LEBANESE UNIVERSITY

DEPARTMENT ELECTRICAL AND ELECTRONIC ENGINEERING

TELECOMMUNICATION

ROBOT PROJECT REPORT

Mini-Projet – CTE

NAME:ABDEL AZIZ MOHAMAD / TAREK ISMAIL

Proton 2.0: Advanced Voice-Controlled Robot

INTRODUCTION

Proton 2.0 is an advanced version of the original Proton robot, designed to be more interactive and user-friendly. By implementing voice recognition using the Arduino Nano 33 BLE Sense and Picovoice's open-source models, Proton 2.0 offers a natural and intuitive way of controlling the robot. This report details the design, components, implementation, and functionalities of Proton 2.0, highlighting its capabilities and potential applications.

Components and Hardware

Arduino Nano 33 BLE Sense

The Arduino Nano 33 BLE Sense is a powerful microcontroller that integrates several sensors and Bluetooth Low Energy (BLE) functionality.

It includes:

- Built-in Microphone: For voice recognition.
- Accelerometer and Gyroscope: For motion detection and orientation.
- Temperature, Humidity, and Pressure Sensors: For environmental sensing.
- Proximity and Gesture Sensor: For interaction detection.

The Nano 33 BLE Sense is ideal for projects requiring advanced sensing capabilities and wireless communication, making it the perfect choice for Proton 2.0.

Voice Recognition Implementation

Picovoice Models

We utilized Picovoice's open-source models to implement voice recognition in Proton 2.0. Picovoice provides two key models:

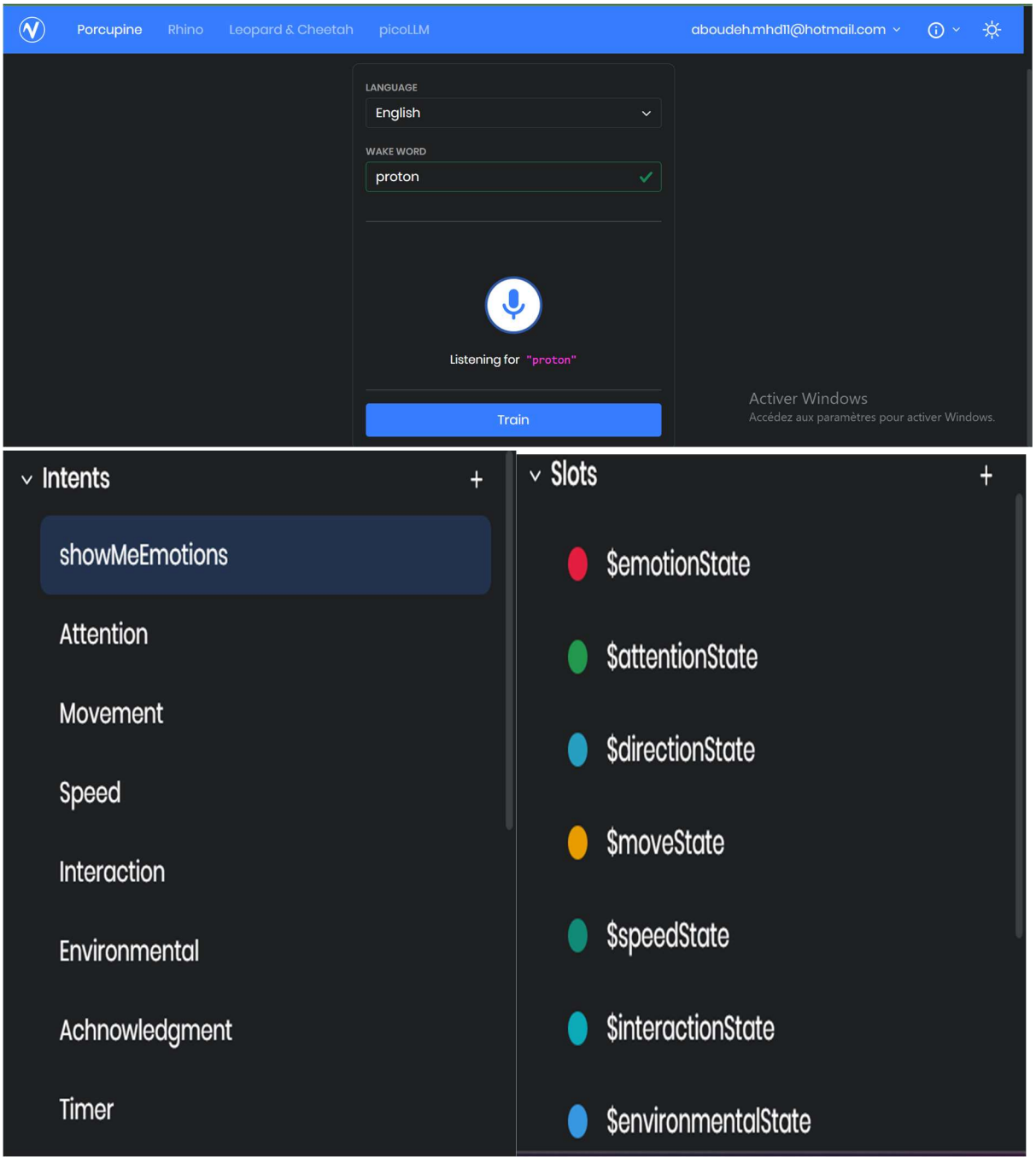
1. Porcupine: A wake word detection engine that listens for a specific wake word (e.g., "Proton") to activate the robot.
2. Rhino: A speech-to-intent engine that processes voice commands to determine user intent and execute corresponding actions.

