GoPiGo Guide

Prepared by:

Akhila Chowdary Kongara

Document Overview:

This guide provides comprehensive instructions for programming and operating the GoPiGo robot. It includes terminal commands, file management, Python examples, and practical applications for robotics education.

Contents:

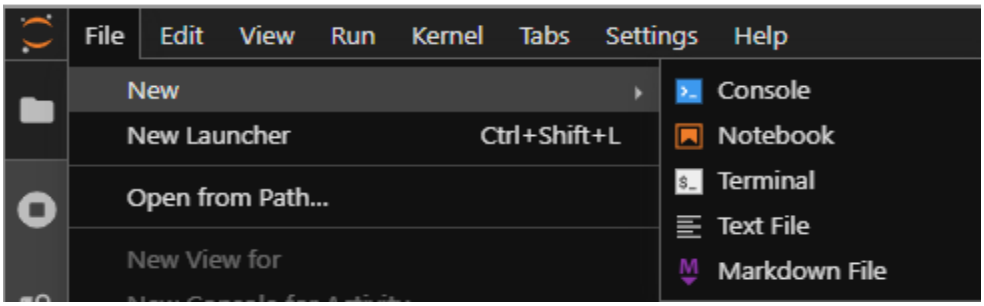
1. Accessing the GoPiGo Terminal via Browser
2. Basic File & Directory Terminal Commands
3. Searching
4. File Creating and Running
5. Permissions
6. Nano Text Editor Shortcuts
7. Terminal Key Shortcuts
8. Installing Software and Reboot
9. Curl
10. Python & pip
11. Python Virtual Environment
12. Disk Info
13. Raspberry Pi Camera Setup
14. GoPiGo3
15. Programs

1.Accessing the GoPiGo Terminal via Browser:

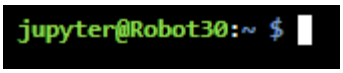
1. **Power on your GoPiGo3 robot.**
    - Make sure it's either:
        - Connected to the same Wi-Fi network as your computer, **or**
        - Acting as a Wi-Fi access point (default behavior).
2. **Open Google Chrome** (recommended for best compatibility).
3. **Enter the following in the address bar**:
   http://10.10.10.10

Note: This is the default IP when GoPiGo is in access point mode.

4. **Navigate to the terminal:**
    - Click on **File** in the top menu.
    - Select **Terminal** from the dropdown.



    - A terminal window will open inside your browser like this.



2. Basic File & Directory Terminal Commands:

- **ls -** List files and folders in current directory
- **cd -** Change Directory
- **cd myfolder/** - Move into "myfolder"
- **cd ..** - Go back one directory
- **mkdir** projects - Create a folder called "projects"
- **rm** myfile.txt - Delete a file
- **rm -r myfolder/** - Delete a folder and all its contents
- **cp source.txt destination.txt** – Copy a file
- **cp -r dir1 dir2** - Copy an entire directory
- **mv oldname.txt newname.txt** - Rename or move a file
- **mv file.txt /path/to/dir/** - Move file to another directory
- **pwd** - Show the current directory (Print Working Directory)
- **tree** - Display directories and files in a tree-like format
- **du -sh *** - Show disk usage for files and folders in human-readable form

- **file filename** - Show the file type (e.g., text, binary)
- **touch notes.txt**- Create an empty file called notes.txt
- **clear** - Clear the terminal screen
- **history** - Show command history

3. Searching:

- **grep 'text' file.txt -** Search for "text" inside a file
- **grep -r 'text' ./folder -** Recursively search for "text" in all files inside a folder
- **find . -name "*.py" -** Find all .py Python files in current directory
- **locate filename -** Find the location of a file (requires mlocate package)

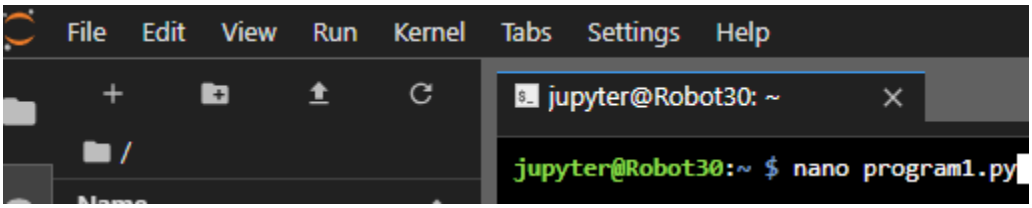
4. File Creating and Running:

- Eg: Creating and Running a Python Script Using nano

  Step 1: Open the file in the nano editor

  Use the nano command followed by your desired filename to open or create a Python script.

