

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий

# Курсовая работа

Колесная платформа  
по дисциплине «Микропроцессорные системы»

Выполнили:

Ферапонтов М.В.  
Савчук А.А.  
Дорошин Д.А.  
Луцай П.П.  
Артеев Д.Д.  
Яровой В.Д.

Группа:

гр. 3530904/00104

Проверил:

Круглов С.К.

Санкт-Петербург  
2022

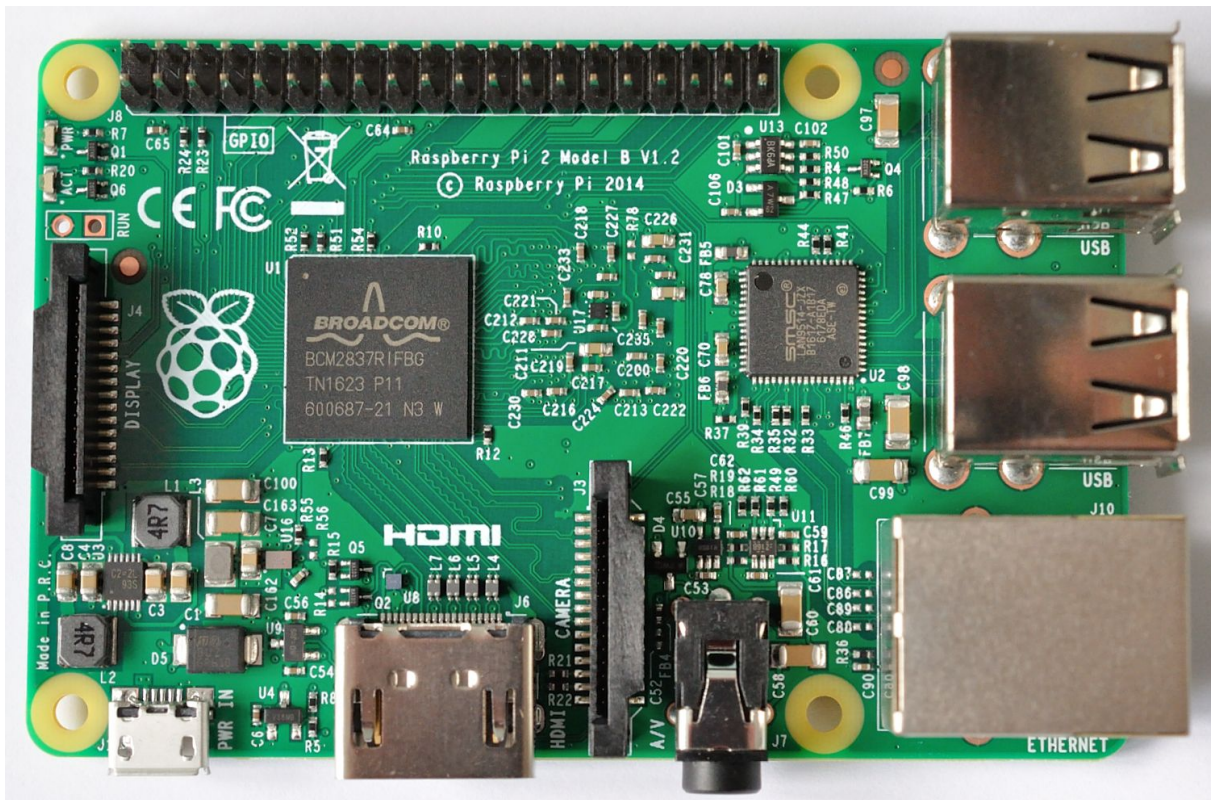
# Содержание

<b>1</b>	<b>Аппаратная реализация</b>	<b>2</b>
1.1	Raspberry pi 2b+ . . . . .	2
1.2	Используемые детали и модули . . . . .	2
1.3	Широтно-импульсная модуляция . . . . .	3
1.4	Питание . . . . .	4
<b>2</b>	<b>Схемы</b>	<b>4</b>
2.1	Принципиальная схема драйвера L289N . . . . .	4
2.2	Принципиальная схема Raspberry Pi . . . . .	5
2.3	Схема подключений . . . . .	6
<b>3</b>	<b>Программное обеспечение</b>	<b>8</b>
3.1	Используемые инструменты . . . . .	8
3.2	Код программы . . . . .	8

# 1 Аппаратная реализация

## 1.1 Raspberry pi 2b+

При разработке был использован Raspberry Pi 2 model B+.