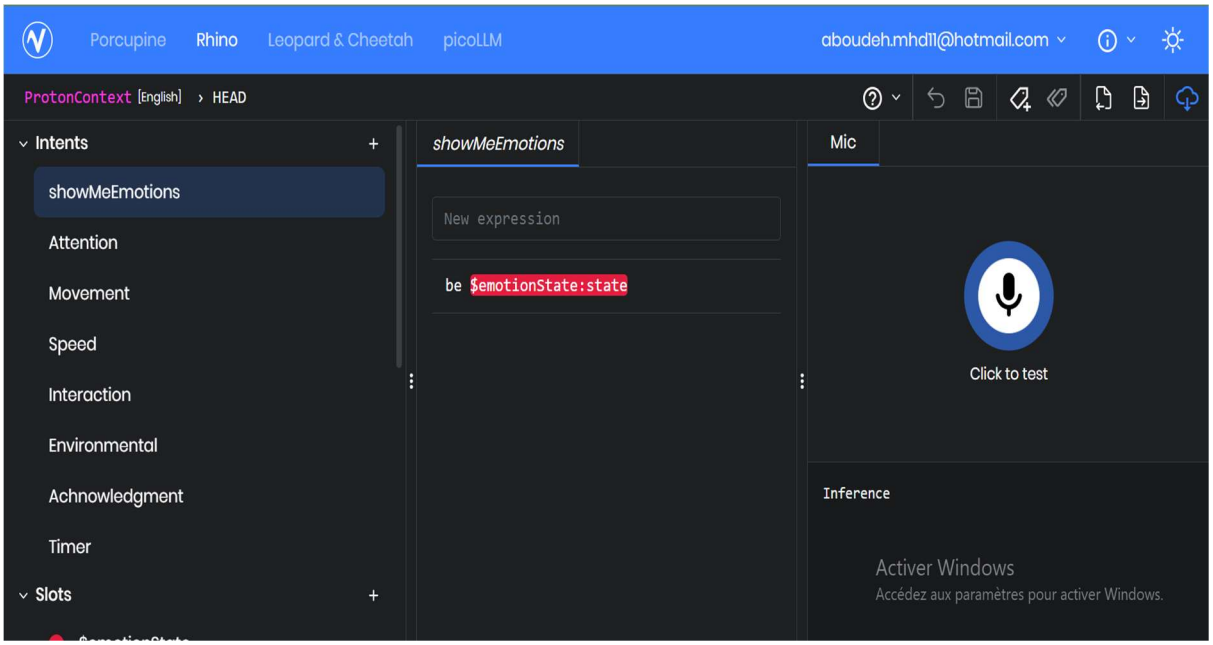
Wake Word Detection

The Porcupine model detects the wake word "Proton." Once the wake word is recognized, the Rhino model takes over to process the following voice command.

Command Recognition

The Rhino model interprets voice commands and maps them to predefined intents and actions. The model uses context arrays to understand different commands such as showing emotions, moving, interacting, and providing environmental information.





In the Rhino model by Picovoice, an "intent" represents a specific action or command that the user wants the robot to perform, while "slots" are variables within those intents that provide additional details or parameters necessary for carrying out the command. Let's break down these concepts and how optional words and alternatives work in Rhino's context.

