FACILITATOR ROBOT

VR_RPI_ROBO

Table of Contents

1. Introduction
2. Components Used
3. Software Implementation
4. Challenges and Troubleshooting
5. Future Improvements
6. Conclusion
7. References
8. Code Explanation

Introduction

The rapid advancement of technology has opened new possibilities in the field of education, where the need for a seamless blend of physical and virtual presence has become increasingly important. The facilitator robot is designed to address the challenges faced by educators and students who may not always be physically present in the classroom. Whether due to health issues, unforeseen circumstances, or the need for remote participation, this telepresence robot offers a solution that allows teachers, students, and administrators to maintain their presence and engagement within the school environment.

This robot serves as a bridge between the physical and digital worlds, enabling a teacher who cannot attend school to still interact with students in real-time, moving around the classroom and engaging with students as if they were physically there. Similarly, students who are unable to attend school due to medical issues can still participate in classroom activities, feeling connected to their peers and the learning environment. Furthermore, school principals can use this robot to oversee school activities remotely, ensuring their presence and authority are felt even when they are not on the premises. The facilitator robot thus represents a significant step forward in creating a more inclusive and flexible educational environment.

COMPONENTS USED			
Name	Component	Quantity	Price
{For Head Setup}			
RP3	*Raspberry Pi 3 Model B+*	*1*	
RP(c)	*Raspberry Pi Camera Module Rev 1.3*	*1*	
PS(r)	*Power Supply (for RP3)*	*1*	
MC	*Micro SD Card (with Minimum 32GB)*	*1*	
PH(g)	*Smartphone (with Gyro Sensor App)*	*1*	
SM	*Servo Motor (for Pan and Tilt)*	*2*	
JW(r)	*Jumper Wires*	*-*	
MS	*Monitor Setup (with HDMI port to Code in RP3)*	*1*	
VR	*VR Box (with Phone Support)*	*1*	
{For Legs Setup}			
DC	*DC Geared Motors*	*4*	
NM	*Nord MCU*	*1*	
MD	*Motor Driver*	*1*	
WS	*Wheels*		
JW(m)	*Jumper Wires*	*-*	
PS(m)	*Power Supply (for NM)*	*1*	
SW	*Switch*	*1*	
{Extras For Setup}			
EX	*Wood + Mica + Glue + Nails*		
Grand Total			

Software Implementation

Install Raspberry Pi OS (Bookworm):

- *Download the Raspberry Pi Imager from the official website.*
- *Select the Raspberry Pi OS (Bookworm) and write it to your micro SD card.*
- *Insert the micro SD card into your Raspberry Pi and power it on.*

Initial Setup:

- *Connect your Raspberry Pi to a monitor, keyboard, and mouse for initial setup.*
- *Power on the Raspberry Pi and follow the on-screen instructions to set up your system.*
- *Connect the Raspberry Pi to your Wi-Fi network.*

Install Required Software In Raspberry Pi:

- *Once your Raspberry Pi is set up, open the terminal and install the necessary packages.*

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3-pip python3-flask python3-gpiozero python3-picamera2
```

- *This installs pip, Flask (for the web server), GPIO Zero (to control the GPIO pins), and the Picamera2 library (to interface with the camera).*

Enable the Camera:

- *Open the Raspberry Pi configuration tool.*

```
sudo raspi-config
```

- *Navigate to Interface Options and enable the camera.*
- *Reboot your Raspberry Pi.*

```
sudo reboot
```

Setting Up GPIO for Servos:

- *Pan Servo: Connect the signal wire to GPIO 17 (pin 11).*
- *Tilt Servo: Connect the signal wire to GPIO 18 (pin 12).*
- *Connect the power and ground wires of the servos to the 5V and GND pins on the Raspberry Pi.*

Connecting and Controlling the Servos:

- *We'll use PWM (Pulse Width Modulation) to control the servos. The code provided handles the movement of the servos using gyroscope data.*

Receiving Gyroscope Data from the Phone:

Option 1: Using an Existing App

- *Install the app "Sensor UDP" from the Google Play Store.*
- *Configure it to send gyroscope data to your Raspberry Pi's IP address and the specified port (e.g., 12345).*

Option 2: Custom App

- *If you're comfortable with Android development, you can create a simple app that reads gyroscope data and sends it over UDP.*

Streaming Video as a VR Feed:

We'll use Flask to create a web server that streams video as a VR feed.

- *Socket Server: Listens for gyroscope data.*
- *Flask Server: Streams the camera feed as a VR video.*
- *Servo Control: Moves the camera based on the received gyroscope data.*

The code provided handles these tasks. Make sure to place the code in a Python file, e.g., Server.py.

Running the Project:

Run the Python Script.

- *Open a terminal and navigate to the directory where the script is saved.*
- *Run the script using*

