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

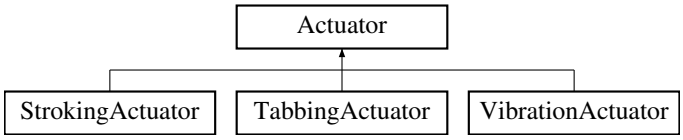
Class Documentation

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

Public Member Functions

- std::vector< int > **reorderVectorBySensitivity** (const std::vector< int > &values)

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

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

4.3 BrailleMapper Class Reference

Public Member Functions

- int **getBrailleHash** (char letter) const
- std::vector< int > **stringToIntegerList** (const String &input) const

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

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

4.4 Controller Class Reference

Public Member Functions

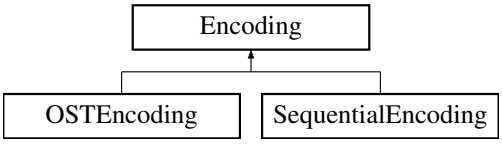
- **Controller** (bool isSlave)
- void **setup** ()
- void **loop** ()

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

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

4.5 Encoding Class Reference

Inheritance diagram for Encoding:



Static Public Member Functions

- static void **customDelay** (unsigned long timeInMs)
- static bool **validIndex** (int number, Hand hand)

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

- src/Models/EncodingScheme/Encoding.h

4.6 GloveModel Class Reference

Public Member Functions

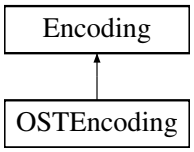
- **GloveModel** (Hand hand, Actuator &actuator1, Actuator &actuator2, Actuator &actuator3)
- void **resetAllActuators** ()
- void **executePatternAt** (int index)
- void **pauseBetweenLetters** ()
- void **vibrateOnNumber** (int number)
- void **setPattern** (std::vector< int > newValues)
- std::vector< int > **getPattern** ()
- int **getPatternLength** ()
- void **setChordMode** (ChordingScheme chordMode)

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

- src/Models/GloveModel.h

4.7 OSTEncoding Class Reference

Inheritance diagram for OSTEncoding:



Static Public Member Functions

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

Static Public Member Functions inherited from Encoding

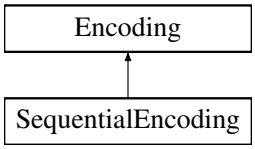
- static void **customDelay** (unsigned long timeInMs)
- static bool **validIndex** (int number, Hand hand)

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

- src/Models/EncodingScheme/OSTEncoding.h

4.8 SequentialEncoding Class Reference

Inheritance diagram for SequentialEncoding:



Static Public Member Functions

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

Static Public Member Functions inherited from Encoding

- static void **customDelay** (unsigned long timeInMs)
- static bool **validIndex** (int number, Hand hand)

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

- src/Models/EncodingScheme/SequentialEncoding.h

4.9 SingeltonGloveSettings Class Reference

Static Public Member Functions

- static SingeltonGloveSettings & **getInstance** ()

Public Attributes

- const int **OST_OFFSET** = 10
- const int **DURATION** = 200
- const int **PAUSE** = 2000
- const int **NUM_ACTUATORS** = 3
- const int **AUDIO_VIBRATION_OFFSET** = 100
- const int **SEQ_OFFSET** = 1000
- const int **studyOstRepititions** = 126
- const int **studySeqRepititions** = 44

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

- src/Settings/SingeltonGloveSettings.h

4.10 SingeltonWifiConnector Class Reference

Static Public Member Functions

- static SingeltonWifiConnector & **getInstance** ()

Public Attributes

- const char * **MASTER_SSID** = "MV-Glove"
- const char * **SLAVE_SSID** = "VS-Glove"
- const uint8_t **SLAVE_MAC** [6] = {0x48, 0x55, 0x19, 0xF6, 0xC9, 0xB3}

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

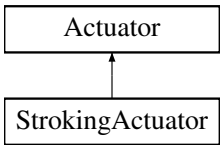
- src/Settings/SingeltonWifiSettings.h

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.