Intent and Slots

- Intent: This is the main action or purpose of the command. For example, an intent could be showMeEmotions, Movement, Interaction, Speed, or Environmental. Each intent corresponds to a particular function or task the robot is expected to execute.
- Expressions: Variations of phrases or sentences that all map to the same intent. These expressions are different ways users might naturally issue the command.
- Slots: These are placeholders within the intent that capture specific details or parameters needed to fulfill the intent. For instance, in the showMeEmotions intent, a slot could capture the specific emotion to display (e.g., happy, sad). Slots can be thought of as variables that provide context or detail about the intent.

Optional Words and Alternatives

- Optional Words (Parentheses): Words or phrases enclosed in parentheses are considered optional. This means that the command can still be understood and processed correctly even if the optional word is omitted. For example, in the intent (please) turn (the) lights (on), the words please, the, and on are optional. The command can be understood as "turn lights" or "turn the lights on" without losing its meaning.

- Alternatives (Square Brackets): Words or phrases enclosed in square brackets represent alternatives. These are different words or phrases that can be used interchangeably in the command. For example you can use the word “Switch” instead of “Turn” and the has the same meaning for a lightning system for example.

CODE AND IMPLEMENTATION

The core of Proton 2.0's functionality is embedded in the following code, which integrates voice recognition, motion detection, and environmental sensing. (NOTE only part of the code will be below)

1

```
#include <Picovoice_EN.h>

#include <Arduino_HTS221.h>

#include <Arduino_APDS9960.h>

#include <Arduino_LPS22HB.h>

#include "params.h"

#include "ScreenImages.h"

#include "timer.h"

#define MEMORY_BUFFER_SIZE (70 * 1024)

#define LED_PWR 25

static const char *ACCESS_KEY = "lVXOn6hWshtqhoePLXNehuwIZkBSCso85Kv48QrF2DMskcoxNY6Q3g==";  // AccessKey string obtained from Picovoice Console (https://picovoice.ai/console/)

static pv_picovoice_t *handle = NULL;

static const int obstacleThreshhold = 210;

bool isAwake = false;

static void wake_word_callback(void) {

  Serial.println("Wake word detected!");

    displayImageWithText("?");

}
```

2

```
static void inference_callback(pv_inference_t *inference) {

  if (inference->is_understood) {

    RemoveTextFromImage();

    if (strcmp(inference->intent, "showMeEmotions") == 0) {

      // Extract slot values

      const char *emotionState = NULL;

      for (int32_t i = 0; i < inference->num_slots; i++) {

        if (strcmp(inference->slots[i], "state") == 0) {

          emotionState = inference->values[i];

          break;

        }

      } // Perform action based on the detected intent and slot values

      if (emotionState != NULL) {

        if (strcmp(emotionState, "happy") == 0) {

          Serial.println("I am happy");

          Happy();

        }
```

3

```
else if (strcmp(emotionState, "bored") == 0) {

  Serial.println("I am bored");

}

  else if (strcmp(emotionState, "sad") == 0) {

Serial.println("I am sad");

Sad();

  }

else if (strcmp(emotionState, "angry") == 0) {

Serial.println("I am angry");

          Angry();

        }

else if (strcmp(emotionState, "mad") == 0) {Serial.println("I am mad");

          Angry();

        }

}

      else {

        Serial.println("Error: Missing required slot value.");

      }

    }
```

1

```
void setup() {
  Serial.begin(9600);
  if (!HTS.begin()) {
    Serial.println("Failed to initialize humidity temperature sensor!");
    while (1);
  }
  if (!APDS.begin()) {
    Serial.println("Error initializing APDS-9960 sensor!");
  }

  if (!BARO.begin()) {
    Serial.println("Failed to initialize pressure sensor!");
    while (1);
  }
  delay(250); // wait for the OLED to power up
  display.begin(i2c_Address, true); // Address 0x3C default
  Sleep();
// initialize the digital Pin as an output
  pinMode(LED_PWR, OUTPUT);
  pv_status_t status = pv_audio_rec_init();
  if (status != PV_STATUS_SUCCESS) {
    Serial.print("Audio init failed with ");
Serial.println(pv_status_to_string(status));
    while (1);
  }
```

