Min Paarvai

Technical Documentation

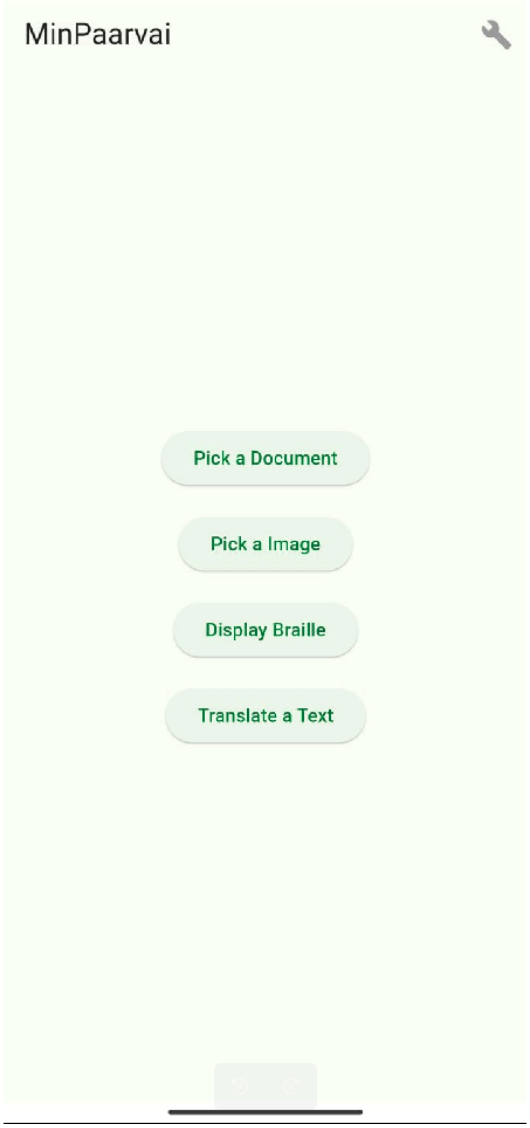
The Mobile App

Flutter based application and responsible for the followings:

- Image to Text OCR using tesseract.
- Translating Tamil to braille.
- Text speech using on-device api.
- Finally sends a character to be displayed to the api server.

System requirements: any modern smartphone

Home Screen:



1. When the top left wrench icon is pressed, it brings up a pop up with a text input field to set the api server ip address.
2. Pick a document button allows the user to pick a document and read its content into a string variable then pushes to the display braille screen. Click here for full source code.

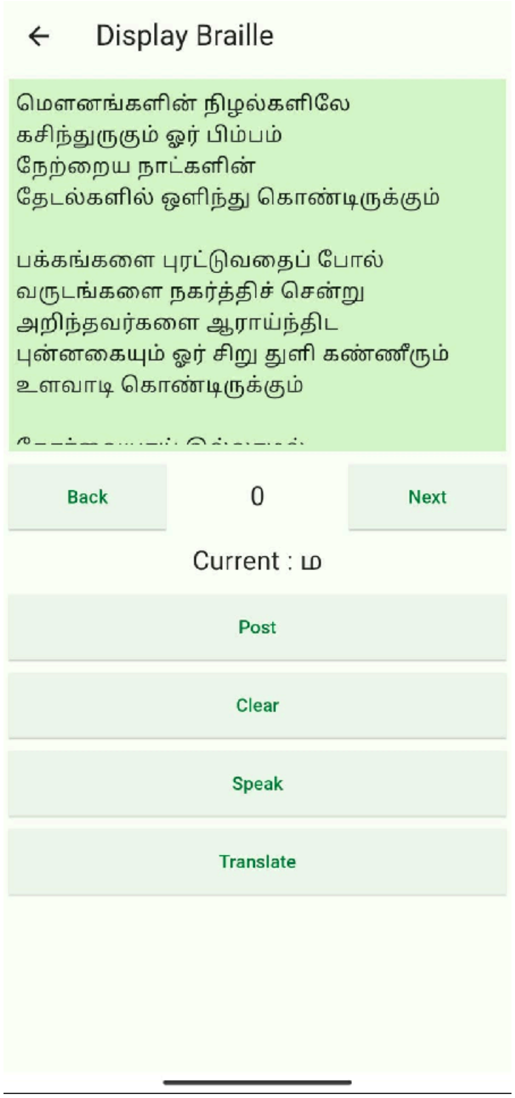
```
 List<String> exts = ['docx'];

 var result = await FilePicker.platform.pickFiles(

   type: FileType.custom,

   allowedExtensions: exts,

 );

final bytes = await file.readAsBytes();

final text = docxToText(bytes);
```

