Controller(bool isSlave);
00042
00049     void setup();
00050
00056     void loop();
00057
00058 private:
00059     bool isSlave;
00060     WifiMaster* master;
00061     WifiSlave* slave;
00062
00068     void initializeMaster();
00069
00075     void initializeSlave();
00076 };
00077
00078
00079 #endif // CONTROLLER_H
```

5.7 ActuatorProcessingOrderMapper.h

```
00001 #ifndef ACTUATOR_PROCESSING_ORDER_MAPPER_H
00002 #define ACTUATOR_PROCESSING_ORDER_MAPPER_H
00003
00004 #ifdef UNIT_TEST
00008     class ActuatorProcessingOrderMapperTestHelper;
00009 #endif
00010
00011 #include <unordered_map>
00012 #include <vector>
00013
00021 class ActuatorProcessingOrderMapper {
00022 private:
00029     std::unordered_map<int, int> SENSITIVITY_ORDER;
00030
00037     void initializeSensitivityOrder();
00038
00048     int reorderBySensitivity(int number);
00049
00050 public:
```

```
00056     ActuatorProcessingOrderMapper();
00057
00067     std::vector<int> reorderVectorBySensitivity(const std::vector<int>& values);
00068
00069 #ifdef UNIT_TEST
00073     friend class ActuatorProcessingOrderMapperTestHelper;
00074 #endif
00075 };
00076
00077 #endif // ACTUATOR_PROCESSING_ORDER_MAPPER_H
```

5.8 BrailleMapper.h

```
00001 #ifndef BRAILLEMAPPER_H
00002 #define BRAILLEMAPPER_H
00003
00004 #ifdef UNIT_TEST
00005     #include "../test/Mocks/String_Mock.h"
00009     class BrailleMapperTestHelper;
00010 #else
00011     #include <Arduino.h>
00012 #endif
00013
00014 #include <unordered_map>
00015 #include <vector>
00016
00024 class BrailleMapper {
00025 private:
00031     std::unordered_map<char, int> brailleMap;
00032
00038     void initializeBrailleMap();
00039
00040 public:
00044     BrailleMapper();
00045
00052     int getBrailleHash(char letter) const;
00053
00063     std::vector<int> stringToIntegerList(const String& input) const;
00064
00065 #ifdef UNIT_TEST
00069     friend class BrailleMapperTestHelper;
00070 #endif
00071 };
00072
00073 #endif // BRAILLEMAPPER_H
```

5.9 WifiMaster.h

```
00001 #ifndef WIFI_MASTER_H
00002 #define WIFI_MASTER_H
00003
00004 #ifdef UNIT_TEST
00005     #ifndef ARDUINO_MOCK_H
00006         #pragma once
00007         #include "../test/Mocks/new_Arduino_Mock.h"
00008     #endif
00009
00010     #include "../test/Mocks/ESP8266WiFi_Mock.h"
00011     #include "../test/Mocks/MockWiFiUDP.h"
00012     #include "../test/Mocks/LittleFS_Mock.h"
00013     #include "../test/Mocks/ESPNOW_Mock.h"
00014     #include "../test/Mocks/ESP_Mock.h"
00015
00016     extern LittleFSMock LittleFS;
00017     #define File MockFile
00018     extern MockWiFi WiFi;
00019     extern MockWifiEspNow WifiEspNow;
00020
00021 #else
00022     #include <ESP8266WiFi.h>
00023     #include <ESP8266WebServer.h>
00024     #include <WiFiUdp.h>
00025     #include <WiFiServer.h>
00026     #include <LittleFS.h>
00027     #include <WifiEspNow.h>
00028 #endif
00029
00030 #include <vector>
```

