OmniPi

Version 2.0.0

Steven Klinefelter

December 20, 2023

Contents

1 Introduction

The goal of the OmniPi is to create a simple omnidirectional robot using parts that were available in the lab, or I already had on hand. This evolved into trying to keep the robot as affordable and expandable as possible while exploring all steps in the design process.

2 Required Components

2.1 Tools

The following tools will be needed to build the robot:

- Screwdriver
- Pliers/ Wrench
- Allen Key
- 3D Printer

2.2 Materials

The following materials will be needed to build the robot:

- Motor Drivers (2x)[1]
- Raspberry Pi 4[2]
- RPI UPS[3]
- Breadboard Wires[4]
- Motors (4x)[5]
- Mecanum Wheels[6]
- Chassis Print[7].

[1] https://a.co/d/0ZAUHJW
[2] https://a.co/d/a6zDcXv
[3] https://a.co/d/fqmVb0i
[4] https://a.co/d/2cKIRl8
[5] https://www.robotshop.com/products/mecanum-wheel-4-pack-w-metal-hubs
[6] https://www.robotshop.com/products/mecanum-wheel-4-pack-w-metal-hubs
[7] https://github.com/Klinefelters/OmniPi/blob/main/Chassis.stl

It is recommended to clone the github repository and follow along to understand the code better.

3 Starting with Motion

Since I got to choose where to start, I wanted to begin with the part I thought would be the most fun: the motion!

3.1 Wiring

I started by wiring up the motor controllers following a basic wiring diagram found in Figure 1. These controllers use 2 wires to describe the motor's direction and 1 wire to control the speed, using pulse-width modulation, or PWM for short. Thankfully, this is all included in a GPIO library, so the specifics of how it works can be glossed over for now. The important part as it relates to this project is that it allows us to control the speed of the DC motors, even if it is rather crude.

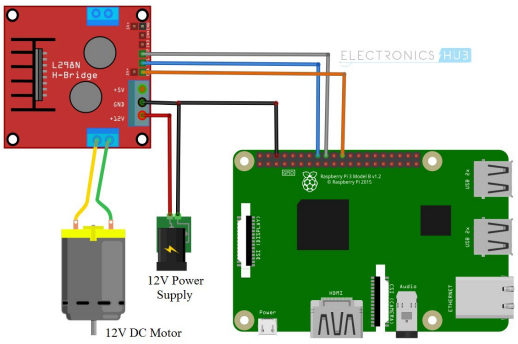

Figure 1: The basic wiring diagram for the motor controller from [1]

3.2 Imports

To control the motors, you'll need to import the GPIO library.

```
import RPi.GPIO as GPIO
```

3.3 Setup

From here, you can set up the board's pins with the following (assuming in1 and in2 are your direction control, and en is the speed control wire):

```
GPIO.setmode(GPIO.BCM)

# Setup all of the needed pins
GPIO.setup(in1, GPIO.OUT)
GPIO.setup(in2, GPIO.OUT)
GPIO.setup(en, GPIO.OUT)
```

3.4 Direction

In order to make the motor spin forwards, we set in1 to high and in2 to low. If we want to go backward, then in2 would be high and in1 would be low. If both are high or both are low, then the motor won't spin. In the end, it looks like this:

```
# Stopping the motor
GPIO.output(in1, GPIO.LOW)
GPIO.output(in2, GPIO.LOW)

# Spinning the motor forwards
GPIO.output(in1, GPIO.HIGH)
GPIO.output(in2, GPIO.LOW)

# Spinning the motor backward
GPIO.output(in1, GPIO.LOW)
GPIO.output(in2, GPIO.HIGH)
```

3.5 Speed

Finally, we can control the speed of the motor using the PWM on pin en. To do that, we need to set up the PWM pin like so:

```
p = GPIO.PWM(en, 1000)
```

Then, we can control the speed of the motor by passing the pin a frequency through the Change Duty Cycle function. In theory, this should have a max of 1000 and a min of 0, but the motor won't spin if the freq is 0, and after a certain point, the motor doesn't get any faster.

```
p.ChangeDutyCycle(freq)
```

3.6 Cleanup

Finally, when we are done, we have to release the GPIO pin resources:

```
GPIO.cleanup()
```

When putting all of this together, you can build a Python class that allows you to easily control any motor. Then we can create four objects of this class and use them to control our wheels. For that code, visit the L298N Class Github file[8].

4 Putting Together 4 motors

Now that we have one motor moving, we need to control 4 at the same time. Luckily, this is easy because when you set the PWM frequency, it stays at that frequency until you change it. This means we can simply loop through the motors and update their control signals and frequencies in order to get a sort of velocity control.