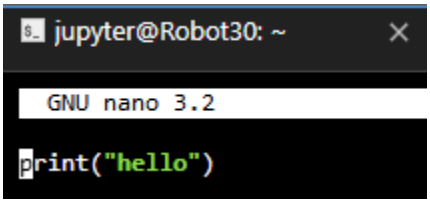
  nano program1.py



  Step 2: Type your Python code

  In the editor, type your Python program. For example:

  print("hello")



  Step 3: Save the file

  Press Ctrl + O to save the file.

  Then press Enter to confirm the filename.

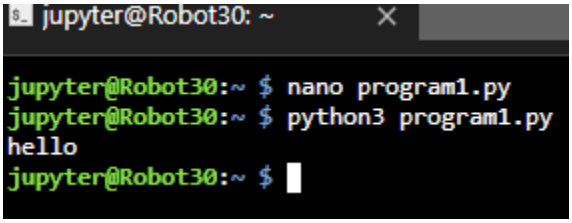
  Step 4: Exit the editor

  Press Ctrl + X to exit nano.

Step 5: Run the Python script

Use the python3 command followed by the script name to run your code:

python3 program.py



5. Permissions:

- **chmod** +x script.sh - Make the script executable

6. Nano Text Editor Shortcuts (used to edit files in terminal):

- Ctrl + O - Write Out (save) the file
- Ctrl + X - Exit nano
- Ctrl + K - Cut current line
- Ctrl + U - Paste (after cut)
- Ctrl + W - Search inside the file
- Ctrl + G - Help menu
- Ctrl + C - Show current line, column position
- Ctrl + _ - Go to line number

7. Terminal Key Shortcuts:

- Ctrl + C - Stop running command or script
- Ctrl + Z - Suspend (pause) a process
- Ctrl + D - Logout or end input (EOF)
- Ctrl + L - Clear the terminal screen
- Ctrl + A - Move to start of the line
- Ctrl + E - Move to end of the line
- Ctrl + U - Delete from cursor to beginning
- Ctrl + K - Delete from cursor to end
- Ctrl + R - Search command history
- Tab - Auto-complete command or filename
- Arrow ↑ ↓ - Scroll through command history
- If you're editing Python scripts for GoPiGo3, you'll mostly use:
  nano filename.py → to open the script

Ctrl + O, then Enter → to save
Ctrl + X → to exit

8. Installing Software and Reboot:

- **sudo apt update**- Update package lists
- **sudo apt upgrade**- Upgrade installed packages
- **Eg:** sudo apt install git - Install a package (example: git)
- **sudo reboot -** Reboot to apply hardware configs

9. curl:

- **curl** https://example.com **-** Fetch content from a website

10. Python & pip:

- **sudo apt install python3-pip** - Install pip for Python 3
- **Eg:** pip3 install numpy - Install a Python package (example: numpy)

11. Python Virtual Environment:

- **python3 -m venv myenv** - Create virtual environment in "myenv"
- **source myenv/bin/activate** - Activate the virtual environment
- **deactivate** - Exit the virtual environment

12. Disk Info:

- **lsblk -** Show connected storage devices (SD card, USB, etc.)

