

LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

NLP-Powered Service Robot on LEGO Spike and Raspberry Pi 5

Objective(s):

In this lab, students will learn to:

- Configure the single-board computer (SBC) on the LEGO robot for remote wireless connectivity.
- Prepare a Python development environment for robot programming, including creating and activating a virtual environment dedicated to NLP-based interaction.
- Install and configure the libraries required for building a Natural Language Processing (NLP) Human–Robot Interaction interface.
- Apply object-oriented programming (OOP) and NLP techniques — including substring matching, regular expressions, and fuzzy matching — to develop the robot's interactive capabilities step-by-step.
- Implement a function dispatch mechanism to map user intents to corresponding robot actions.

Tools, Equipment and Materials:

- Personal Computer with Internet access
- Raspberry Pi 5 (RPI5) with LEGO Build HAT
- Pre-assembled LEGO Spike robot
- Keyboard, mouse, monitor or VNC remote connection
- Microphone and speaker

Pre-requisites:

- Familiarity with Linux terminal commands and Python programming.

LABSHEET XX

Module

Service Robots
Applications

Module Code

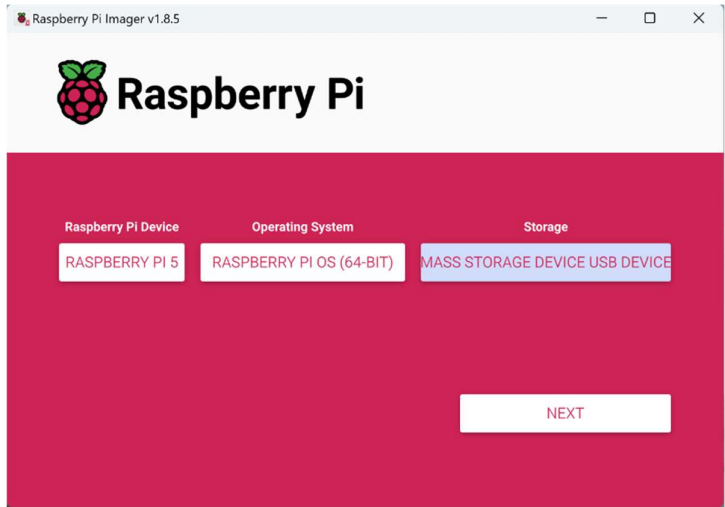
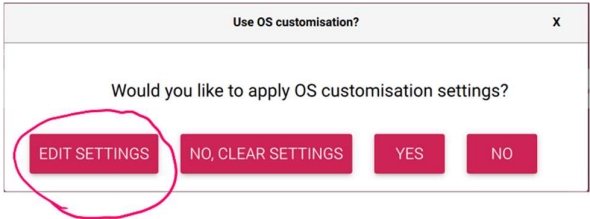
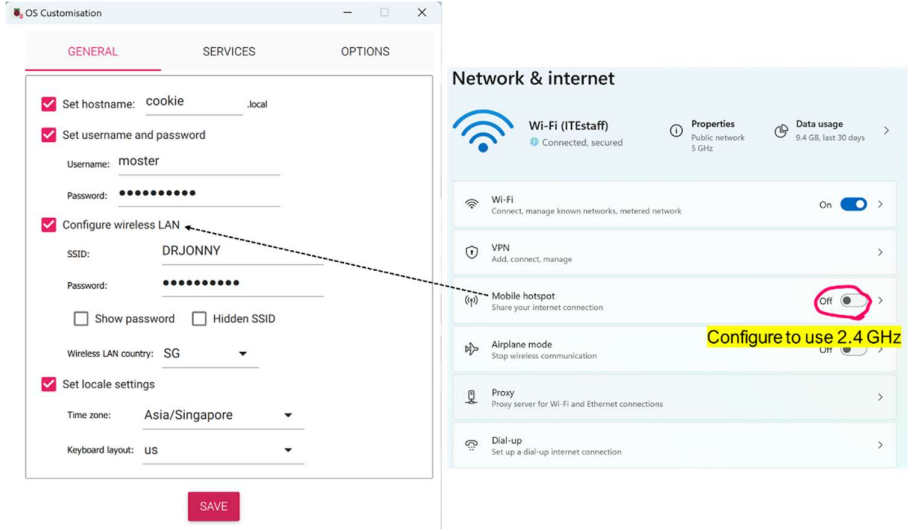
AI53002FP