< Attach the servo to the specified pin.

< Set servo to 0 degrees initially.

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. < Move the servo to 180 degrees.

Implements Actuator.

4.11.3.2 deactivate()

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

Deactivates the stroking actuator.

Moves the servo back to 0 degrees. < Move the servo back to 0 degrees.

Implements Actuator.

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

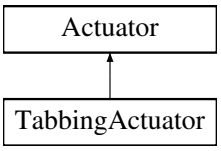
- src/ActuatorTypes/StrokingActuator.h

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.

< Attach the servo to the specified pin.

< Set servo to 180 degrees initially.

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. < Move the servo to 90 degrees.

Implements Actuator.

4.12.3.2 deactivate()

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

Deactivates the tabbing actuator.

Moves the servo back to 180 degrees. < Move the servo back to 180 degrees.

Implements Actuator.

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

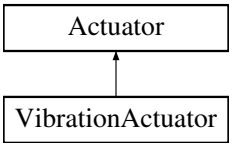
- src/ActuatorTypes/TabbingActuator.h

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.

< Set the pin as an output.

< Ensure the actuator is off initially.

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. < Turn on vibration.

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. < Turn off vibration.

Implements Actuator.

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

- src/ActuatorTypes/VibrationActuator.h

4.14 WifiMaster Class Reference

Public Member Functions

- **WifiMaster** (GloveModel gloveModel)
- void **setup** ()
- void **loop** ()
- void **sendVectorToSlave** (const std::vector< int > &reorderedValues, const ChordingScheme status, int repeat)
- void **sendVectorToSlave** (const std::vector< int > &reorderedValues, const ChordingScheme status)

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

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

4.15 WifiSlave Class Reference

Public Member Functions

- **WifiSlave** (GloveModel gloveModel)
- void **setup** ()
- void **loop** ()
- void **processMessage** (const uint8_t ∗mac, const uint8_t ∗buf, size_t count)

Static Public Member Functions

- static void **onReceiveCallback** (const uint8_t ∗mac, const uint8_t ∗buf, size_t count, void ∗arg)

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
00009     #include "../test/Mocks/ESPNOW_Mock.h"
00010     #include "../test/Mocks/ESP_Mock.h"
00011 #else
00012     #include <ESP8266WiFi.h>
00013     #include <ESP8266WebServer.h>
00014     #include <WiFiUdp.h>
00015 #endif
00016
00017 #include <vector>
00018
00019 #include "../Models/GloveModel.h"
00020 #include "../ActuatorTypes/VibrationActuator.h"
00021 #include "../Mapper/ActuatorProcessingOrderMapper.h"
00022 #include "../Mapper/BrailleMapper.h"
00023 #include "../Models/HandEnum.h"
00024
00025 #include "../Master/WifiMaster.h"
00026 #include "../Slave/WifiSlave.h"
00027
00028
00029
00030
00031
00032
00033 class Controller {
00034 public:
00035     Controller(bool isSlave);
00036     void setup();
00037     void loop();
00038
00039 private:
00040     bool isSlave;
00041     WifiMaster* master;
00042     WifiSlave* slave;
00043
00044     void initializeMaster();
00045     void initializeSlave();
00046 };
00047
00048 #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
00005     // Only define the friend class relationship during unit testing
00006     class ActuatorProcessingOrderMapperTestHelper;
00007 #endif
00008
00009 #include <unordered_map>
00010 #include <vector>
00011
00012 class ActuatorProcessingOrderMapper {
00013   private:
00014     // Sensitivity order for the actuators, from most to least sensitive based on the braille number
00015     std::unordered_map<int, int> SENSITIVITY_ORDER;
```