```
00031 #include <cstring>
00032
00033 #include "Mapper/BrailleMapper.h"
00034 #include "Mapper/ActuatorProcessingOrderMapper.h"
00035 #include "Models/GloveModel.h"
00036 #include "Models/EncodingScheme/ChordingScheme.h"
00037 #include "Models/HandEnum.h"
00038 #include "../Settings/SingeltonWifiSettings.h"
00039
00045 class WifiMaster {
00046 public:
00052     WifiMaster(GloveModel gloveModel);
00053
00059     void setup();
00060
00066     void loop();
00067
00075     void sendVectorToSlave(const std::vector<int> &reorderedValues, const ChordingScheme status, int
      repeat);
00076
00083     void sendVectorToSlave(const std::vector<int> &reorderedValues, const ChordingScheme status);
00084
00085 private:
00086     int idx;
00087     String pattern;
00088     ESP8266WebServer server;
00089
00090     BrailleMapper brailleMapper = BrailleMapper();
00091     ActuatorProcessingOrderMapper queue = ActuatorProcessingOrderMapper();
00092     GloveModel gloveModel;
00093
00097     void setupWifi();
00098
00102     void setupESPNow();
00103
00109     void sendVectorToSlave(std::vector<int> reorderedValues);
00110
00116     void sendIntegerToSlave(int singleValueToSend);
00117
00121     void setFrontend();
00122
00130     void frontendSetPattern(String pattern, ChordingScheme status, bool longPattern);
00131
00138     void frontendSetPattern(String pattern, ChordingScheme status);
00139
00147     void computePatternAndDistribute(String text, ChordingScheme status, bool longPattern);
00148
00155     std::vector<int> computePatternFromText(String text);
00156
00162     void distributePatternToGloves(std::vector<int> pattern);
00163
00167     void frontendAjaxCall();
00168
00174     void customDelay(unsigned long timeInMs);
00175 };
00176
00177 #endif // WIFI_MASTER_H
```

5.10 ChordingScheme.h

```
00001 #ifndef CHORDING_SCHEME_H
00002 #define CHORDING_SCHEME_H
00003
00011 enum ChordingScheme {
00012     OST_ENCODING,
00013     SEQUENTIAL_ENCODING
00014 };
00015
00016 #endif
```

5.11 Encoding.h

```
00001 #ifndef ENCODING_H
00002 #define ENCODING_H
00003
00004 #ifdef UNIT_TEST
00005     #ifndef ARDUINO_MOCK_H
00006         #pragma once
```

```
00007         #include "../test/Mocks/new_Arduino_Mock.h"
00008     #endif
00009 #else
00010     #include <Arduino.h>
00011 #endif
00012
00019 class Encoding {
00020 public:
00029     static void customDelay(unsigned long timeInMs) {
00030         unsigned long startMillis = millis();
00031         while (millis() - startMillis < timeInMs) {
00032             yield();
00033         }
00034     }
00035
00046     static bool validIndex(int number, Hand hand){
00047         if ((hand == Left && number > SingeltonGloveSettings::getInstance().NUM_ACTUATORS) ||
00048             (hand == Right && number < SingeltonGloveSettings::getInstance().NUM_ACTUATORS + 1)) {
00049             return false;
00050         } else {
00051             return true;
00052         }
00053     }
00054 };
00055
00056 #endif
```

5.12 OSTEncoding.h

```
00001 #ifndef OST_ENCODING_H
00002 #define OST_ENCODING_H
00003
00004 #include "../../ActuatorTypes/Actuator.h"
00005 #include "../../Settings/SingeltonGloveSettings.h"
00006 #include "../../Models/HandEnum.h"
00007 #include "Encoding.h"
00008
00015 class OSTEncoding : public Encoding {
00016 public:
00028     static void handle(int number, Actuator** actuators, Hand hand) {
00029         if (validIndex(number, hand)) {
00030             int actuatorIdx = (number - 1) % SingeltonGloveSettings::getInstance().NUM_ACTUATORS;
00031             actuators[actuatorIdx]->activate();
00032         }
00033         customDelay(SingeltonGloveSettings::getInstance().OST_OFFSET);
00034     }
00035 };
00036
00037 #endif
```

5.13 SequentialEncoding.h