2

```
char **message_stack = NULL;
  int32_t message_stack_depth = 0;
  pv_status_t error_status;
  status = pv_picovoice_init(
            ACCESS_KEY,
            MEMORY_BUFFER_SIZE,
            memory_buffer,
           sizeof(KEYWORD_ARRAY),
            KEYWORD_ARRAY,
            PORCUPINE_SENSITIVITY,
            wake_word_callback,
            sizeof(CONTEXT_ARRAY),
            CONTEXT_ARRAY,
            RHINO_SENSITIVITY,
            RHINO_ENDPOINT_DURATION_SEC,
RHINO_REQUIRE_ENDPOINT,
            inference_callback,
            &handle);
  if (status != PV_STATUS_SUCCESS) {
    Serial.print("Picovoice init failed with ");
Serial.println(pv_status_to_string(status));
    error_status =
pv_get_error_stack(&message_stack,
&message_stack_depth);
    if (error_status != PV_STATUS_SUCCESS) {
      Serial.println("Unable to get Porcupine error state");
      while (1);
    }
print_error_message(message_stack,
message_stack_depth);
    pv_free_error_stack(message_stack);


    while (1);
  }
```

```
const char *rhino_context = NULL;
 status = pv_picovoice_context_info(handle, &rhino_context);
 if (status != PV_STATUS_SUCCESS) {
  Serial.print("retrieving context info failed with");
  Serial.println(pv_status_to_string(status));
  while (1);
 }
 Serial.println("Wake word: 'proton'");
 Serial.println(rhino_context);
 randomSeed(analogRead(0));
 Serial1.begin(9600);
}
```

3

```
void loop() {
 const int16_t *buffer =
pv_audio_rec_get_new_buffer();
 if (buffer) {
  const pv_status_t status =
pv_picovoice_process(handle, buffer);
  if (status != PV_STATUS_SUCCESS) {
   Serial.print("Picovoice process failed with
");
   Serial.println(pv_status_to_string(status));
   char **message_stack = NULL;
   int32_t message_stack_depth = 0;
   pv_get_error_stack(
    &message_stack,
    &message_stack_depth);
   for (int32_t i = 0; i <
message_stack_depth; i++) {
    Serial.println(message_stack[i]);
   }
```

1

```
pv_free_error_stack(message_stack);
    while (1);
  }
 }

 if (isMoving) {
  if (APDS.proximityAvailable()) {
   // read the proximity
   // - 0   => close
   // - 255 => far
   // - -1  => error
   int proximity = APDS.readProximity();
   if (proximity <= obstacleThreshhold) {
    Serial.println("obstacle detected");
    //send data to the other arduino to
avoide obstacle
   }
  }
 }
```

2

Explanation of The Code

1. Initialization: The Arduino initializes various sensors and the Picovoice models. Sets up Serial communication for debugging and interaction with additional components. Initializes the display and shows a starting image.
2. Wake Word Detection and Command Processing: Picovoice models listen for the wake word "Proton." Upon detection, the Rhino model processes the voice command to determine the intent and executes corresponding actions.
3. Voice Command Execution: Commands include showing emotions, moving in different directions, interacting with the user, and providing environmental information.
4. Motion and Obstacle Detection: Uses proximity sensor data to detect obstacles and adjust movement accordingly.

Functionality

Proton 2.0 supports various voice commands, categorized into different functionalities:

1. Emotional Responses: Commands: "Show me emotions [happy/sad/angry/etc.]" Proton 2.0 displays the corresponding emotion on its screen.
2. Movement: Commands: "Move forward/backward," "Stop," "Turn [left/right]." Proton 2.0 moves or stops based on the command.
3. Interaction: Commands: "Hi/Hello," "Tell me a joke," "Tell me about yourself." Proton 2.0 responds with text on the screen or predefined actions.
4. Environmental Information: Commands: "What's the temperature/humidity/pressure?" Proton 2.0 displays sensor data on the screen.

CONCLUSION

This version of Proton is notably powerful, highlighting the capabilities of modern voice recognition and sensor integration.

If we combine the Arduino Uno and all the hardware components from Proton 1.0, we can build a complete, powerful robot capable of performing more complex and varied tasks, enhancing interactivity, and providing a richer user experience.

Proton 2.0 serves as a demonstration of voice recognition and intent handling, paving the way for future advancements in interactive robotic systems.

REFERENCES

Arduino nano ble 33 sense Tutorial
https://www.youtube.com/playlist?list=PL3E6XmqhhLBHXX2fG2dVER-LOq_7nl9p6

Picovoice Documents
https://picovoice.ai/docs/

Picovoice Example
https://picovoice.ai/blog/on-device-nlu-on-arduino/