Характеристики:

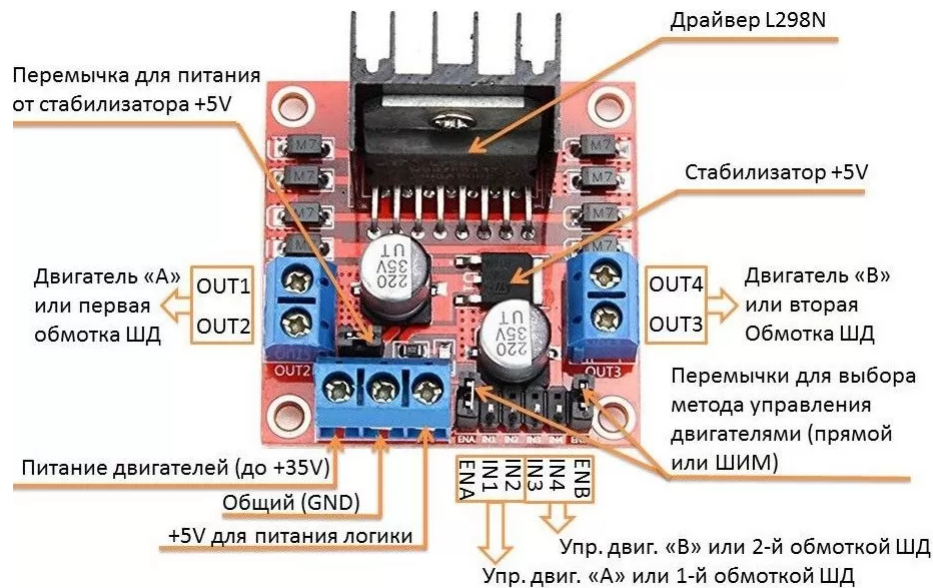
- Процессор - ARM Cortex-A7 CPU 900MHz
- Количество ядер - 4
- Оперативная память - 1Gb

## 1.2 Используемые детали и модули

Были использованы детали из набора Arduino

1. Драйвер управления движения моторов L289N
2. Колеса 4 шт.
3. Кнопка для управления движения 4шт.
4. Резистр  $22\Omega$  4 шт
5. Корпус
6. Плата расширения

Драйвер L298N используется для многофункционального управления двигателями постоянного тока. Схема модуля, состоящая из двух Н-мостов, позволяет подключать к нему один биполярный шаговый двигатель или одновременно два щёточных двигателя постоянного тока. При этом есть возможность изменять скорость и направление вращения моторов. Управление осуществляется путём подачи соответствующих сигналов на командные входы, выполненные в виде штыревых контактов. На рисунке показан внешний вид модуля с кратким описанием всех его составляющих.



### 1.3 Широтно-импульсная модуляция

В драйвере L289N *ENA*, *ENB* – контакты для активации/деактивации первого и второго двигателей или соответствующих обмоток ШД. Подача логической единицы на эти контакты разрешает вращение двигателей, а логический ноль – запрещает. Для изменения скорости вращения щёточных моторов на эти контакты подаётся ШИМ-сигнал.