```
00016
00017     void initializeSensitivityOrder();
00018
00019     int reorderBySensitivity(int number);
00020
00021   public:
00022     ActuatorProcessingOrderMapper();
00023
00024     std::vector<int> reorderVectorBySensitivity(const std::vector<int>& values);
00025
00026 #ifdef UNIT_TEST
00027     // Define the friend class only if UNIT_TEST is defined
00028     friend class ActuatorProcessingOrderMapperTestHelper;
00029 #endif
00030 };
00031
00032 #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"
00006     // Forward declare the test helper class
00007     class BrailleMapperTestHelper;
00008 #else
00009     #include <Arduino.h>
00010 #endif
00011
00012 #include <unordered_map>
00013 #include <vector>
00014
00015 class BrailleMapper {
00016 private:
00017     std::unordered_map<char, int> brailleMap; // HashMap to store mappings
00018
00019     void initializeBrailleMap(); // Initializes the braille mappings
00020
00021 public:
00022     BrailleMapper(); // Constructor to initialize the mappings
00023
00024     int getBrailleHash(char letter) const; // Retrieve Braille hash for a letter
00025     std::vector<int> stringToIntegerList(const String& input) const; // Convert string to a list of
      Braille integers
00026
00027 #ifdef UNIT_TEST
00028     // Grant access to BrailleMapperTestHelper during unit tests
00029     friend class BrailleMapperTestHelper;
00030 #endif
00031 };
00032
00033 #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>  // Include for TCP server
00026     #include <LittleFS.h>
00027     #include <WifiEspNow.h>
00028
00029 #endif
00030
00031 #include <vector>
00032
00033 #include <cstring>
00034
00035 #include "Mapper/BrailleMapper.h"
00036 #include "Mapper/ActuatorProcessingOrderMapper.h"
00037 #include "Models/GloveModel.h"
00038 #include "Models/EncodingScheme/ChordingScheme.h"
00039 #include "Models/HandEnum.h"
00040 #include "../Settings/SingeltonWifiSettings.h"
00041
00042 class WifiMaster {
00043 public:
00044     WifiMaster(GloveModel gloveModel);
00045     void setup();
00046     void loop();
00047
00048     void sendVectorToSlave(const std::vector<int> &reorderedValues, const ChordingScheme status, int
      repeat);
00049
00050     void sendVectorToSlave(const std::vector<int> &reorderedValues, const ChordingScheme status);
00051
00052 private:
00053     int idx;
00054     String pattern;
00055     ESP8266WebServer server;
00056
00057     // View view;
00058     BrailleMapper brailleMapper = BrailleMapper();
00059     ActuatorProcessingOrderMapper queue = ActuatorProcessingOrderMapper();
00060     GloveModel gloveModel;
00061     // DataSender dataSender;
00062
00063     void sendVectorToSlave(std::vector<int> reorderedValues);
00064     void sendIntegerToSlave(int singleValueToSend);
00065     void setFrontend();
00066     void frontendSetPattern(String pattern, ChordingScheme status, bool longPattern);
00067     void printConnectedDevices();
00068     void frontendSetPattern(String pattern, ChordingScheme status);
00069     void computePatternAndDistribute(String text, ChordingScheme status, bool longPattern);
00070     std::vector<int> computePatternFromText(String text);
00071     void distributePatternToGloves(std::vector<int> pattern);
00072     void startFunction();
00073     void frontendAjaxCall();
00074
00075     void customDelay(unsigned long timeInMs){ //this is needed for wifi compatability
00076       unsigned long startMillis = millis();  // Get the current time
00077       while (millis() - startMillis < timeInMs) {
00078           yield();  //the programm doesn't stop
00079       }
00080     }
00081 };
00082 #endif // WIFI_MASTER_H
```

5.10 ChordingScheme.h

```
00001 #ifndef CHORDING_SCHEME_H
00002 #define CHORDING_SCHEME_H
00003
00004
00005 enum ChordingScheme {
00006     OST_ENCODING,
00007     SEQUENTIAL_ENCODING
00008 };
00009
00010 #endif
```

5.11 Encoding.h