Duration

4 hours

Flash Raspberry Pi OS Bookworm

1. Install the Raspberry Pi Imager (<https://www.raspberrypi.com/software/>) and SD Card Formatter (<https://www.sdcard.org/downloads/formatter/>).
2. Format the provided microSD card using the SD card formatter.
3. Flash the microSD card with the Raspberry Pi OS following the figures below:

3.1	
3.2	
3.3	

LABSHEET XX

Module

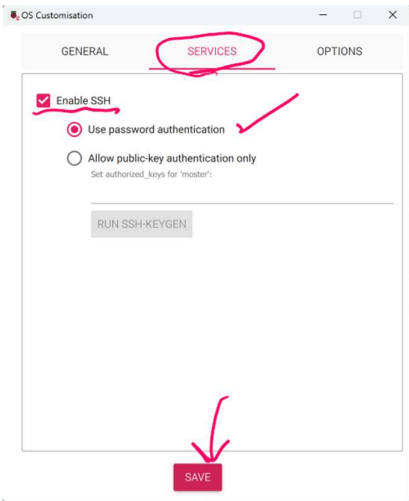
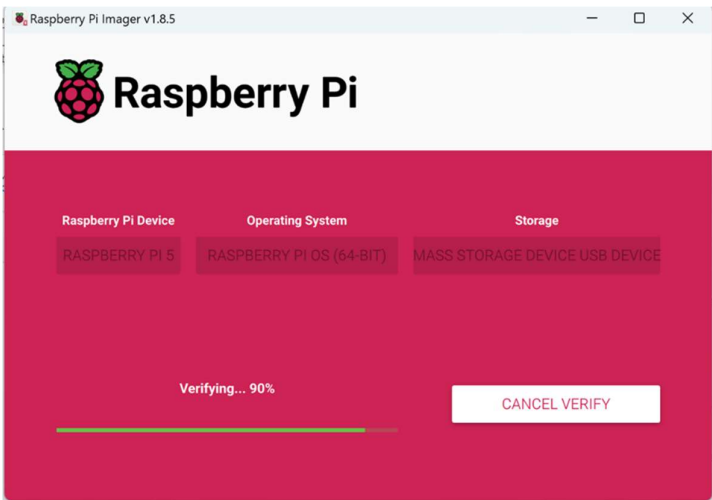
Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

3.4	
3.5	

- Insert the microSD card into Raspberry Pi on the robot and plug in the power supply unit (PSU). Wait 5 minutes for the OS to boot up and connect to your **laptop hotspot**.
- Secure Shell (SSH)** into your Raspberry Pi:

WINDOWS USERS: download and install **PUTTY** (<https://www.putty.org/>).

LABSHEET XX

Module

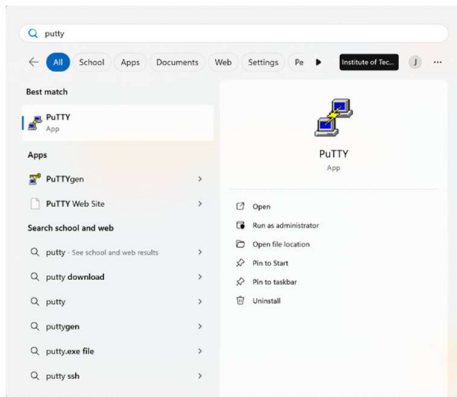
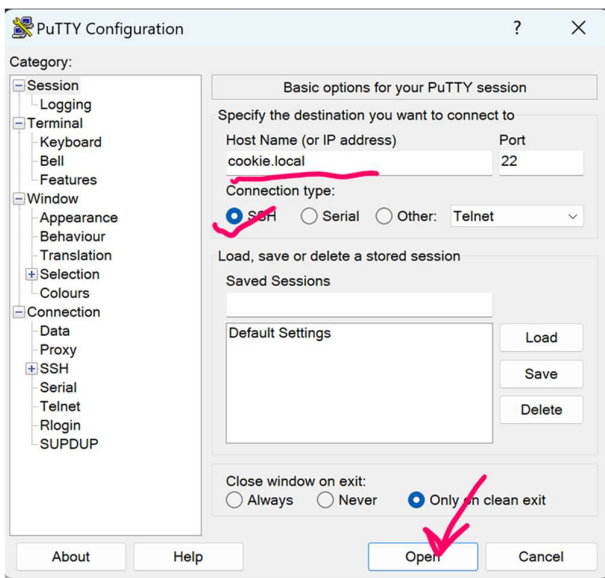
Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