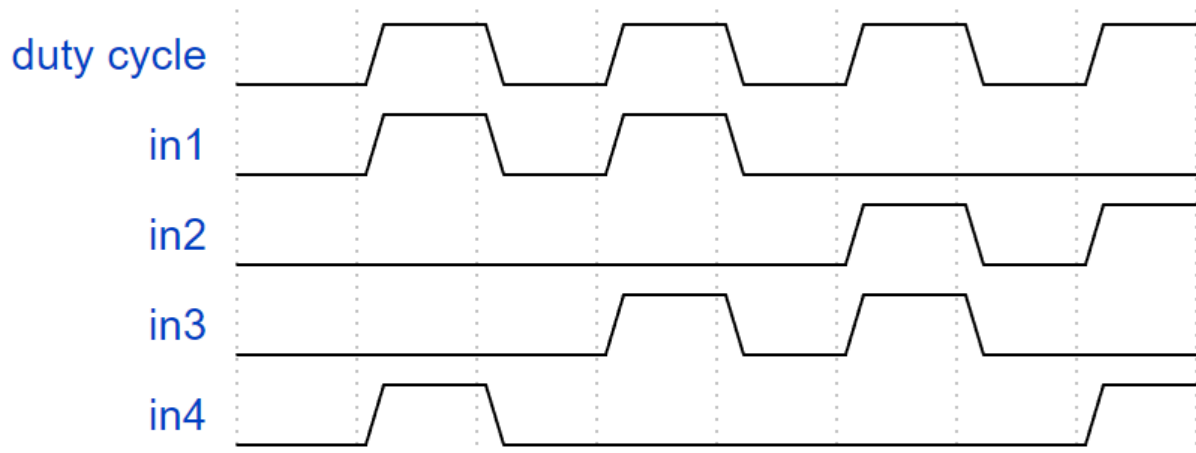
**ШИМ** - метод уменьшения средней мощности, передаваемой электрическим сигналом, путем эффективного разделения его на отдельные части. Среднее значение напряжения (и тока), подаваемого на нагрузку, регулируется быстрым включением и выключением переключателя между питанием и нагрузкой. Чем дольше переключатель включен по сравнению с периодами выключения, тем выше общая мощность, подаваемая на нагрузку.

```
1 pub fn set_frequency(&self, frequency: f64, duty_cycle: f64) -> Result<()>
```

- frequency - частота
- duty cycle - скважность, задается числом с плавающей точкой в промежутке между 0.0 и 1.0.

В нашей программе изначально значение скважности установлено на 0, что означает что на двигатели будет подаваться в среднем 0V. Никакого движения не происходит.

При нажатии на кнопку мы подаем сигналы на левые и правые двигатели и устанавливаем скважность на 1, что означает что на двигатели будет подаваться в среднем 5V.