13. Raspberry Pi Camera Setup:

- **Enable the Camera**
  sudo raspi-config - Interface Options -> Camera -> Enable -> Reboot

14. GoPiGo3:

- **jupyter@Robot30: ~ $ python3 ~/Dexter/GoPiGo3/Software/Python/Examples/Read_Info.py** - Check GoPiGo3 device info
- **jupyter@Robot30:~ $ cd ~/Dexter/GoPiGo3/Software/Python/Examples/** - Go to example scripts
- **jupyter@Robot30:~ $ sudo python3 basic_robot.py -** Run a basic GoPiGo3 robot script

15. Programs:

- **Standard Steps for Every GoPiGo Program Implementation**

1. Import Libraries

- import time - Essential for delays and timing control
- import easygopigo3 as easy - Core GoPiGo3 library
- Additional libraries as needed: threading, random, etc.

2. Initialize GoPiGo Robot

- gpg = easy.EasyGoPiGo3() - Create the main robot controller instance
- Initialize sensors based on port connections:
- Button: button = gpg.init_button_sensor("AD1")
- Distance: distance_sensor = gpg.init_distance_sensor("I2C")
- Servo: servo = gpg.init_servo("SERVO1")
- Buzzer: buzzer = gpg.init_buzzer("AD2")

3. Set Initial Parameters

- Set speed: gpg.set_speed(200) (range 0-300)
- Print startup message: print("Program starting...")
- Define safety thresholds: MIN_DISTANCE = 100 (in mm)

4. Define Core Functions

- Basic movement:

```
def move_forward():
    gpg.forward()
def stop():
    gpg.stop()
```

- Sensor reading with validation:

```
def get_distance():
    distance = distance_sensor.read_mm()
    if distance is None or distance == 0:
        return last_valid_distance
    return distance
```

- Button handling with debounce:

```
def is_button_pressed():
    if button.read() == 1:
        time.sleep(0.1)  # Debounce delay
        if button.read() == 1:
            return True
    return False
```

5. Implement Main Control Loop

```
try:
    while True:
        # Main robot logic here
        time.sleep(0.1)  # Standard delay to prevent CPU overuse
except KeyboardInterrupt:
    # Handle program termination
    gpg.stop()
    print("Program terminated.")
```

6. Error Handling Protocol

- Validate all sensor readings before use
- Use try-except blocks around sensor operations
- Implement fallback values for failed readings
- Always include KeyboardInterrupt handler to stop motors

7. Cleanup on Exit

- Stop all motors: gpg.stop()
- Reset servo positions: servo.reset_servo()
- Turn off LEDs: gpg.close_eyes()
- Turn off sounds: buzzer.sound_off()

Here is some of the programs:

1. One is when I starts the button, the song plays and when I stop the button, it stops.

```
import time
import easygopigo3 as easy
import threading

# Create an instance of the GoPiGo3 class.
gpg = easy.EasyGoPiGo3()

# Initialize the buzzer (connected to AD2).
buzzer = gpg.init_buzzer("AD2")

# Initialize the button sensor connected to AD1.
button = gpg.init_button_sensor("AD1")  # Button connected to AD1.

# A flag to track whether the music is playing.
music_playing = False

# Twinkle Twinkle Little Star Notes (scale note names, e.g., "C4", "D4", etc.)
twinkle_notes = ["C4", "C4", "G4", "G4", "A4", "A4", "G4", "F4", "F4", "E4", "E4", "D4", "D4",
"C4"]

# Function to play a song on the buzzer
def play_song(notes):
    global music_playing  # Use the global flag to track the state of the music.

    for note in notes:
        if not music_playing:  # If music is stopped, break out of the loop.
            break

        print(f"Playing note: {note}")
        # Play note (using scale mapping for the buzzer)
        if note in buzzer.scale:  # Ensure note exists in scale
            buzzer.sound(buzzer.scale[note])  # Play the note
            time.sleep(0.5)  # Delay between notes
```

```
            buzzer.sound_off()  # Turn off sound between notes
            time.sleep(0.25)  # Short delay before the next note
        else:
            print(f"Note {note} is not in the buzzer scale.")

# Function to start playing the song in a new thread
def start_playing():
    global music_playing
    music_playing = True
    play_song(twinkle_notes)

# Main loop
while True:
    # Check if the button is pressed (button will return 1 if pressed)
    if button.read() == 1:  # Button is pressed
        if not music_playing:
            # Start playing the song in a separate thread
            print("Button pressed! Playing Twinkle Twinkle Little Star.")
            threading.Thread(target=start_playing).start()
        else:
            # Stop the music by setting music_playing to False
            print("Button pressed! Stopping music.")
            music_playing = False  # Set the flag to False (music stopped).
            buzzer.sound_off()  # Immediately stop the buzzer sound

        # Wait for the button to be released to avoid multiple detections.
        while button.read() == 1:  # Wait until the button is released.
            time.sleep(0.1)

        # Add a short delay to avoid accidental multiple presses.
        time.sleep(0.5)


    time.sleep(0.1)  # Small delay to avoid high CPU usage.
```