```
00001 #ifndef SEQUENTIAL_ENCODING_H
00002 #define SEQUENTIAL_ENCODING_H
00003
00004 #include "../../ActuatorTypes/Actuator.h"
00005 #include "../../Settings/SingeltonGloveSettings.h"
00006 #include "../../Models/HandEnum.h"
00007 #include "Encoding.h"
00008
00015 class SequentialEncoding : public Encoding {
00016 public:
00028     static void handle(int number, Actuator** actuators, Hand hand) {
00029         if (validIndex(number, hand)) {
00030             int actuatorIdx = (number - 1) % SingeltonGloveSettings::getInstance().NUM_ACTUATORS;
00031             actuators[actuatorIdx]->activate();
00032             customDelay(SingeltonGloveSettings::getInstance().DURATION);
00033             actuators[actuatorIdx]->deactivate();
00034             customDelay(SingeltonGloveSettings::getInstance().SEQ_OFFSET);
00035         } else {
00036             customDelay(SingeltonGloveSettings::getInstance().DURATION);
00037             customDelay(SingeltonGloveSettings::getInstance().SEQ_OFFSET);
00038         }
00039     }
00040 };
00041
00042 #endif
```

5.14 GloveModel.h

```
00001 #ifndef GLOVE_MODEL_H
00002 #define GLOVE_MODEL_H
00003
00004 #ifdef UNIT_TEST
00005 #else
00006     #include <Arduino.h>
00007 #endif
00008
00009 #include <unordered_map>
00010 #include <vector>
00011 #include "../ActuatorTypes/Actuator.h"
00012 #include "../Models/HandEnum.h"
00013 #include "../Settings/SingeltonGloveSettings.h"
00014 #include "./EncodingScheme/ChordingScheme.h"
00015 #include "./EncodingScheme/SequentialEncoding.h"
00016 #include "./EncodingScheme/OSTEncoding.h"
00017
00026 class GloveModel {
00027 private:
00028     Actuator* actuators[3];
00029     Hand hand;
00030     std::vector<int> values;
00031     ChordingScheme playMode;
00032
00033 public:
00044     GloveModel(Hand hand, Actuator& actuator1, Actuator& actuator2, Actuator& actuator3) {
00045         actuators[0] = &actuator1;
00046         actuators[1] = &actuator2;
00047         actuators[2] = &actuator3;
00048         this->hand = hand;
00049     }
00050
00056     void resetAllActuators() {
00057         for (int i = 0; i < SingeltonGloveSettings::getInstance().NUM_ACTUATORS; i++) {
00058             if (actuators[i] != nullptr) {
00059                 actuators[i]->deactivate();
00060             }
00061         }
00062     }
00063
00071     void executePatternAt(int index) {
00072         resetAllActuators();
00073         activateOnNumber(values[index]);
00074     }
00075
00081     void pauseBetweenLetters(){
00082         SequentialEncoding::customDelay(SingeltonGloveSettings::getInstance().DURATION);
00083         resetAllActuators();
00084         SequentialEncoding::customDelay(SingeltonGloveSettings::getInstance().PAUSE);
00085     }
00086
00095     void activateOnNumber(int number) {
00096         if (number < 1) {   // -1 is a pause, so reset every actuator
00097             pauseBetweenLetters();
00098             return;
00099         }
00100         while (number > 0) {
00101             int lastDigit = number % 10;
00102             number = (int)number / 10;
00103
00104             if (playMode == SEQUENTIAL_ENCODING) {
00105                 SequentialEncoding::handle(lastDigit, actuators, hand);
00106             } else {
00107                 OSTEncoding::handle(lastDigit, actuators, hand);
00108             }
00109         }
00110         if (playMode == OST_ENCODING) {
00111             SequentialEncoding::customDelay(SingeltonGloveSettings::getInstance().DURATION);
00112             resetAllActuators();
00113         }
00114         SequentialEncoding::customDelay(SingeltonGloveSettings::getInstance().PAUSE);
00115     }
00116
00124     void setPattern(std::vector<int> newValues) {
00125         values = newValues;
00126     }
00127
00135     std::vector<int> getPattern() {
00136         return values;
00137     }
00138
00144     int getPatternLength() {
00145         return values.size();
00146     }
```

```
00147
00155     void setChordMode(ChordingScheme chordMode) {
00156         playMode = chordMode;
00157     }
00158 };
00159
00160 #endif
```

5.15 HandEnum.h