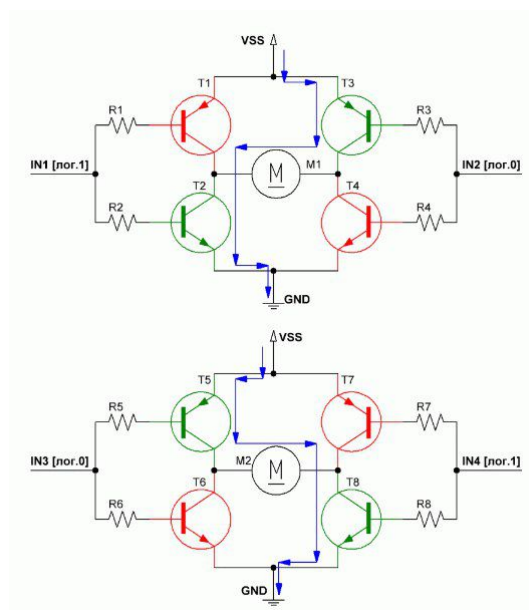


## 1.4 Питание

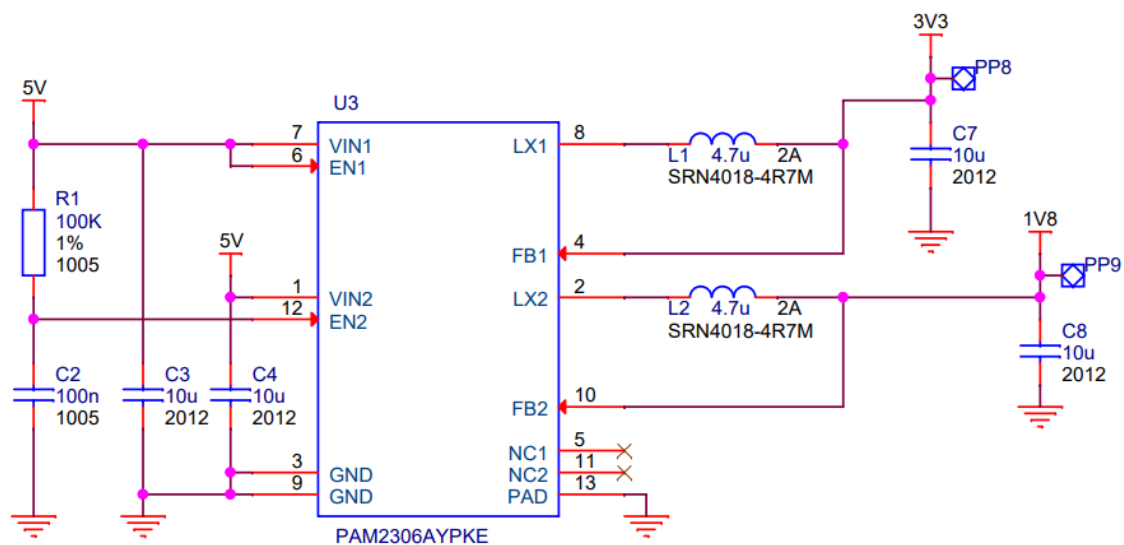
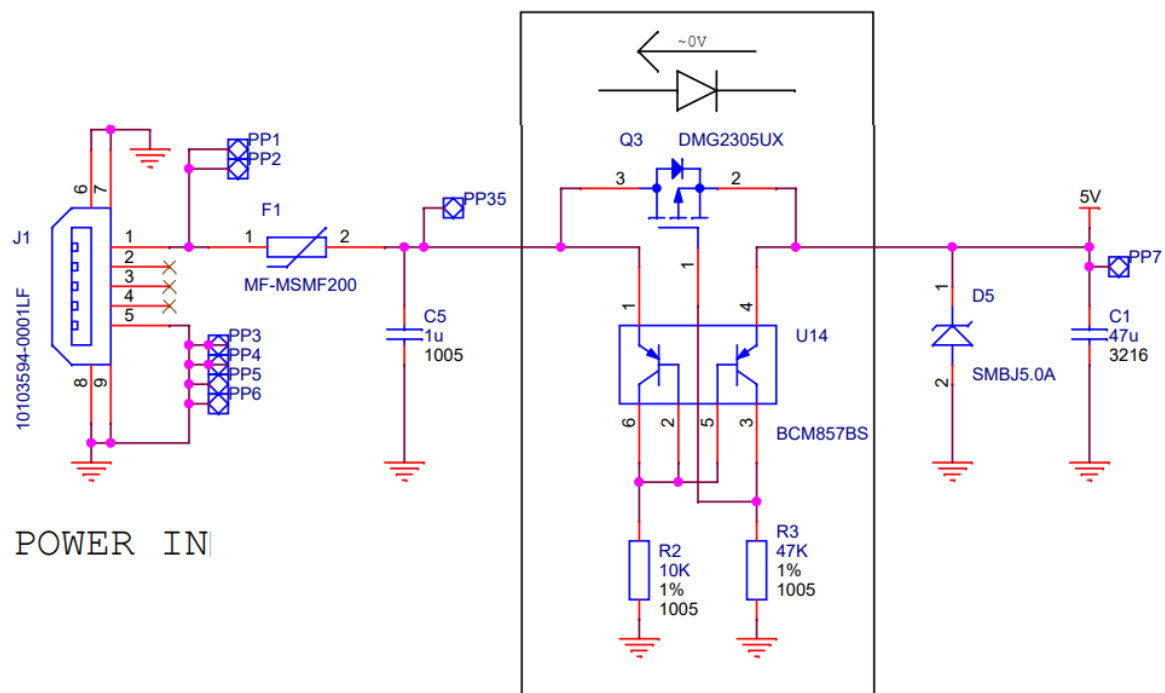
1. L289N - Логическое устройство - 5V
2. L289N - Моторы - 12V
3. Кнопка - 3.3V
4. Raspberry pi 2b+ - 5V