3. Pick a image button allows the user to pick an image and using ocr, reads its content into a string variable then pushes to the display braille screen. Click here for full source code.

```
  List<String> exts = ['jpg', 'jpeg', 'png'];

  var result = await FilePicker.platform.pickFiles(

    type: FileType.custom,

    allowedExtensions: exts,

  );

  String text = await FlutterTesseractOcr.extractText(path, language:
'tam+eng');
```

4. Display braille button pushes to the display braille screen.
5. Translate a text button pushed to the translator screen.

Display Braille Screen:



மௌனங்களின் நிழல்களிலே
கசிந்துருகும் ஓர் பிம்பம்
நேற்றைய நாட்களின்
தேடல்களில் ஒளிந்து கொண்டிருக்கும்

பக்கங்களை புரட்டுவதைப் போல்
வருடங்களை நகர்த்திச் சென்று
அறிந்தவர்களை ஆராய்ந்திட
புன்னகையும் ஓர் சிறு துளி கண்ணீரும்
உளவாடி கொண்டிருக்கும்

Current : ம

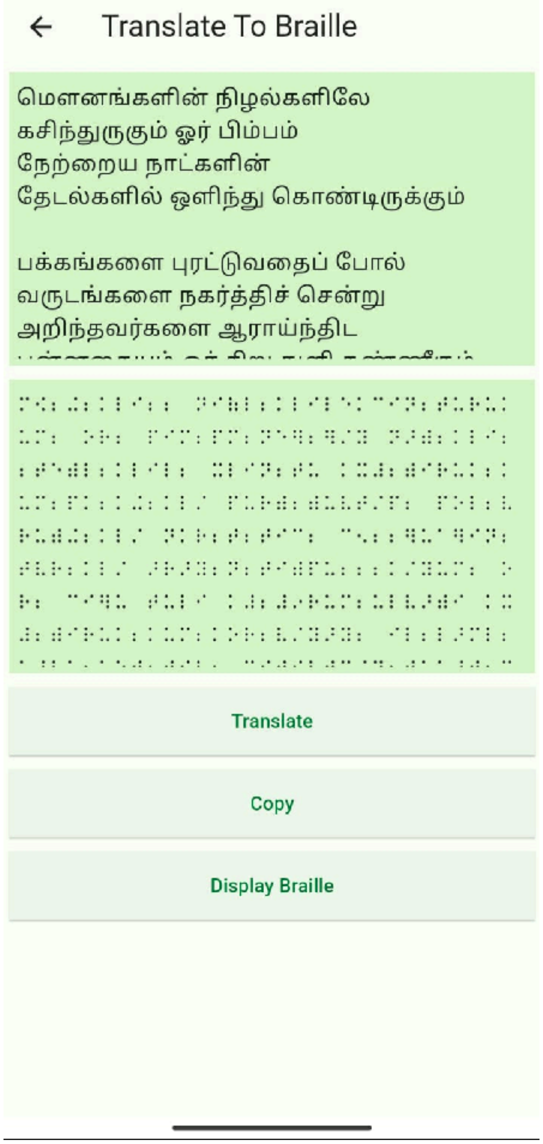
1. Giant green input field: If we came to this via pick a document / image button it will have contents of document / image. If we come to the screen with a display braille button then the input field will be empty.
2. The Back 0 Next row: Number in-between the buttons represents the current index of the character being displayed. Next is used to increment the index and back is used to decrement the index.
3. Post button: post the current character to the api server. Click here to view full source code.
4. Clear button: posts a empty character to api server to clear the braille display.

5. Speak button: Using native TTS api of the device the text is read out.

```
await flutterTts.speak(_textController.text);
```