2. When I press the button it turns 30, 60, 90 degrees.

```
import time
import easygopigo3 as easy

# Create an instance of the GoPiGo3 class.
gpg = easy.EasyGoPiGo3()

# Initialize the servo motor (connected to PORT 1).
servo = gpg.init_servo("SERVO1")

# Initialize the button sensor connected to AD1.
button = gpg.init_button_sensor("AD1")  # Button connected to AD1.
# Define the angles to move the servo to.
```

```
angles = [30, 60, 90]
current_angle_index = 0  # To track the current position in the angles list.
# Function to move the servo to the next angle
def move_servo():
    global current_angle_index
    # Get the current angle to move to
    target_angle = angles[current_angle_index]
    # Move the servo to the target angle
    print(f"Moving servo to {target_angle} degrees.")
    servo.rotate_servo(target_angle)  # Use the correct method to rotate the servo.
    # Update the index to the next angle
    current_angle_index = (current_angle_index + 1) % len(angles)
# Main loop
while True:
    # Check if the button is pressed (button will return 1 if pressed)
    if button.read() == 1:  # Button is pressed
        print("Button pressed! Moving servo.")
        move_servo()  # Move the servo to the next angle
        # Wait for the button to be released to avoid multiple detections.
        while button.read() == 1:  # Wait until the button is released.
            time.sleep(0.1)
        # Add a short delay to avoid accidental multiple presses.
        time.sleep(0.5)
time.sleep(0.1)  # Small delay to avoid high CPU usage.
```

3. #This code controls a GoPiGo3 robot by making it move forward and detect obstacles using a distance sensor. When an obstacle is detected, it randomly chooses to turn left, turn right, or move backward while playing corresponding songs via a buzzer. The robot also stops when a button is pressed, with debouncing implemented to prevent multiple detections of the button press.

```
import time
import random
import easygopigo3 as easy
import threading
gpg = easy.EasyGoPiGo3()
buzzer = gpg.init_buzzer("AD2")
button = gpg.init_button_sensor("AD1")
servo = gpg.init_servo("SERVO2")
my_distance_sensor = gpg.init_distance_sensor("I2C")
stop_program = threading.Event()
twinkle = ["C4", "C4", "G4", "G4", "A4", "A4", "G4", "F4", "F4", "E4", "E4", "D4", "D4", "C4"]
#left
birthday = ["C4", "C4", "D4", "C4", "F4", "E4", "C4", "C4", "D4", "C4", "G4", "F4"] #Right
jingleBells = ["E4", "E4", "E4", "E4", "E4", "E4", "E4", "G4", "C4", "D4", "E4", "F4", "F4",
"F4"] #Backward
# Function to play a song on the buzzer
def play_song(notes):
    def play():
        for note in notes:
            buzzer.sound(buzzer.scale[note])
            time.sleep(0.5)
            buzzer.sound_off()
```

```
            time.sleep(0.25)
    threading.Thread(target=play, daemon=True).start() # This will help to do multiple tasks. e.g.
moving and at the same time playing song
def move_forward():
    print("Moving forward.")
    gpg.forward()
def stop():
    print("Stopping the robot.")
    gpg.stop()
def turn_left():
    print("Turning left.")
    servo.rotate_servo(90)
    gpg.left()
    play_song(twinkle)
    time.sleep(1)
    gpg.stop()
    servo.rotate_servo(0)
def turn_right():
    print("Turning right.")
    servo.rotate_servo(-90)
    gpg.right()
    play_song(birthday)
    time.sleep(1)
    gpg.stop()
    servo.rotate_servo(0)
def move_backward():
    print("Moving backward.")
    gpg.backward()
    play_song(jingleBells)
    time.sleep(1)
    gpg.stop()
def is_button_pressed():
    if button.read() == 1:
        time.sleep(0.2)
        if button.read() == 1:
            return True
    return False
# Main loop
try:
    while True:
        # Check if the button is pressed (debounced)
        if is_button_pressed():
            print("Button pressed! Stopping robot.")
            stop()
            stop_program.set()
            break
        # Move forward
        move_forward()
        distance = my_distance_sensor.read_mm()
        if distance < 100:
            print("Obstacle detected!")
```

```
            stop()
            time.sleep(1)

            # Randomly choose to turn left, right, or move backward
            action = random.choice(["left", "right", "backward"])
            if action == "left":
                turn_left()
            elif action == "right":
                turn_right()
            else:
                move_backward()
            time.sleep(2)
        time.sleep(0.1)
except KeyboardInterrupt:
    print("Program terminated.")
    stop()
```