```
python3 Server.py
```

Connect Your Phone:

- *Ensure your Raspberry Pi and phone are on the same network.*
- *Open the "Sensor UDP" app or similar, and set it to send gyroscope data to the Raspberry Pi's IP address and port 12345.*

View the VR Feed:

- *On your phone, open a web browser and navigate to {http://<Raspberry_Pi_IP>:5000/video_feed}*
- *Place your phone in a VR headset and move it around to see the camera following your movements.*

Challenges and Troubleshooting

Q. Will This Code Run Live With No Delay?

Running this code in real-time without noticeable delay can be challenging, but the performance depends on several factors:

- ***Network Latency****: The speed of your Wi-Fi network will impact how quickly gyroscope data is sent from your phone to the Raspberry Pi and how fast the video stream is served back to your phone.*
- ***Processing Power****: The Raspberry Pi 3 Model B+ has limited processing power. The simultaneous tasks of receiving gyroscope data, controlling servos, capturing and processing video, and serving a web stream may introduce some latency.*
- ***Optimizations****:*
  - *Consider reducing the camera resolution or framerate if you notice delays.*
  - *Avoid heavy processing in the generate_vr_frames() function, as this could slow down the video stream.*
  - *Ensure that the phone and Raspberry Pi are connected to a stable and fast Wi-Fi network.*

In summary, while the code should work, some latency might occur. Optimizations may be required to minimize delays.

Q. Starting the Code on Raspberry Pi Boot?

You can set up your Raspberry Pi to run this script automatically on startup. Here's how:

Method 1: Using rc.local

1. *Open the rc.local file with the following command:*

```
sudo nano /etc/rc.local
```

2. *Before the 'exit 0' line, add the following:*

```
sudo python3 /home/pi/camera_vr.py &
```

   - *Make sure to replace /home/pi/camera_vr.py with the correct path to your Python script.*
   - *The & at the end of the line ensures the script runs in the background.*

3. *Save and exit the file (CTRL + X, then Y, then Enter).*

Method 2: Using cron

1. *Open the cron table for editing:*