5.1	
5.2	
MAC USERS	
5.1 *	<p>Goto Applications > Utilities, then enter the following command to connect to SSH into Raspberry Pi:</p> <pre>ssh <username>@<hostname>.local</pre>

6. Log into your Raspberry Pi OS user account.

7. Activate the **Virtual Network Computing (VNC)** on Raspberry Pi by following the figures below:

7.1	<p>Enter Raspberry Pi configuration by typing,</p> <ul style="list-style-type: none"> sudo raspi-config
-----	---

LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

7.2	 <pre> moster@cookie: ~ Raspberry Pi 5 Model B Rev 1.0, 8GB Raspberry Pi Software Configuration Tool (raspi-config) 1 System Options Configure system settings 2 Display Options Configure display settings 3 Interface Options Configure connections to peripherals 4 Performance Options Configure performance settings 5 Localisation Options Configure language and regional settings 6 Advanced Options Configure advanced settings 8 Update Update this tool to the latest version 9 About raspi-config Information about this configuration tool <Select> <Finish> </pre>
7.3	 <pre> moster@cookie: ~ Raspberry Pi Software Configuration Tool (raspi-config) I1 SSH Enable/disable remote command line access using SSH I2 RPi Connect Enable/disable Raspberry Pi Connect I3 VNC Enable/disable graphical remote desktop access I4 SPI Enable/disable automatic loading of SPI kernel module I5 I2C Enable/disable automatic loading of I2C kernel module I6 Serial Port Enable/disable shell messages on the serial connection I7 1-Wire Enable/disable one-wire interface I8 Remote GPIO Enable/disable remote access to GPIO pins <Select> <Back> </pre>
7.4	 <pre> moster@cookie: ~ Would you like the VNC Server to be enabled? <Yes> <No> </pre>

8. Install and open **TigerVNC** (<https://sourceforge.net/projects/tigervnc/>) on your laptop computer.
9. In the pop-up window, type **<hostname>.local** in the VNC server field followed by clicking on the **CONNECT** button.
10. Enter user credentials in the new pop-up to gain access to the Raspberry Pi remote desktop. Click **YES** for the subsequent warning message.

LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

Setup Local NLP Environment with LLM and TTS on Raspberry Pi

1. Check that the Raspberry Pi OS date and time are correct. If not, open a new terminal and update it.
 - `sudo date -s "mm/dd/yyyy hh:mm"`
2. Afterwards, update the APT repository and install upgrades.
 - `sudo apt update`
 - `sudo apt full-upgrade`
3. Follow instructions on the following webpage to set up LEGO Build-Hat.
 - <https://www.raspberrypi.com/documentation/accessories/build-hat.html>
4. Install other necessary libraries:
 - `sudo apt install python3-opencv`
 - `sudo apt install libhdf5-dev`
 - `sudo apt-get install libportaudio2`
5. Create and activate **Python virtual environment**:
 - `python3 -m venv --system-site-packages nlp_robot`
 - `source nlp_robot/bin/activate`
6. Upgrade Python Package Manager PIP:
 - `pip install --upgrade pip`
7. Install Text-to-Speech and Audio Libraries
 - `pip install piper-tts==1.2.0`
 - `pip install sounddevice`
 - `pip install soundfile`
8. Install Ollama (LLM runtime)
 - `curl -fsSL https://ollama.com/install.sh | sh`
9. Install Ollama Python Client
 - `pip install ollama`
10. Run a Local LLM
 - `ollama run gemma3:1b`