4. In this program I used Button, distance and servo.

```
#Button is pressed, the robot starts moving forward, the servo rotates to 90 degrees, and the eyes are opened.
#While moving, if the robot detects an object within 150mm, it stops, the servo moves back to 0 degrees, and the eyes close. If an object is between 150mm and 300mm, the eyes blink as a warning.
# If the button is pressed again, the robot stops, the servo returns to 0 degrees, and the eyes close. The robot waits until the button is pressed again to resume movement.

import time
import easygopigo3 as easy
gpg = easy.EasyGoPiGo3()
btn = gpg.init_button_sensor("AD1")
dst_snsr = gpg.init_distance_sensor("I2C2")
my_servo = gpg.init_servo("SERVO2")
print("Press the button to start moving.")
print("Robot stops automatically if an obstacle is too close.")
s = 0
m = 1
state = s
last_valid_distance = 1000
def blink_eyes():
    """Blink the GoPiGo3 eyes once as a warning signal."""
    gpg.open_eyes()
    time.sleep(0.2)
    gpg.close_eyes()
    time.sleep(0.2)
def move_servo_on_start():
    """Move the servo to 90 degrees when the robot starts moving."""
    my_servo.rotate_servo(90)
    print("Servo moved to 90 degrees")
def move_servo_on_stop():
    """Move the servo back to 0 degrees when the robot stops."""
```

```
    my_servo.rotate_servo(0)
    print("Servo moved to 0 degrees")
while True:
    # Button Press Handling
    if btn.read() == 1:
        while btn.read() == 1:
            time.sleep(0.05)
        if state == s:
            print("Starting movement!")
            gpg.forward()
            gpg.open_eyes()
            move_servo_on_start()
            state = m
        else:
            print("Stopping manually!")
            gpg.stop()
            gpg.close_eyes()
            move_servo_on_stop()
            state = s
        time.sleep(0.3)
    # If moving, check distance
    if state == m:
        distance = dst_snsr.read_mm()
        if distance is None or distance == 0:
            distance = last_valid_distance
        else:
            last_valid_distance = distance
        print(f"Distance: {distance} mm")
        if 150 < distance <= 300:
            print("Warning! Object ahead.")
            blink_eyes()
            print("STOP! Obstacle too close.")
            gpg.stop()
            gpg.close_eyes()
            move_servo_on_stop()
            state = s
    time.sleep(0.1)
```