```
crontab –e
```

2. *Add the following line to the end of the file:*

```
@reboot /usr/bin/python3 /home/pi/camera_vr.py &
```

   - *Replace /home/pi/camera_vr.py with the correct path to your script.*

3. *Save and exit the file.*

This setup will ensure your script starts automatically when the Raspberry Pi is powered on.

Future Improvements

Advanced Interaction Capabilities:

- *Integrate natural language processing and AI-driven conversational agents to allow the robot to respond to student queries autonomously, enhancing the interactive experience.*
- *Implement gesture recognition so the robot can interpret and respond to non-verbal cues from students and teachers, making interactions more intuitive.*

Enhanced Mobility:

- *Equip the robot with obstacle detection and avoidance systems, ensuring safe navigation in a busy school environment.*

Improved VR Experience:

- *Utilize higher-resolution cameras and advanced VR technologies to create an even more immersive experience, allowing the teacher or student to feel more connected to the classroom.*
- *Explore the use of stereoscopic vision to provide true 3D VR output, enhancing the sense of presence.*

Customizable User Interface:

- *Design a user-friendly interface that can be tailored to the specific needs of different users, such as teachers, students, or administrators. This could include options for camera control, movement, and interaction modes.*
- *Integrate with existing educational software platforms, allowing seamless access to lesson plans, student information, and other educational resources.*

Scalability and Network Optimization:

- *Optimize the robot's communication protocols to handle multiple robots in a single network, allowing for large-scale implementation across different classrooms or even schools.*
- *Improve network efficiency to reduce latency, ensuring real-time interaction without delays or interruptions.*

Conclusion

The facilitator robot represents a transformative tool in the modern educational landscape, bridging the gap between physical and virtual presence. By allowing teachers, students, and administrators to maintain their presence in the classroom regardless of their physical location, this robot fosters a more inclusive and connected learning environment. As technology continues to evolve, there is vast potential for enhancing the robot's capabilities, making it an even more integral part of the educational experience. Whether through advanced AI, improved mobility, or immersive VR, the future of this project holds exciting possibilities for revolutionizing the way we teach and learn.

References

- ***Raspberry Pi Documentation:***
  {https://www.raspberrypi.org/documentation}
- ***Bookworm OS Setup Guide:***
  {https://www.debian.org/releases/bookworm}
- ***Servo Motor Control with Raspberry Pi:***
  {https://learn.adafruit.com/adafruits-raspberry-pi-lesson-8-using-a-servo-motor/overview}
- ***Using the Raspberry Pi Camera Module:***
  {https://projects.raspberrypi.org/en/projects/getting-started-with-picamera}
- ***Python Sockets for Beginners:***
  {https://realpython.com/python-sockets}
- ***Flask Web Framework Documentation:***
  {https://flask.palletsprojects.com/en/2.0.x}

Telepresence Robot: Facilitator for Modern Education

Theme: *Educational Tools*

With education increasingly embracing both physical and virtual learning environments, our ***telepresence robot*** *bridges the gap, ensuring seamless participation for teachers, students, and administrators. Built using* ***Raspberry Pi 3 Model B+****,* ***Raspberry Pi Camera Module****,* ***Nord MCU****, and a* ***motor driver****, this robot allows remote users to engage in real-time classroom activities. Teachers unable to attend in person can navigate the classroom, interact with students, and teach effectively through the robot. Similarly, students facing health issues can attend classes virtually, staying connected with their peers. Even administrators can supervise and monitor school operations remotely, ensuring active oversight.*

The robot's ***Wi-Fi connectivity*** *enables remote control via smartphones ensuring smooth movement.*

While the robot is currently designed as an educational tool, its versatility allows it to serve various other purposes with minimal adjustments. It can support ***disabled individuals*** *with remote assistance, aid in* ***patient care****, or function in* ***smart homes*** *and* ***cities*** *for surveillance or telecommunication. This adaptability highlights the broad potential of the robot in addressing challenges across diverse fields.*

Our project demonstrates the power of robotics and IoT in creating inclusive, engaging, and accessible solutions for modern education and beyond.

Code Explanation

```python
import socket
import RPi.GPIO as GPIO
import time
import threading
from flask import Flask, Response
from picamera2 import Picamera2
from io import BytesIO
from PIL import Image

# Configuration Variables
SERVER_IP = '0.0.0.0'   # IP address for the socket server (0.0.0.0 means listen on all available interfaces)
SERVER_PORT = 12345     # Port for the socket server to receive data
FLASK_PORT = 5000       # Port for the Flask web server to serve the video feed
PAN_PIN = 17  # GPIO pin number for the pan servo motor
TILT_PIN = 18 # GPIO pin number for the tilt servo motor
CAMERA_RESOLUTION = (640, 480)  # Resolution of the camera
CAMERA_FRAMERATE = 24           # Frame rate of the camera
PWM_FREQUENCY = 50  # PWM frequency for controlling the servos
GYRO_MIN = -180  # Minimum value from the gyroscope
GYRO_MAX = 180   # Maximum value from the gyroscope
SERVO_MIN_ANGLE = 0     # Minimum angle for the servos
SERVO_MAX_ANGLE = 180   # Maximum angle for the servos

# Setup GPIO pins for servos
GPIO.setmode(GPIO.BCM)  # Use BCM pin numbering
GPIO.setup(PAN_PIN, GPIO.OUT)  # Set PAN_PIN as output for the pan servo
GPIO.setup(TILT_PIN, GPIO.OUT)  # Set TILT_PIN as output for the tilt servo

# Create PWM instances for each servo
pan_pwm = GPIO.PWM(PAN_PIN, PWM_FREQUENCY)  # Create a PWM instance for the pan servo
tilt_pwm = GPIO.PWM(TILT_PIN, PWM_FREQUENCY)  # Create a PWM instance for the tilt servo