```
00001 #ifndef HAND_ENUM_H
00002 #define HAND_ENUM_H
00003
00010 enum Hand {
00011     Left,
00012     Right
00013 };
00014
00015 #endif
```

5.16 SingeltonGloveSettings.h

```
00001 #ifndef SINGELTON_GLOVE_SETTINGS
00002 #define SINGELTON_GLOVE_SETTINGS
00003
00012 class SingeltonGloveSettings {
00013 private:
00019     SingeltonGloveSettings() {}
00020
00024     SingeltonGloveSettings(const SingeltonGloveSettings&) = delete;
00025
00029     void operator=(const SingeltonGloveSettings&) = delete;
00030
00031 public:
00036     static SingeltonGloveSettings& getInstance() {
00037         static SingeltonGloveSettings instance;
00038         return instance;
00039     }
00040
00042     const int OST_OFFSET = 10;
00043
00045     const int DURATION = 200;
00046
00048     const int PAUSE = 2000;
00049
00051     const int NUM_ACTUATORS = 3;
00052
00054     const int AUDIO_STIMULI_OFFSET = 100;
00055
00057     const int SEQ_OFFSET = 1000;
00058
00060     const int studyOstRepititions = 126;
00061
00063     const int studySeqRepititions = 44;
00064 };
00065
00066 #endif
```

5.17 SingeltonWifiSettings.h

```
00001 #ifndef SINGELTON_WIFI_SETTINGS
00002 #define SINGELTON_WIFI_SETTINGS
00003
00004 #include <cstdint>
00005
00014 class SingeltonWifiConnector {
00015 private:
00021     SingeltonWifiConnector() {}
00022
00026     SingeltonWifiConnector(const SingeltonWifiConnector&) = delete;
00027
00031     void operator=(const SingeltonWifiConnector&) = delete;
00032
```

```
00033 public:
00038     static SingeltonWifiConnector& getInstance() {
00039         static SingeltonWifiConnector instance;
00040         return instance;
00041     }
00042 
00044     const char* MASTER_SSID = "MV-Glove";
00045 
00047     const char* SLAVE_SSID = "VS-Glove";
00048 
00054     const uint8_t SLAVE_MAC[6] = {0x48, 0x55, 0x19, 0xF6, 0xC9, 0xB3};
00055 };
00056 
00057 #endif
```

5.18 WifiSlave.h

```
00001 #ifndef WIFI_SLAVE_H
00002 #define WIFI_SLAVE_H
00003 
00004 #ifdef UNIT_TEST
00005     #include "../test/Mocks/ESP8266WiFi_Mock.h"
00006     #include "../test/Mocks/MockWiFiUDP.h"
00007     #include "../test/Mocks/ESPNOW_Mock.h"
00008     #include "../test/Mocks/ESP_Mock.h"
00009 #else
00010     #include <ESP8266WiFi.h>
00011     #include <ESP8266WebServer.h>
00012     #include <WiFiUdp.h>
00013     #include <WiFiServer.h>
00014     #include <LittleFS.h>
00015     #include <WifiEspNow.h>
00016 #endif
00017 
00018 #include <vector>
00019 #include "Models/GloveModel.h"
00020 
00026 class WifiSlave {
00027 public:
00033     WifiSlave(GloveModel gloveModel);
00034 
00040     void setup();
00041 
00047     void loop();
00048 
00060     static void onReceiveCallback(const uint8_t* mac, const uint8_t* buf, size_t count, void* arg);
00061 
00072     void processMessage(const uint8_t* mac, const uint8_t* buf, size_t count);
00073 
00074 private:
00075     GloveModel gloveModel;
00076     bool hasPatternFlag = false;
00077     bool nextCharacterFlag = false;
00078     int characterIndex = 0;
00079 
00085     void runProgram();
00086 
00092     void receivedIndex(int index);
00093 
00099     void receivedPatten(std::vector<int> sensitivityPattern);
00100 };
00101 
00102 #endif // WIFI_SLAVE_H
```

Index