6. Translate button: pushes to translator screen with content.

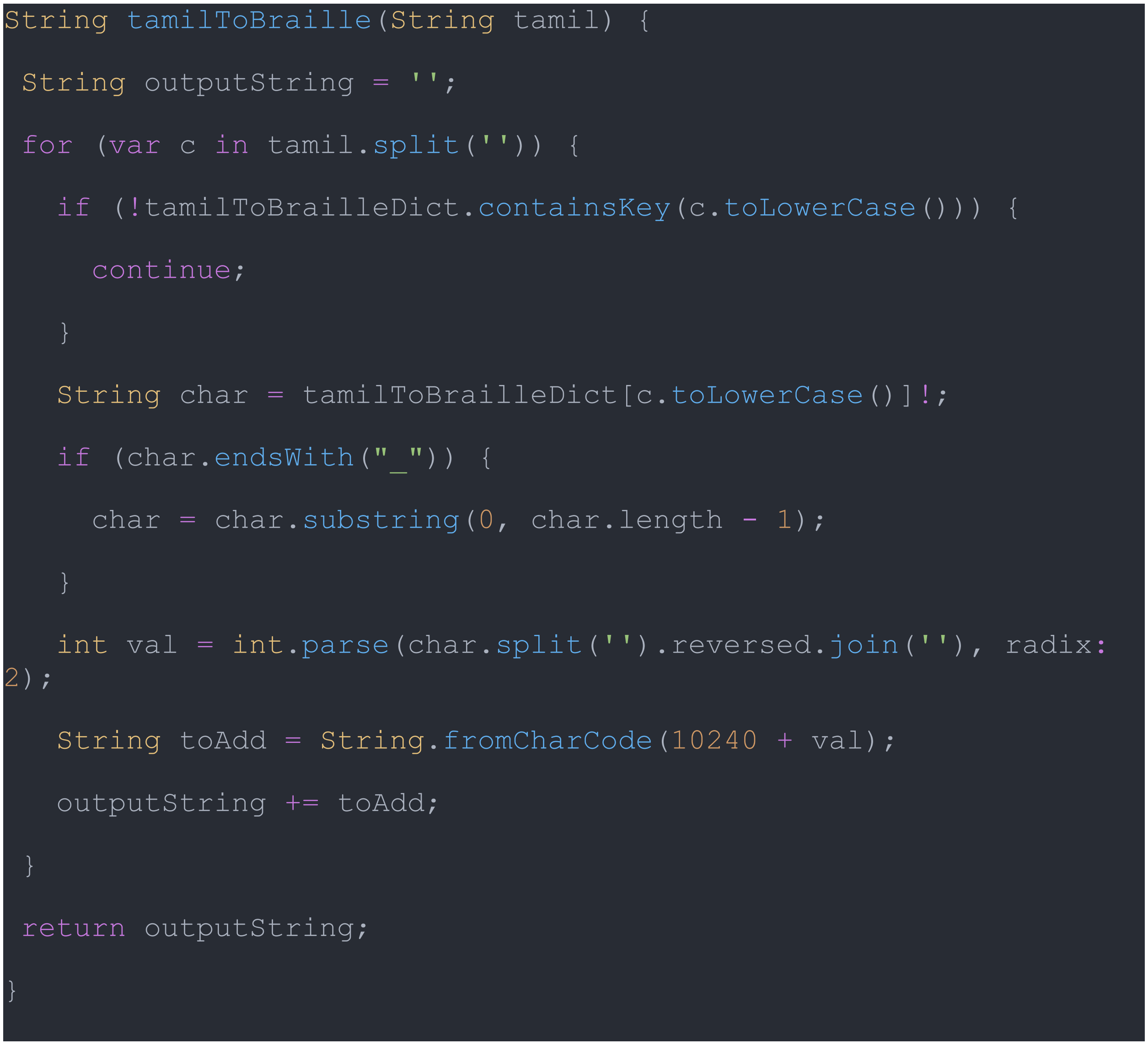
Translator screen:



மௌனங்களின் நிழல்களிலே
கசிந்துருகும் ஓர் பிம்பம்
நேற்றைய நாட்களின்
தேடல்களில் ஒளிந்து கொண்டிருக்கும்

பக்கங்களை புரட்டுவதைப் போல்
வருடங்களை நகர்த்திச் சென்று
அறிந்தவர்களை ஆராய்ந்திட

Translates the text in the input field to braille format using a dictionary. Click here for full source code.



```
String tamilToBraille(String tamil) {
 String outputString = '';
 for (var c in tamil.split('')) {
   if (!tamilToBrailleDict.containsKey(c.toLowerCase())) {
     continue;
   }
   String char = tamilToBrailleDict[c.toLowerCase()]!;
   if (char.endsWith("_")) {
     char = char.substring(0, char.length - 1);
   }
   int val = int.parse(char.split('').reversed.join(''), radix:
2);
   String toAdd = String.fromCharCode(10240 + val);
   outputString += toAdd;
 }
 return outputString;
}
```

API Server

Flash based rest api and responsible for the followings:

- Host an api endpoint for mobile apps to send characters to be displayed.
- Converts the received character into binary format where each bit represents braille dot.
- Sends the converted binary data to the MQTT server.