```
00001 #ifndef ENCODING_H
```

```
00002 #define ENCODING_H
00003
00004
00005 #ifdef UNIT_TEST
00006     #ifndef ARDUINO_MOCK_H
00007         #pragma once
00008         #include "../test/Mocks/new_Arduino_Mock.h"
00009     #endif
00010 #else
00011     #include <Arduino.h>
00012 #endif
00013
00014 class Encoding {
00015 public:
00016     static void customDelay(unsigned long timeInMs) {
00017         unsigned long startMillis = millis();  // Get the current time
00018         while (millis() - startMillis < timeInMs) {
00019             yield();  // the program doesn't stop
00020         }
00021     }
00022
00023     static bool validIndex(int number, Hand hand){
00024         if ((hand == Left && number > SingeltonGloveSettings::getInstance().NUM_ACTUATORS) ||
00025             (hand == Right && number < SingeltonGloveSettings::getInstance().NUM_ACTUATORS + 1)) {
00026             return false;
00027         }else{
00028             return true;
00029         }
00030
00031     }
00032 };
00033 #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
00009 class OSTEncoding : public Encoding {
00010 public:
00011     static void handle(int number, Actuator** actuators, Hand hand) {
00012         if (validIndex(number, hand)) {
00013             int actuatorIdx = (number - 1) % SingeltonGloveSettings::getInstance().NUM_ACTUATORS; // 3
      actuators on each hand
00014             actuators[actuatorIdx]->activate();
00015         }
00016         customDelay(SingeltonGloveSettings::getInstance().OST_OFFSET);
00017     }
00018 };
00019
00020 #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
00009 class SequentialEncoding : public Encoding {
00010 public:
00011     static void handle(int number, Actuator** actuators, Hand hand) {
00012         if (validIndex(number, hand)) {
00013             int actuatorIdx = (number - 1) % SingeltonGloveSettings::getInstance().NUM_ACTUATORS; // 3
      actuators on each hand
00014             actuators[actuatorIdx]->activate();
00015             customDelay(SingeltonGloveSettings::getInstance().DURATION);
00016             actuators[actuatorIdx]->deactivate();
00017             customDelay(SingeltonGloveSettings::getInstance().SEQ_OFFSET);
00018         }else{
```

```
00019             customDelay(SingeltonGloveSettings::getInstance().DURATION);
00020             customDelay(SingeltonGloveSettings::getInstance().SEQ_OFFSET);
00021         }
00022     }
00023 };
00024
00025 #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
00018 class GloveModel {
00019 private:
00020     Actuator* actuators[3];
00021     Hand hand;
00022     std::vector<int> values;
00023     ChordingScheme playMode;
00024
00025 public:
00026     GloveModel(Hand hand, Actuator& actuator1, Actuator& actuator2, Actuator& actuator3) {
00027         actuators[0] = &actuator1;
00028         actuators[1] = &actuator2;
00029         actuators[2] = &actuator3;
00030         this->hand = hand;
00031     }
00032
00033     void resetAllActuators() {
00034         for (int i = 0; i < SingeltonGloveSettings::getInstance().NUM_ACTUATORS; i++) {
00035             if (actuators[i] != nullptr) {
00036                 actuators[i]->deactivate();
00037             }
00038         }
00039     }
00040
00041     void executePatternAt(int index) {
00042         resetAllActuators();
00043         vibrateOnNumber(values[index]);
00044     }
00045
00046     void pauseBetweenLetters(){
00047         SequentialEncoding::customDelay(SingeltonGloveSettings::getInstance().DURATION);
00048         resetAllActuators();
00049         SequentialEncoding::customDelay(SingeltonGloveSettings::getInstance().PAUSE);
00050     }
00051
00052     void vibrateOnNumber(int number) {
00053         if (number < 1) {   // -1 is a pause, so reset every actuator
00054             pauseBetweenLetters();
00055             return;
00056         }
00057         while (number > 0) {
00058             int lastDigit = number % 10;
00059             number = (int)number / 10;
00060
00061             if (playMode == SEQUENTIAL_ENCODING) {
00062                 SequentialEncoding::handle(lastDigit, actuators, hand);
00063             } else {
00064                 OSTEncoding::handle(lastDigit, actuators, hand);
00065             }
00066         }
00067         if(playMode == OST_ENCODING){
00068             SequentialEncoding::customDelay(SingeltonGloveSettings::getInstance().DURATION);
00069             resetAllActuators();
00070         }
00071         SequentialEncoding::customDelay(SingeltonGloveSettings::getInstance().PAUSE);
00072     }
00073
```