4.1 Imports

To start with, we have to import the L298N class explained in the previous section:

```
from l298n import L298N
```

4.2 Setup

Now we can set up 4 of these motors with our desired pins:

```
frontLeft = L298N(in1=27, in2=17, en=22,)
backLeft = L298N(in1=26, in2=19, en=13,)
frontRight = L298N(in1=20, in2=21, en=16,)
backRight = L298N(in1=24, in2=23, en=25,)
```

4.3 Axis Control

And now we can add some logic to convert robot velocities into wheel velocities. We'll take vx, vy, and theta. All of these values will be from -1 to 1. -1 will

[8] https://github.com/Klinefelters/OmniPi/blob/main/api/l298n.py

be left, rotate left, or backward, while 1 will be right, rotate right, or forward. All of these will be relative to the robot.

In order to achieve this, we'll tackle one at a time. To move forward, every motor has to go forwards, so we'll add the y to every motor. To rotate in place, we need to make the left motors go forward and the right go backward, so we'll add theta to the left motors and subtract from the right. Finally, to tackle sideways, it's an x pattern. So to go right, the front left is going forwards, as well as the back right. The back left and front right are going backward. This means we can add vx to the front left and back right while subtracting vx from the back left and front right. In order to even things out, we'll make sure none of the speeds are greater than 1 by dividing all of the speeds by the highest speed we could calculate. It sounds like a lot, but it boils down to this:

```
def move(self, vx: float, vy: float, theta: float):
    denominator = max(abs(vy) + abs(vx) + abs(theta), 1)
    frontLeft.move((vy + vx + theta) / denominator)
    backLeft.move((vy - vx + theta) / denominator)
    frontRight.move((vy - vx - theta) / denominator)
    backRight.move((vy + vx - theta) / denominator)
```

Now we have a way to control the robot with a given axis measurement, from -1 to 1. Luckily, pygame allows us to use game controllers while measuring their axis from -1 to 1. This makes it relatively simple to bind motion to a game controller. Putting all of it together makes the Robot Class Github file[9].

5 Making a Remote Server

Since we have the logic to control a robot let's make it remote over WiFi. I won't cover how to set up a hotspot because there are plenty of tutorials on that out there. Let's assume you set one up on your laptop or the Pi so they can communicate over the local area network.

5.1 Server

Let's start with the server. This will be a short script that will start the flask server on the robot allowing the client to interact with it.

```
from flask import Flask
from flask_cors import CORS
from events import events_bp
```

[9] https://github.com/Klinefelters/OmniPi/blob/main/api/robot.py

```
app = Flask(__name__)
CORS(app, resources={
     r"*": {"origins": ["http://localhost:5173", "http://localhost:8000"]}})
app.register_blueprint(events_bp)


if __name__ == '__main__':
    app.run(port=8000, debug=True)
```

5.2 Events

You'll notice that the events and endpoints are defined in events.py as a flask blue print. This is mostly boiler plate code, and can be edited as needed. Here we will create our robot and camera that we want to work with and define how the client can interact with them.

```
from flask import Blueprint, jsonify, request, Response
from camera import Camera
from mockRobot import Robot

events_bp = Blueprint('events', __name__)

robot = Robot()
camera = Camera()


@events_bp.route('/control', methods=['POST'])
def control_robot():
    data = request.get_json()
    x = data.get('x', 0)
    y = data.get('y', 0)
    r = data.get('r', 0)
    robot.move(x, y, r)
    return jsonify({"message": "Robot moved successfully"})


@events_bp.route('/video_feed')
def video_feed():
    return Response(camera.gen_frames(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
```

5.3 Camera

You may have noticed that we added in a camera object. this is defined in the api as well, but is just a simple wrapper around a cv2 video capture object. We make use of python generators here using yield instead of return.

```
import cv2


class Camera(object):
    def __init__(self):
        self.cap = cv2.VideoCapture(0)

    def gen_frames(self):
        while True:
            _, frame = self.cap.read()
            _, buffer = cv2.imencode('.jpg', frame)
            frame_data = buffer.tobytes()
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n'
                   + frame_data + b'\r\n')
```

6 Creating the Client for that Server

Due to a lack of support for react minting in latex, this section won't have as much detail as the last. It is recommended to explore the code outside of the most basic examples provided.

6.1 Setup

Let's navigate into the client directory of the github and run npm i in a terminal window. This will install of the packages and prepare it to be run

```
cd client
npm i
```

Now we can start the client in the dev enviroment. We do this so it will automatically reload if we make any coding changes.

```
npm run dev
```

That's it! Now you can visit the web address in the terminal window to see the client. The whole thing is programmed in react using popular libraries to control the robot.