Testing Service Robot components

1. Verify Text-to-Speech Working

1.1	Download a voice:
https://huggingface.co/rhasspy/piper-voices/tree/main/en/en_GB/semaine/medium	

LABSHEET XX

Module

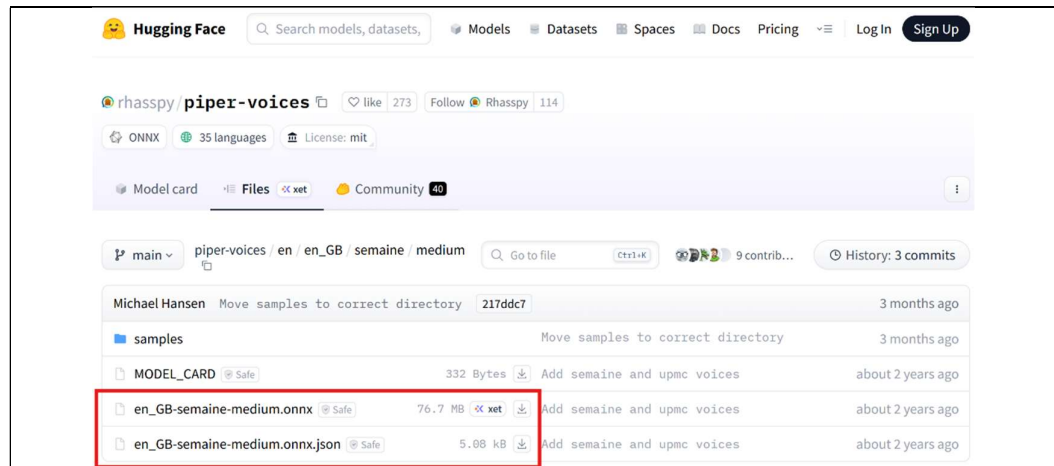
Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours



- 1.2 Connect to your Bluetooth speaker and run the Python script below in Thonny.

```
import numpy as np

import soundfile as sf
import sounddevice as sd
from piper.voice import PiperVoice

MODEL_PATH = "en_GB-semaine-medium.onnx"
voice = PiperVoice.load(MODEL_PATH)
stream = sd.OutputStream(samplerate=voice.config.sample_rate, channels=1, dtype='int16')

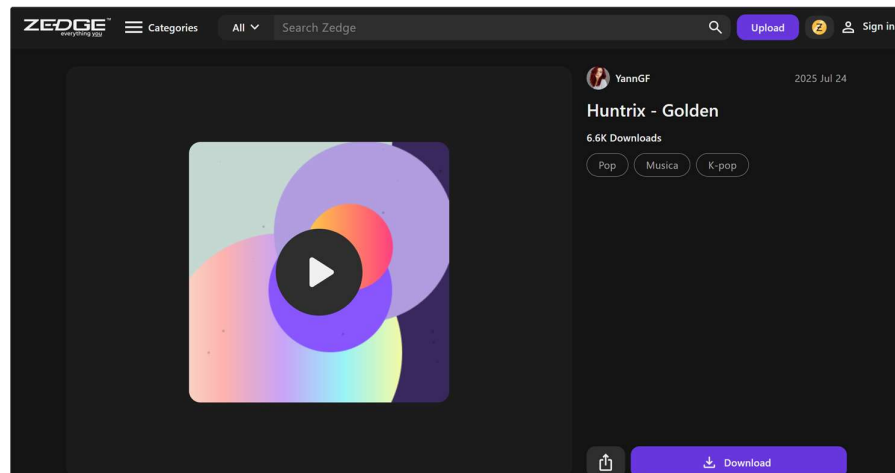
text = "(a) Doctor Jonny is the greatest of all time!"

stream.start()
for audio_bytes in voice.synthesize_stream_raw(text):
    stream.write(np.frombuffer(audio_bytes, dtype=np.int16))
stream.stop()
```

2. Verify sound file working.

- 2.1 Download a ringtone:

<https://www.zedge.net/ringtones/5a9ff96f-be36-4a6a-9d3b-ab9a4b08bae0>



LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

2.2 Connect to your Bluetooth speaker and run the Python script below in Thonny.

```
import soundfile as sf
import sounddevice as sd

filename = "huntrix_golden.mp3"
audio_data, sample_rate = sf.read(filename, dtype="int16")
sd.play(audio_data, samplerate=sample_rate)
print("Playing sound!")
```