System Requirements:

- 1 core cpu or more
- 512mb ram or more
- 512mb hdd or more
- Any OS with python 3 installed.

Endpoint: `POST /display` click here for source code.

```
data = request.data.decode('utf-8')
character = data[0]
```

1. The request body is decoded as a UTF-8 encoded string.
2. First character is extracted from string.

```
to_translate = braille.tamilToBrailleDict.get(character, "000000")
```

3. The character is then converted into a binary string where 1 is braille dot.

```
val = int(to_translate, 2)
mqtt_client.publish(app.config['MQTT_TOPIC'], val.to_bytes(1, 'little', signed=False))
```

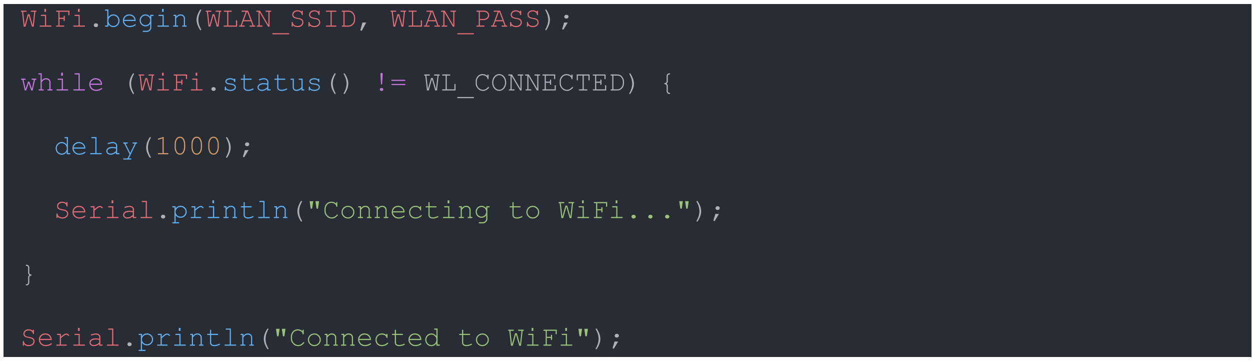
4. The binary string is converted to byte.
5. The byte is published to the MQTT server.

Braille Cell

Receives the binary data from MQTT server and based on each bit, pulls the pin down for 0 and pushes pin up for 1.

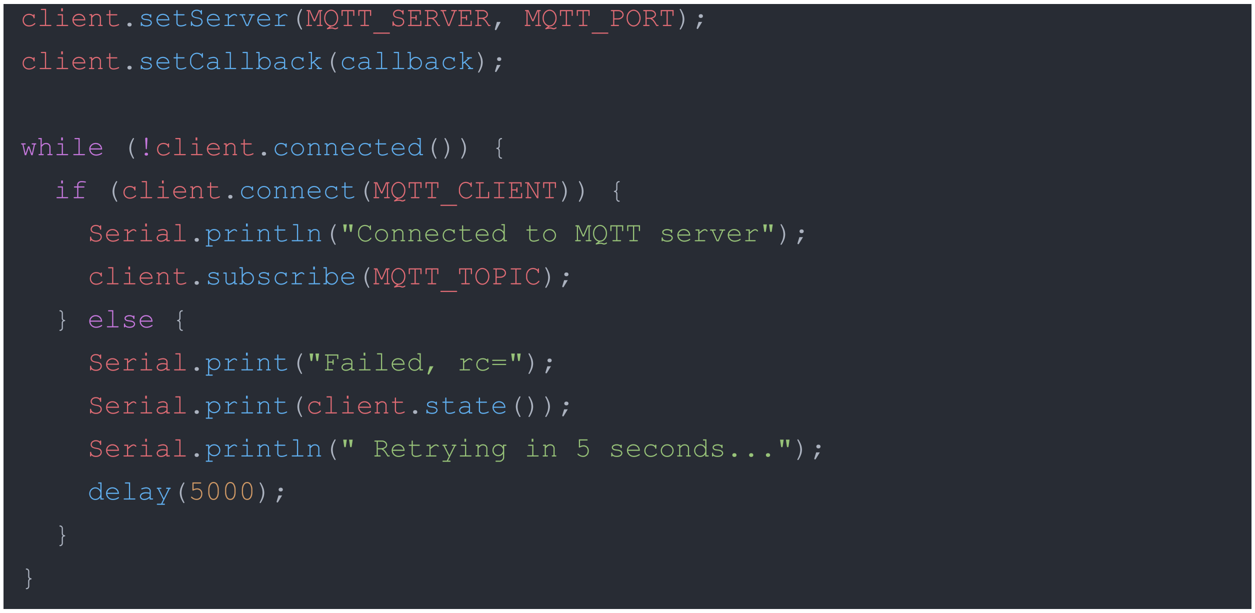
Components (for single character):