5. Here is the code which operates a **GoPiGo3** robot, enabling movement, obstacle detection, and voice feedback via a speaker. The robot moves forward while a **distance sensor** monitors for obstacles; if one is too close, it **stops** and randomly turns left, right, or moves backward. A **buzzer** emits sound alerts, and a **speaker (espeak)** announces actions like movement and stopping. Pressing a **button halts** the robot, with built-in error handling for stability. The loop continues running until manually stopped or interrupted by a button press.

```
import time

import random
import easygopigo3 as easy
import threading
```

```
from subprocess import call
gpg = easy.EasyGoPiGo3()
buzzer = gpg.init_buzzer("AD2")
button = gpg.init_button_sensor("AD1")
servo = gpg.init_servo("SERVO2")
try:
    my_distance_sensor = gpg.init_distance_sensor("I2C")
    print("Distance sensor initialized successfully.")
except Exception as e:
    my_distance_sensor = None
    print(f" Error initializing distance sensor: {e}")
def beep():
    buzzer.sound(1000)
    time.sleep(0.2)
    buzzer.sound_off()
def move_forward():
    speak("Moving forward")
    print(" Moving forward.")
    gpg.forward()
def stop():
    speak("Stopping")
    print(" Stopping the robot.")
    gpg.stop()
def turn_left():
    speak("Turning left")
    print("Turning left.")
    servo.rotate_servo(90)
    gpg.left()
    beep()
    time.sleep(1)
    gpg.stop()
    servo.rotate_servo(0)
def turn_right():
    speak("Turning right")
    print(" Turning right.")
    servo.rotate_servo(-90)
    gpg.right()
    beep()
    time.sleep(1)
    gpg.stop()
    servo.rotate_servo(0)
def move_backward():
    speak("Moving backward")
    print(" Moving backward.")
    gpg.backward()
    beep()
    time.sleep(1)
    gpg.stop()
def is_button_pressed():
    if button.read() == 1:
        time.sleep(0.2)
```

```
        if button.read() == 1:
            return True
    return False
# Main loop
try:
    while True:
        if is_button_pressed():
            print(" Button pressed! Stopping robot.")
            stop()
            break
        move_forward()
        if my_distance_sensor:
            try:
                distance = my_distance_sensor.read_mm()
                if distance is None or distance <= 0:
                    print(" Invalid distance reading! Retrying...")
                    continue
                print(f" Distance Sensor Reading: {distance} mm")
                if distance < 30:
                    print(f" Obstacle detected at {distance} mm!")
                    stop()
                    time.sleep(0.5)
                    action = random.choice(["left", "right", "backward"])
                    if action == "left":
                        turn_left()
                    elif action == "right":
                        turn_right()
                    else:
                        move_backward()
                    time.sleep(2)
            except Exception as e:
                print(f"Distance sensor error: {e}")
        time.sleep(0.05)
except KeyboardInterrupt:
    print(" Program terminated.")
    stop()
```

6. This script tests the Raspberry Pi camera using the picamera library by initializing the camera, setting resolution, capturing an image, and saving it locally. It also includes basic error handling to report camera initialization or capture issues.

```
import picamera

import time

print("Testing basic camera functionality...")
try:
    # Initialize the camera directly
    camera = picamera.PiCamera()
    print("Camera initialized successfully using picamera library")
    # Set resolution
    camera.resolution = (1024, 768)
```

```
    # Wait for camera to initialize
    time.sleep(2)
    # Take a picture
    local_file = "test_direct.jpg"
    print(f"Taking photo and saving to {local_file}")
    camera.capture(local_file)
    print("Photo captured successfully!")
    # Clean up
    camera.close()
except Exception as e:
    print(f"Camera error: {e}")
```

7. The robot moves forward and constantly checks for nearby obstacles using a distance sensor.
   If something is too close, it stops and waits for the button to be pressed.
   When you press the button, it takes a photo with the camera and then keeps moving again.

```
import time
import easygopigo3 as easy
import picamera
from datetime import datetime
# Initialize GoPiGo3, Distance Sensor, and Button
gpg = easy.EasyGoPiGo3()
distance_sensor = gpg.init_distance_sensor()
button = gpg.init_button_sensor("AD1")
# Camera setup
camera = picamera.PiCamera()
camera.resolution = (1024, 768)
# Constants
THRESHOLD_MM = 300  # 30 cm
SPEED = 150
def take_photo():
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"photo_{timestamp}.jpg"
    print(f"📸 Capturing photo: {filename}")
    camera.capture(filename)
    print("☑ Photo saved.")
try:
    print("🤖 PhotoBot Patrol starting. Press Ctrl-C to stop.")
    gpg.set_speed(SPEED)
    while True:
        distance = distance_sensor.read_mm()
        print(f"📏 Distance: {distance} mm")
        if distance < THRESHOLD_MM:
            print("🚧 Obstacle detected! Stopping.")
            gpg.stop()
            gpg.open_eyes()
            print("🔛 Waiting for button press to take a photo.")
            while button.read() == 0:
```

```
            time.sleep(0.05)
        take_photo()
        # Blink eyes for feedback
        for _ in range(3):
            gpg.close_eyes()
            time.sleep(0.2)
            gpg.open_eyes()
            time.sleep(0.2)
        print("🔄 Resuming patrol...")
        gpg.close_eyes()
        gpg.set_speed(SPEED)
        gpg.forward()
    else:
        gpg.forward()
    time.sleep(0.1)
except KeyboardInterrupt:
    print("\n🛑 Program interrupted by user.")
finally:
    gpg.stop()
    gpg.close_eyes()
    camera.close()
    print("🔚 Robot safely stopped. Goodbye!")
```