3. Verify Ollama works with Python and Text-to-Speech correctly.

```
from ollama import chat
import numpy as np
import sounddevice as sd
from piper.voice import PiperVoice

MODEL_PATH = "en_GB-semaine-medium.onnx"
voice = PiperVoice.load(MODEL_PATH)
stream = sd.OutputStream(samplerate=voice.config.sample_rate, channels=1, dtype='int16')

messages = [{'role': 'system', 'content': 'This is a role play and you are a personal assistant.'}]

while True:
    user_input = input("You: ")
    if user_input.lower() in ["exit", "quit"]:
        break

    messages.append({'role': 'user', 'content': user_input})

    response = chat(model='gemma3:1b', messages=messages, stream=True)

    print("Assistant: ", end='', flush=True)
    full_reply = ""

    for chunk in response:
        content = chunk['message']['content']
        print(content, end='', flush=True)
        full_reply += content

    stream.start()
    for audio_bytes in voice.synthesize_stream_raw(full_reply):
        stream.write(np.frombuffer(audio_bytes, dtype=np.int16))
    stream.stop()

    print()

    messages.append({'role': 'assistant', 'content': full_reply})
```

4. Verify LEGO motors working.

LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

```
from buildhat import Motor

motor_a = Motor('A')
motor_b = Motor('B')
motor_c = Motor('C')

motor_a.run_for_seconds(2)
motor_b.run_for_seconds(2)
motor_c.run_for_seconds(2)
```

5. Verify servo working.

```
import time
from gpiozero import AngularServo

expressions = {"smiley": -90, "indifferent": 0, "witty": 90}

servo = AngularServo(23, min_pulse_width=0.0006, max_pulse_width=0.0023)
servo.angle = 0
time.sleep(1)
servo.value = None

expression = input("You want smiley, indifferent, or witty? ")
if expression in expressions:
    servo.angle = expressions[expression]
    time.sleep(1)
    servo.value = None
```

Programming Service Robot

1. A **SIMPLE ROBOT CLASS** to create service robots that can interact with a user through text-to-speech.

1.1	JSON configuration file to dynamically change robot identity and temperament:
-----	---

```
{
  "name": "Alpha Mini",
  "backstory": "Alpha Mini is friendly robot from Earth and manufactured by the Institute of Technical Education in Singapore.",
  "favorite_color": "red",
  "role": "assistant"
}
```

1.2	The Code:
-----	-----------

LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

```
import json
import numpy as np
import sounddevice as sd
import soundfile as sf
from piper.voice import PiperVoice

class Robot:
    def __init__(self, config_path="robot_config.json"):
        # ----- Load config -----
        try:
            with open(config_path, "r", encoding="utf-8") as f:
                cfg = json.load(f)
        except Exception:
            cfg = {}

        self.name = cfg.get("name", "Alpha Mini")
        self.backstory = cfg.get("backstory", "No backstory provided.")
        self.role = cfg.get("role", "assistant")
        self.favorite_color = cfg.get("favorite_color", "red")

        # ----- TTS setup -----
        self.MODEL_PATH = "en_GB-senaine-medium.onnx"
        self.voice = PiperVoice.load(self.MODEL_PATH)
        self.sample_rate = getattr(self.voice.config, "sample_rate", 22050)
        self.stream = sd.OutputStream(samplerate=self.sample_rate, channels=1, dtype="int16")

        # ----- Function dispatch dictionary -----
        self.dispatch = {
            "greet": {
                "keywords": ["hello", "hi", "good morning", "good afternoon", "good evening"],
                "response": "(a) Hello there! How may I help you sir/ma",
                "action": self.action_greet
            },
            "introduce": {
                "keywords": ["who are you", "introduce", "your name", "your role"],
                "response": "(a) My name is {self.name}. I am your robot {self.role}.",
                "action": self.action_introduce
            },
            "say": {
                "keywords": ["say", "repeat", "speak", "tell me"],
                "response": "(a) Okay, tell me what to say.",
                "action": self.action_say
            },
            "music": {
                "keywords": ["music", "dance", "play", "apt"],
                "response": "(a) I shall play A P T!",
                "action": self.action_music
            }
        }

    # ----- Core utilities -----
    def speak(self, text):
        self.stream.start()
        for audio_bytes in self.voice.synthesize_stream_raw(text):
            self.stream.write(np.frombuffer(audio_bytes, dtype=np.int16))
        self.stream.stop()

    def playsound(self, filename):
        try:
            audio, sr = sf.read(filename, dtype="int16")
        except Exception:
            self.speak("I cannot read the audio file.")
            return
        sd.play(audio, samplerate=sr)
        #sd.wait()

    # ----- Parse user intents -----
    def parse_prompt(self, user_input):
        user_input = user_input.lower()
        for intent, data in self.dispatch.items():
            for kw in data["keywords"]:
                if kw in user_input:
                    self.speak(data["response"])
                    data["action"]()
                    return
        self.speak("I do not understand that command.")

    # ----- Actions -----
    def action_greet(self):
        pass

    def action_introduce(self):
        self.speak(self.backstory)

    def action_say(self):
        say_text = input("What would you like me to say? ")
        self.speak("(a)" + say_text)

    def action_music(self):
        self.playsound("apt.mp3")

    # ----- Cleanup -----
    def close(self):
        if getattr(self.stream, "active", False):
            self.stream.stop()
        self.stream.close()

# -----Instantiating the Service Robot -----
if __name__ == "__main__":
    robot_001 = Robot(config_path="robot_config.json")
    try:
        while True:
            cmd = input("Enter a command (or 'quit' to exit): ")
            if cmd.lower() == "quit":
                break
            robot_001.parse_prompt(cmd)
    finally:
        robot_001.close()
```

LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

2. Service Robot Upgrade: ACTUATOR CONTROL.

2.1 Servo for changing robot expression.

```
# ----- Add these library imports to your program ----
import time
from gpiozero import AngularServo

# ----- Add these class attributes to your program ----
self.expressions = {"smiley": -90, "indifferent": 0, "witty": 90}
self.servo = AngularServo(23, min_pulse_width=0.0006, max_pulse_width=0.0023)
self.servo.angle = self.expressions["indifferent"]
time.sleep(1)
self.servo.value = None

# ----- Add these function dispatch to your program ----
"smiley": {
    "keywords": ["smile", "happy", "smiley"],
    "response": "(a) Smiling is my favorite exercise!",
    "action": lambda: self.change_expression("smiley")
},
"indifferent": {
    "keywords": ["indifferent", "bored", "whatever"],
    "response": "(a) Sure, whatever.",
    "action": lambda: self.change_expression("indifferent")
},
"witty": {
    "keywords": ["witty", "funny", "clever"],
    "response": "I'm not short; I'm concentrated awesome.",
    "action": lambda: self.change_expression("witty")
}

# ----- Add this action function definition to your program ----
def change_expression(self, expression_name):
    if expression_name in self.expressions:
        self.servo.angle = self.expressions[expression_name]
        time.sleep(1)
        self.servo.value = None # Release servo
        print(f"Changed expression to {expression_name}")
```

2.2 LEGO motor for robot motion.

LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

```
# ----- Add these library imports to your program ----
from buildhat import Motor, MotorPair

# ----- Add these class attributes to your program ----
self.pair = MotorPair('A', 'B')
self.body = Motor('C')

# ----- Add these function dispatch to your program ----
"forward": {
    "keywords": ["forward", "walk forward", "go forward"],
    "response": "(a) Walking forward.",
    "action": lambda: self.move_robot("forward")
},
"backward": {
    "keywords": ["backward", "walk back", "reverse", "go back"],
    "response": "(a) Moving backward.",
    "action": lambda: self.move_robot("backward")
},
"left": {
    "keywords": ["left", "turn left"],
    "response": "(a) Turning left.",
    "action": lambda: self.move_robot("left")
},
"right": {
    "keywords": ["right", "turn right"],
    "response": "(a) Turning right.",
    "action": lambda: self.move_robot("right")
},
"rotate body": {
    "keywords": ["rotate body", "turn body", "twist body"],
    "response": "(a) Rotating body.",
    "action": lambda: self.move_robot("rotate body")
}

# ----- Add this action function definition to your program ----
def move_robot(self, direction):
    def drive(rotations, speed_l, speed_r):
        self.pair.run_for_rotations(rotations, speedl=speed_l, speedr=speed_r)

    def rotate_body():
        self.body.run_for_rotations(0.25, speed=10, blocking=True)
        self.body.run_for_rotations(0.25, speed=-10, blocking=True)

    actions = {
        # rotations, left_speed, right_speed
        "forward": lambda: drive(1, 15, -15),
        "backward": lambda: drive(1, -15, 15),
        "left": lambda: drive(1, -20, -20),
        "right": lambda: drive(1, 20, 20),
        "rotate body": rotate_body
    }
    action = actions.get(direction)
    if action:
        action()
```

LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

2.3 Make the robot dance!

```
# ----- Add these function dispatch to your program ----
"groovy": {
    "keywords": ["groovy", "dance party", "party time", "shake it"],
    "response": "(a) You want me to move my booty? Pick a song: 1. song one, 2. song two, 3. song three, or 4. A P T?",
    "action": self.dance
}

# ----- Add this action function definition to your program ----
def dance(self):
    # Map menu choice -> filename stem
    songs = {
        "1": "song1",    # TODO: students add song1.mp3
        "2": "song2",    # TODO: students add song2.mp3
        "3": "song3",    # TODO: students add song3.mp3
        "4": "apt"       # already available: apt.mp3
    }

    choice = input("Choose tracks 1, 2, 3, or 4 ('quit' to exit): ").strip()
    if choice.lower() == "quit":
        self.speak("(a) Okay, no dancing.")
        return

    track_stem = songs.get(choice)
    if not track_stem:
        self.speak("(a) I don't know that one!")
        return

    filename = f"{track_stem}.mp3"
    self.speak("(a) Let's get GROOVING!")
    try:
        # Reuse your existing audio helper
        self.playsound(filename)
    except Exception:
        self.speak("(a) I can't play that track.")
        return

    # ===== DANCE MOVE PLACEHOLDER =====
    # Students fill in servo + LEGO motor moves below.
    # Suggested starter ideas (uncomment and tweak):
    #
    # self.change_expression("smiley")
    # self.move_robot("left")
```

LABSHEET XX

Module

Service Robots
Applications

Module Code

AI53002FP

Duration

4 hours

3. Service Robot Upgrade: LARGE-LANGUAGE MODELS.

```
# ----- Add these libraries to your program -----
import re
import ollama

# ----- Add these class attributes to your program -----
self.llm_model = "gemma3:1b" # adjust if needed
self.conversation_history = [
    {'role': 'system', 'content': f"You are an {self.role}. Your name is {self.name}. {self.backstory}"}
]

# ----- Add this function dispatch to your program -----
"chat": {
    "keywords": ["chat", "talk", "conversation", "ask", "question"],
    "response": f"(a) {self.name} is ready to chat. What would you like to talk about? You can type 'quit' to exit.",
    "action": self.chat_loop
}

# ----- Add this action function definition to your program -----
# ===== LLM helpers =====
def clean_for_tts(self, text):
    """Clean text for TTS - remove unwanted symbols, keep punctuation and quotes."""
    return re.sub(r"^[^ws,?!?'\u2018\u2019\u201C\u201D]", "", text)

# ===== LLM chat =====
def chat(self, user_input):
    """Send one message to the LLM and speak the cleaned reply."""
    self.conversation_history.append({'role': 'user', 'content': user_input})
    print("Thinking...")
    try:
        response = ollama.chat(
            model=self.llm_model,
            messages=self.conversation_history
        )
    except Exception as e:
        err_msg = "(a) I cannot reach the language model right now."
        print("LLM error:", e)
        self.speak(err_msg)
        return

    raw_reply = response['message']['content']
    cleaned = self.clean_for_tts(raw_reply)

    print(f"\n{self.name}: {cleaned}")
    self.speak(cleaned)

    self.conversation_history.append({'role': 'assistant', 'content': cleaned})

def chat_loop(self):
    while True:
        user_input = input("\nYou: ")
        if user_input.lower() == "quit":
            print("Exiting chat mode...")
            self.speak("(a) Goodbye!")
            break
        self.chat(user_input)
```

- End -