6.2 Client API Calls

The robot is controlled using axios in the Omni Pi component. You can see the simple requests we defined in events.py coming into play here.

```
import  { useEffect } from 'react';
import axios from 'axios';
import { Box, Image, Flex, Heading } from '@chakra-ui/react';

export default function OmniPi({vx, vy, vr}) {

  const videoUrl = 'http://localhost:8000/video_feed';

  const sendData = () => {
    const data = {
      x: 0,
      y: 0,
      r: 0
    };

    axios.post('http://localhost:8000/control', data)
      .then(response => {
        console.log(response.data);
      })
      .catch(error => {
        console.error(error);
      });
  };

  useEffect(() => {
    const intervalId = setInterval(() => {
      sendData();
    }, 50);

    // Clear the interval when the component unmounts
    return () => {
      clearInterval(intervalId);
    };
  }, [vx, vy, vr]);

  return (
    <Box bg="brand.900" p={4} borderRadius='lg'>
      <Flex p={2} alignItems='center'>
        <Heading color="white" size='xl' textAlign={"center"} >OmniPi</Heading>
      </Flex>
```

```
      <Image id="videoFeed" src={videoUrl} alt="video_feed"  w="640px" h="480px"
    </Box>
  );
}
```

6.3 Input presses

Then we can use this in the page we want to control things from. Take the keyboard control page for example. We import the combined control, which is the simulator and the omnipi bundled together, and set it up! Using a simple keylistener to update our state variables, we can control the robot and a simulator!

```
import { useState, useEffect } from "react";
import { Box, Flex, Spacer } from '@chakra-ui/react'
import CombinedControl from "../components/CombinedControl";

export default function Keyboard() {

  const [vx, setVX] = useState(0);
  const [vy, setVY] = useState(0);
  const [vr, setVR] = useState(0);

  const handleKeyPress = (event) => {
    // Check if a specific key is pressed
    if (event.key === 'a') {setVX(-1);}
    if (event.key === 'd') {setVX(1);}
    if (event.key === 'q') {setVR(-1);}
    if (event.key === 'e') {setVR(1);}
    if (event.key === 'w') {setVY(1);}
    if (event.key === 's') {setVY(-1);}
  };

  // Define a function to handle key releases
  const handleKeyRelease = (event) => {
    if (event.key === 'a') {setVX(0);}
    if (event.key === 'd') {setVX(0);}
    if (event.key === 'q') {setVR(0);}
    if (event.key === 'e') {setVR(0);}
    if (event.key === 'w') {setVY(0);}
    if (event.key === 's') {setVY(0);}
  };

  // Add the keydown event listener when the component mounts
```

```
  useEffect(() => {
    window.addEventListener('keydown', handleKeyPress);
    window.addEventListener('keyup', handleKeyRelease);

    // Remove the event listeners when the component unmounts
    return () => {
      window.removeEventListener('keydown', handleKeyPress);
      window.removeEventListener('keyup', handleKeyRelease);
    };
  }, []);


  return (
    <Box bg="black" h="100vh">
      <Spacer h="15px"/>
      <Flex>
        <Spacer />
        <CombinedControl vx={vx} vy={vy} vr={vr}/>
        <Spacer />
      </Flex>

    </Box>
  );
}
```

This basic formula can be used to create simple pages all the way to the most complex pages!

7 Use of Generative AI

This project used generative AI to assist in multiple aspects of development. Most notably, Chat GPT was prompted to generate all of the doc strings for the python files. The following prompt was provided with each file:

> Please generate complete docstrings for the following python file:
>
> *** Python Code Here ***

Chat GPT was also used for troubleshooting various python bugs and errors. Errors would be copied and pasted into Chat GPT with speculation around what is causing the errors. Since GPT 3.5 isn't good with limited context, it's important to provide it with what you think is causing the error. Typically it can narrow it down from there. Typically it looks like this:

> *** Error Here ***
>
> The following error was encountered when trying to run the robot class. I believe that the error was caused because the library I was using was requesting the GPIO resources twice. How can I address this issue?

Finally Chat GPT was used to proofread each section of this document! By feeding GPT 3.5 a simple prompt, it can catch grammatical errors, as well as provide feedback on overall structure.

> Please proofread the following .tex document that's used as a section in a larger document. Provide feedback on the structure, grammar, and point out any simple mistakes.
>
> *** LaTex Code Here ***

References

[1] Administrator, *Raspberry Pi L298N Interface Tutorial*. Electronicshub, Feb. 09, 2018. https://www.electronicshub.org/raspberry-pi-l298n-interface-tutorial-control-dc-motor-l298n-raspberry (accessed Nov. 12, 2023).