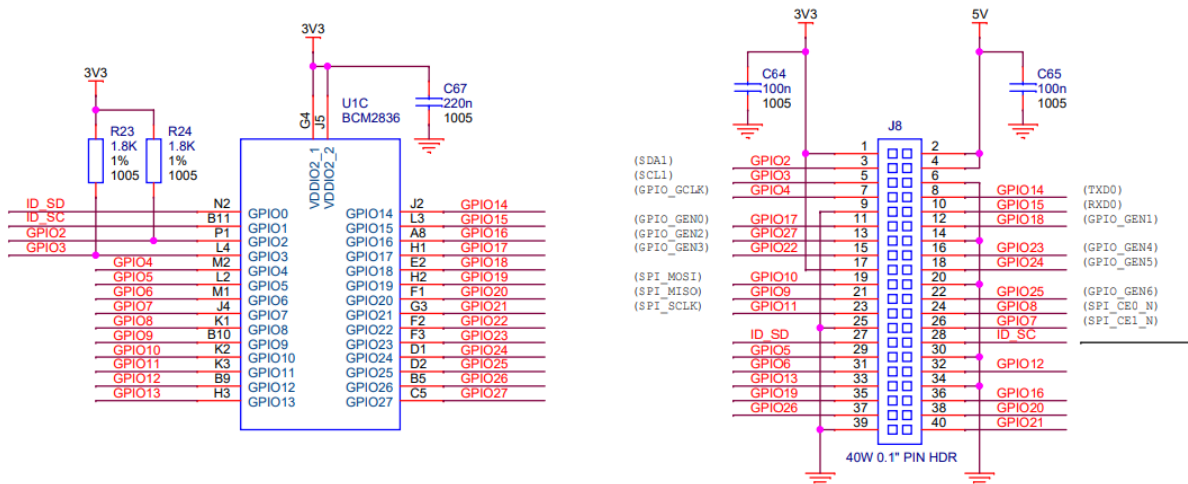
## 2 Схемы

### 2.1 Принципиальная схема драйвера L289N



## 2.2 Принципиальная схема Raspberry Pi

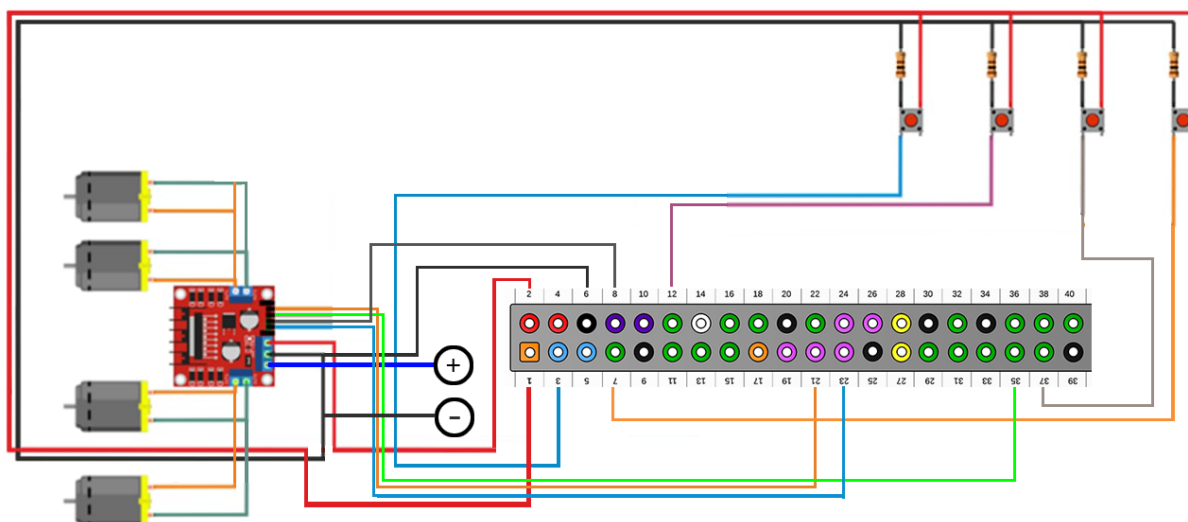




Полную принципиальную схему можно найти [здесь](#).

## 2.3 Схема подключений

Пин №1 подает 3.3 V на кнопки. Пин №2 подает 5 V питания на логическое устройство. Моторы питаются из внешнего зарядного устройства.