- 1x Esp32
- 3x DRV 8833
- 6x solenoids



```
WiFi.begin(WLAN_SSID, WLAN_PASS);

while (WiFi.status() != WL_CONNECTED) {

  delay(1000);

  Serial.println("Connecting to WiFi...");

}

Serial.println("Connected to WiFi");
```

1. Esp32 Repeatedly tries to connect to WIFI with 1 second delay in-between tries



```
client.setServer(MQTT_SERVER, MQTT_PORT);
client.setCallback(callback);

while (!client.connected()) {
  if (client.connect(MQTT_CLIENT)) {
    Serial.println("Connected to MQTT server");
    client.subscribe(MQTT_TOPIC);
  } else {
    Serial.print("Failed, rc=");
    Serial.print(client.state());
    Serial.println(" Retrying in 5 seconds...");
    delay(5000);
  }
}
```

2. Sets Server IP address and Port

3. Sets Callback function to handle received data.
4. Repeatedly tries to connect to Server with 5 second delay in-between tries

```
void callback(char* topic, byte* payload, unsigned int length) {
 byte b = payload[0];
 display(b);
}
```

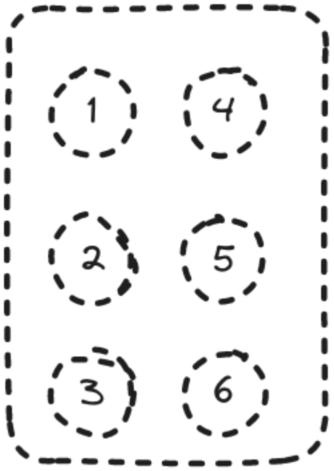
5. Extracts the first byte from received data and sends it to display function.

```
void display(byte b) {
 bool p1 = (b & 1) == 1;
 bool p2 = (b & 2) == 2;
 bool p3 = (b & 4) == 4;
 bool p4 = (b & 8) == 8;
 bool p5 = (b & 16) == 16;
 bool p6 = (b & 32) == 32;

 if (p1) {
   ToHigh(PIN1);
 } else {
   ToHigh(PIN1R);
 }

      ... full code in github ...

 if (p6) {
   ToHigh(PIN6);
 } else {
   ToHigh(PIN6R);
 }

 delay(500);
 CleanUp();
}
```

6. Using bit wise operation if bit is 1 then sets the pin to true.

7. If the pin is true the pushes the pin up and if the pin is false pulls the pin down.
8. Waits 500 milliseconds and cuts power. Freezing the pins in state.

Solenoids Layout:



Pin Configurations

Esp32 to Drv8833:

*Drv(n) used to represent corresponding Drv8833.

GPIO 23	Drv(1) IN 1
GPIO 22	Drv(1) IN 2
GPIO 21	Drv(1) IN 3
GPIO 19	Drv(1) IN 4
GPIO 18	Drv(2) IN 1
GPIO 5	Drv(2) IN 2
GPIO 17	Drv(2) IN 3
GPIO 16	Drv(2) IN 4
GPIO 4	Drv(3) IN 1
GPIO 2	Drv(3) IN 2
GPIO 27	Drv(3) IN 3

GPIO 26	Drv(3) IN 4

Drv8833 to Solenoids

*Drv(n) and Sol(n) used to represent corresponding Drv8833 and Solenoid.

Drv(1) OUT 1	Sol(1) Positive end
Drv(1) OUT 2	Sol(1) Negative end
Drv(1) OUT 3	Sol(4) Positive end
Drv(1) OUT 4	Sol(4) Negative end
Drv(2) OUT 1	Sol(2) Positive end
Drv(2) OUT 2	Sol(2) Negative end
Drv(2) OUT 3	Sol(5) Positive end
Drv(2) OUT 4	Sol(5) Negative end
Drv(3) OUT 1	Sol(3) Positive end
Drv(3) OUT 2	Sol(3) Negative end
Drv(3) OUT 3	Sol(6) Positive end
Drv(3) OUT 4	Sol(6) Negative end