# Start the PWM with a 0% duty cycle (servos will not move initially)
pan_pwm.start(0)
tilt_pwm.start(0)

# Initialize the camera
camera = Picamera2()  # Create a camera object
camera.configure(camera.create_still_configuration())  # Configure the camera settings
camera.start()  # Start the camera

# Flask app to serve the camera feed
app = Flask(__name__)  # Create a Flask app instance

# Function to map gyroscope values to servo angles
def map_value(value, min_gyro, max_gyro, min_angle, max_angle):
    # Convert the gyroscope reading to a servo angle
    return (value - min_gyro) * (max_angle - min_angle) / (max_gyro - min_gyro) + min_angle

# Function to move a servo to a specific angle
def move_servo(pwm, angle):
    duty = map_value(angle, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE, 2, 12)  # Map the angle to a PWM duty cycle
    pwm.ChangeDutyCycle(duty)  # Set the PWM duty cycle to move the servo
    time.sleep(0.02)  # Wait for the servo to move
    pwm.ChangeDutyCycle(0)  # Stop sending the PWM signal to prevent jitter

# Function to handle incoming gyroscope data via UDP
def socket_server():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create a UDP socket
    sock.bind((SERVER_IP, SERVER_PORT))  # Bind the socket to the IP and port

    try:
        while True:
            data, addr = sock.recvfrom(1024)  # Receive data from the phone
            if data:
                gyro_x, gyro_y, _ = map(float, data.decode('utf-8').split(','))  # Parse the gyroscope data
                pan_angle = map_value(gyro_x, GYRO_MIN, GYRO_MAX, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE)  # Map X to pan angle
                tilt_angle = map_value(gyro_y, GYRO_MIN, GYRO_MAX, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE)  # Map Y to tilt angle
                move_servo(pan_pwm, pan_angle)  # Move the pan servo
                move_servo(tilt_pwm, tilt_angle)  # Move the tilt servo

    except KeyboardInterrupt:
        pass  # Handle the program being interrupted by the user

    finally:
        pan_pwm.stop()  # Stop the pan servo PWM
        tilt_pwm.stop()  # Stop the tilt servo PWM
        GPIO.cleanup()  # Clean up the GPIO pins

# Function to generate VR frames from the camera feed
def generate_vr_frames():
    stream = BytesIO()  # Create a buffer to hold the image data
    while True:
        # Capture the image from the camera
        image = camera.capture_array()

        # Convert to a PIL Image object
        pil_image = Image.fromarray(image)
        pil_image = pil_image.resize(CAMERA_RESOLUTION)  # Resize the image to the desired resolution

        # Create two slightly offset images for the VR effect
        left_image = pil_image.crop((0, 0, CAMERA_RESOLUTION[0]//2, CAMERA_RESOLUTION[1]))
        right_image = pil_image.crop((CAMERA_RESOLUTION[0]//2, 0, CAMERA_RESOLUTION[0], CAMERA_RESOLUTION[1]))

        # Create a new image to hold both the left and right images
        combined_image = Image.new('RGB', (CAMERA_RESOLUTION[0] * 2, CAMERA_RESOLUTION[1]))
        combined_image.paste(left_image, (0, 0))  # Paste the left image on the left side
        combined_image.paste(right_image, (CAMERA_RESOLUTION[0], 0))  # Paste the right image on the right side

        # Save the combined image to the buffer
        combined_image.save(stream, format='JPEG')
        stream.seek(0)  # Rewind the buffer to the beginning

        # Yield the image data as part of a multipart response
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + stream.read() + b'\r\n')

        stream.seek(0)  # Clear the buffer for the next frame
        stream.truncate()

# Flask route to serve the video feed
@app.route('/video_feed')
def video_feed():
    return Response(generate_vr_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

# Function to start the Flask server in a separate thread
def start_flask():
    app.run(host=SERVER_IP, port=FLASK_PORT, threaded=True)

# Main entry point of the program
if __name__ == '__main__':
    # Start the Flask server in a separate thread
    flask_thread = threading.Thread(target=start_flask)
    flask_thread.start()

    # Start the socket server to handle gyroscope data
    socket_server()
```