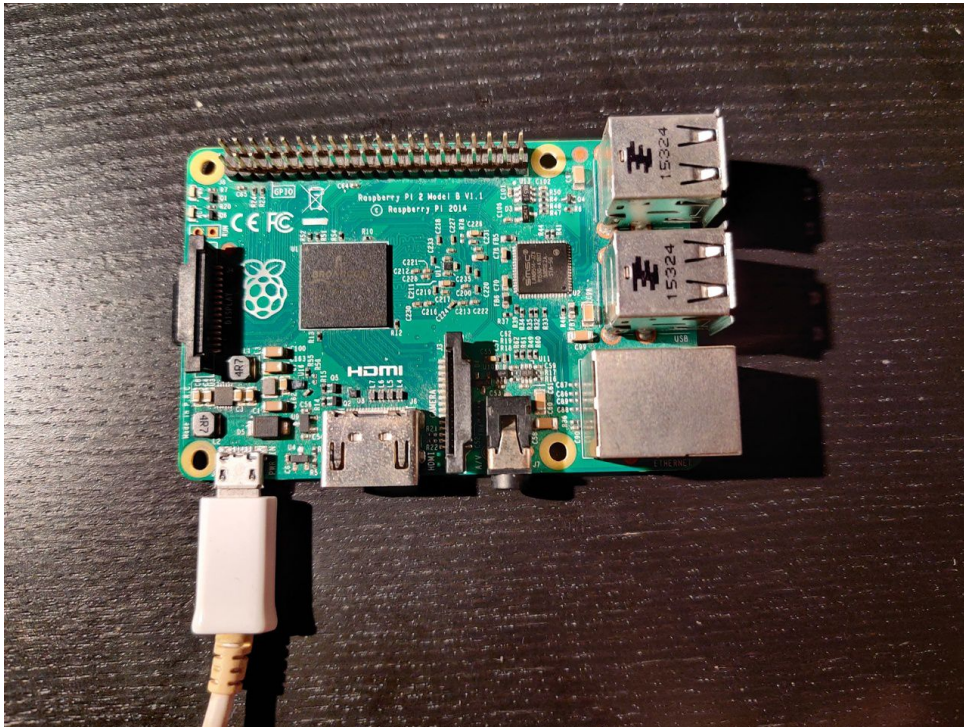


Raspberry Pi 2 Model B (J8 Header)					
GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1		2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3		4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5		6	Ground
7	GPIO 7 GPCLK0	7		8	GPIO 15 TxD (UART)
	Ground	9		10	GPIO 16 RxD (UART)
0	GPIO 0	11		12	GPIO 1 PCM_CLK/PWM0
2	GPIO 2	13		14	Ground
3	GPIO 3	15		16	GPIO 4
	3.3 VDC Power	17		18	GPIO 5
12	GPIO 12 MOSI (SPI)	19		20	Ground
13	GPIO 13 MISO (SPI)	21		22	GPIO 6
14	GPIO 14 SCLK (SPI)	23		24	GPIO 10 CE0 (SPI)
	Ground	25		26	GPIO 11 CE1 (SPI)
30	SDA0 (I2C ID EEPROM)	27		28	SCL0 (I2C ID EEPROM)
21	GPIO 21 GPCLK1	29		30	Ground
22	GPIO 22 GPCLK2	31		32	GPIO 26 PWM0
23	GPIO 23 PWM1	33		34	Ground
24	GPIO 24 PCM_FS/PWM1	35		36	GPIO 27
25	GPIO 25	37		38	GPIO 28 PCM_DIN
	Ground	39		40	GPIO 29 PCM_DOUT

**Attention!** The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>

Raspberry Pi питает от micro-USB, который подает 5V.





## 3 Программное обеспечение

### 3.1 Используемые инструменты

1. [Rust](#) - 1.25.1 - Язык программирования
2. [rppal](#) - 0.14.0 - Raspberry Pi Peripheral Access Library. Предоставляет доступ к GPIO у Raspberry Pi.