8. The program makes the GoPiGo3 robot move forward while using a distance sensor to detect obstacles. When an obstacle is detected within 30 cm, the robot stops, waits for a button press, takes a photo, then blinks its eyes and continues moving forward.

```
import time
import easygopigo3 as easy
import picamera
from datetime import datetime
# Initialize GoPiGo3, Distance Sensor, and Button
gpg = easy.EasyGoPiGo3()
distance_sensor = gpg.init_distance_sensor("AD2")
button = gpg.init_button_sensor("AD1")
# Camera setup
camera = picamera.PiCamera()
camera.resolution = (1024, 768)
# Constants
THRESHOLD_MM = 300  # 30 cm
SPEED = 150
MOVING_FORWARD = False  # Track the intended state of the robot

def take_photo():
    """Capture and save a photo with timestamp filename"""
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
```

```
    filename = f"photo_{timestamp}.jpg"
    print(f"📸 Capturing photo: {filename}")
    camera.capture(filename)
    print("☑️ Photo saved.")
def blink_eyes(blink_count=3, blink_time=0.2):
    """Blink the robot's eyes a specified number of times"""
    for _ in range(blink_count):
        gpg.close_eyes()
        time.sleep(blink_time)
        gpg.open_eyes()
        time.sleep(blink_time)
try:
    print("🤖 PhotoBot Patrol starting. Press Ctrl-C to stop.")
    gpg.set_speed(SPEED)

    # Explicitly start moving forward and track state
    gpg.forward()
    MOVING_FORWARD = True
    print("▶️ Robot moving forward")
    while True:
        # Always read the distance first
        distance = distance_sensor.read_mm()
        print(f"📏 Distance: {distance} mm")
        # Check if we should be stopped due to obstacle
        if distance < THRESHOLD_MM:
            if MOVING_FORWARD:  # Only stop if we're currently moving
                print("🚧 Obstacle detected! Stopping.")
                gpg.stop()
                MOVING_FORWARD = False
                gpg.open_eyes()
                print("📴 Waiting for button press to take a photo.")
                # Wait for button press
                button_pressed = False
                while not button_pressed:
                    if button.read() == 1:  # Button is pressed
                        button_pressed = True
                    time.sleep(0.05)
                # Take photo once button is pressed
                take_photo()
                # Blink eyes for feedback
                blink_eyes()

                print("🔁 Resuming patrol...")
                gpg.close_eyes()

                # Resume moving forward
```

```
                gpg.set_speed(SPEED)
                gpg.forward()
                MOVING_FORWARD = True
                print("▶️ Robot moving forward")
            else:
                # No obstacle - make sure we're moving
                if not MOVING_FORWARD:
                    gpg.forward()
                    MOVING_FORWARD = True
                    print("▶️ Robot moving forward")

            # Add small delay to prevent CPU overuse
            time.sleep(0.1)

    except KeyboardInterrupt:
        print("\n🛑 Program interrupted by user.")
    finally:
        # Clean up - stop robot and close camera
        gpg.stop()
        gpg.close_eyes()
        camera.close()
        print("🔚 Robot safely stopped. Goodbye!")
```