```
00074     void setPattern(std::vector<int> newValues) {
00075         values = newValues;
00076     }
00077
00078     std::vector<int> getPattern() {
00079         return values;
00080     }
00081
00082     int getPatternLength() {
00083         return values.size();
00084     }
00085
00086     void setChordMode(ChordingScheme chordMode) {
00087         playMode = chordMode;
00088     }
00089 };
00090
00091 #endif
```

5.15 HandEnum.h

```
00001 #ifndef HAND_ENUM_H
00002 #define HAND_ENUM_H
00003
00004 enum Hand {
00005     Left,
00006     Right
00007 };
00008
00009 #endif
```

5.16 SingeltonGloveSettings.h

```
00001 #ifndef SINGELTON_GLOVE_SETTINGS
00002 #define SINGELTON_GLOVE_SETTINGS
00003
00004
00005 class SingeltonGloveSettings {
00006     private:
00007         SingeltonGloveSettings() {}
00008         SingeltonGloveSettings(const SingeltonGloveSettings&) = delete;
00009         void operator=(const SingeltonGloveSettings&) = delete;
00010
00011     public:
00012         static SingeltonGloveSettings& getInstance() {
00013             static SingeltonGloveSettings instance;
00014             return instance;
00015         }
00016
00017         //TODO change settings accordingly
00018         const int OST_OFFSET = 10;
00019         const int DURATION = 200;
00020         const int PAUSE = 2000;
00021         const int NUM_ACTUATORS = 3;
00022         const int AUDIO_VIBRATION_OFFSET = 100;
00023
00024         const int SEQ_OFFSET = 1000;
00025         const int studyOstRepititions = 126;
00026         const int studySeqRepititions = 44;
00027
00028
00029 };
00030
00031 #endif
```

5.17 SingeltonWifiSettings.h

```
00001 #include <cstdint>
00002 #ifndef SINGELTON_WIFI_SETTINGS
00003 #define SINGELTON_WIFI_SETTINGS
00004
00005
00006 class SingeltonWifiConnector {
```

```
00007     private:
00008         SingeltonWifiConnector() {}
00009         SingeltonWifiConnector(const SingeltonWifiConnector&) = delete;
00010         void operator=(const SingeltonWifiConnector&) = delete;
00011 
00012     public:
00013         static SingeltonWifiConnector& getInstance() {
00014             static SingeltonWifiConnector instance;
00015             return instance;
00016         }
00017 
00018         const char* MASTER_SSID = "MV-Glove";
00019         const char* SLAVE_SSID = "VS-Glove";
00020         const uint8_t SLAVE_MAC[6] = {0x48, 0x55, 0x19, 0xF6, 0xC9, 0xB3}; //use the right slave mac
00021 };
00022 
00023 #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 
00010 #else
00011     #include <ESP8266WiFi.h>
00012     #include <ESP8266WebServer.h>
00013     #include <WiFiUdp.h>
00014     #include <WiFiServer.h>  // Include for TCP server
00015     #include <LittleFS.h>
00016     #include <WifiEspNow.h>
00017 
00018 #endif
00019 
00020 #include <vector>
00021 
00022 #include "Models/GloveModel.h"
00023 
00024 class WifiSlave {
00025 public:
00026     WifiSlave(GloveModel gloveModel);
00027     void setup();
00028     void loop();
00029     static void onReceiveCallback(const uint8_t* mac, const uint8_t* buf, size_t count, void* arg);
00030     void processMessage(const uint8_t* mac, const uint8_t* buf, size_t count);
00031 
00032 private:
00033     GloveModel gloveModel;
00034     bool hasPatternFlag = false;
00035     bool nextCharacterFlag = false;
00036     int characterIndex = 0;
00037 
00038     void runProgram();
00039     void receivedIndex(int index);
00040     void receivedPatten(std::vector<int> sensitivityPattern);
00041 };
00042 
00043 #endif // WIFI_SLAVE_H
```

Index