### 3.2 Код программы

```
1  /* main.rs */
2  use std::thread;
3  use std::time::Duration;
4  use color_eyre::Result;
5
6  use rppal::gpio::Gpio;
7  use rppal::pwm::{Pwm, Channel};
8  use rppal::system::DeviceInfo;
9
10 mod engine;
11 mod input;
12
13 fn main() -> Result<()> {
14     println!("Working on a {}", DeviceInfo::new()?.model());
15
16     let mut engine_left = engine::Engine{
17         forward_pin: Gpio::new()?.get(14)?.into_output(),
18         backward_pin: Gpio::new()?.get(15)?.into_output(),
19         pwm: Pwm::new(Channel::Pwm0)?,
20     };
21
22     let mut engine_right = engine::Engine{
23         forward_pin: Gpio::new()?.get(26)?.into_output(),
24         backward_pin: Gpio::new()?.get(13)?.into_output(),
25         pwm: Pwm::new(Channel::Pwm1)?,
26     };
27     let mut ddc = engine::DigitalDirectionController{
28         left: engine_left,
29         right: engine_right,
30     };
31     let mut input = input::ButtonInput{
32         forward: Gpio::new()?.get(8)?.into_input(),
33         backward: Gpio::new()?.get(1)?.into_input(),
34         left: Gpio::new()?.get(25)?.into_input(),
35         right: Gpio::new()?.get(7)?.into_input(),
36     };
```

```

37
38     loop {
39         let direction = input.get_direction();
40         println!("Direction: ⬆{:?}", direction);
41         ddc.direction(direction)?;
42         thread::sleep(Duration::from_millis(50));
43     }
44 }

```

```

1  /* input.rs */
2  use color_eyre::Result;
3  use rppal::gpio::InputPin;
4
5  use super::engine::Direction;
6
7  pub struct ButtonInput {
8      pub forward: InputPin,
9      pub backward: InputPin,
10     pub left: InputPin,
11     pub right: InputPin,
12 }
13
14 impl ButtonInput {
15     pub fn get_direction(&mut self) -> Direction {
16         if self.forward.is_high() {
17             return Direction::Forward;
18         };
19         if self.left.is_high() {
20             return Direction::Left;
21         };
22         if self.right.is_high() {
23             return Direction::Right;
24         };
25         if self.backward.is_high() {
26             return Direction::Backward;
27         };
28         Direction::Stop
29     }
30 }

```

```

1  /* engine.rs */
2  use color_eyre::Result;
3  use rppal::gpio::OutputPin;
4  use rppal::pwm::Pwm;
5
6  pub struct Engine {
7      pub forward_pin: OutputPin,

```

```

8     pub backward_pin: OutputPin,
9     pub pwm: Pwm,
10 }
11
12 impl Engine {
13     pub fn set(&mut self, direction: f64) -> Result<()> {
14         if !self.pwm.is_enabled()? {
15
16             self.pwm.set_frequency(20.0, 0.0)?;
17             self.pwm.enable()?;
18         }
19         let direction = direction.clamp(-1.0, 1.0);
20         if direction >= 0.0 {
21             self.forward_pin.set_high();
22             self.backward_pin.set_low();
23         } else {
24             self.forward_pin.set_low();
25             self.backward_pin.set_high();
26         }
27         self.pwm.set_duty_cycle(direction.abs())?;
28         Ok(())
29     }
30 }
31
32 impl Drop for Engine {
33     fn drop(&mut self) {
34         self.forward_pin.set_low();
35         self.backward_pin.set_low();
36     }
37 }
38
39 pub struct AnalogDirectionController {
40     pub left: Engine,
41     pub right: Engine,
42 }
43
44 impl AnalogDirectionController {
45     pub fn direction(&mut self, gradient: f64, radial: f64) -> Result<()> {
46         let gradient = gradient.clamp(-1.0, 1.0);
47         let radial = radial.clamp(-1.0, 1.0);
48         Ok(())
49     }
50 }
51
52 #[derive(Debug)]
53 pub enum Direction {

```

```

54     Stop,
55     Forward,
56     Backward,
57     Left,
58     Right
59 }
60
61 pub struct DigitalDirectionController {
62     pub left: Engine,
63     pub right: Engine,
64 }
65
66 impl DigitalDirectionController {
67     pub fn direction(&mut self, direction: Direction) -> Result<()> {
68         match direction {
69             Direction::Stop => {
70                 self.left.set(0.0)?;
71                 self.right.set(0.0)
72             },
73             Direction::Forward => {
74                 self.left.set(1.0)?;
75                 self.right.set(1.0)
76             },
77             Direction::Backward => {
78                 self.left.set(-1.0)?;
79                 self.right.set(-1.0)
80             },
81             Direction::Left => {
82                 self.left.set(-1.0)?;
83                 self.right.set(1.0)
84             },
85             Direction::Right => {
86                 self.left.set(1.0)?;
87                 self.right.set(-1.0)
88             },
89         }
90     }
91 }

```