

## **MSU ROVER TEAM**

**ОТКРЫТЫЕ БИБЛИОТЕКИ ДЛЯ АВТОНОМНОЙ НАВИГАЦИИ  
ШЕСТИКОЛЕСНОГО ПРОТОТИПА МАРСОХОДА (РОВЕРА) ПО УМЕРЕННО  
ПЕРЕСЕЧЁННОЙ НЕЗНАКОМОЙ МЕСТНОСТИ С ВИЗУАЛЬНЫМ  
РАСПОЗНАВАНИЕМ ЦЕЛИ НАВИГАЦИИ.**

**УТВЕРЖДАЮ:**  
Научный эксперт проекта, к.ф.-м.н.

\_\_\_\_\_ В.М. Буданов

.12.2025

**Открытая библиотека управления движения ровером на низком уровне.**

**Руководство программиста.**

## **ЛИСТ УТВЕРЖДЕНИЯ**

**MSUROVERTEAM-DC-V1.0.0**

**(открытая библиотека в сети Интернет)**

**ИСПОЛНИТЕЛИ:**

\_\_\_\_\_ А.А. Смирнов  
.12.2025

## **MSU ROVER TEAM**

**УТВЕРЖДЕНО**

**ОТКРЫТЫЕ БИБЛИОТЕКИ ДЛЯ АВТОНОМНОЙ НАВИГАЦИИ  
ШЕСТИКОЛЕСНОГО ПРОТОТИПА МАРСОХОДА (РОВЕРА) ПО УМЕРЕННО  
ПЕРЕСЕЧЁННОЙ НЕЗНАКОМОЙ МЕСТНОСТИ С ВИЗУАЛЬНЫМ  
РАСПОЗНАВАНИЕМ ЦЕЛИ НАВИГАЦИИ.**

**Открытая библиотека управления движения ровером на низком уровне.**

**Руководство программиста.**

**MSUROVERTEAM-DC-V1.0.0**

**(открытая библиотека в сети Интернет)**

**Листов .**

## **АННОТАЦИЯ.**

Открытая библиотека управления движения ровером на низком уровне разработана в рамках проекта «Разработки открытых библиотек для автономной навигации шестиколесного прототипа марсохода (ровера) по умеренно пересечённой незнакомой местности с визуальным распознаванием цели навигации». Проект выполнен на средства выделенные «Фондом содействия развитию малых форм предприятий в научно-технической сфере» (Фонд содействия инновациям) по договору предоставления гранта № 64ГУКодИИС13-D7/102402 от 23 декабря 2024г.

Под полностью автономным режимом навигации (движения) в данном проекте понимается режим, при котором ровер с Аккермановой геометрией поворота самостоятельно, без команд оператора (человека), передвигается по умеренно пересечённой и незнакомой местности по указателям направления движения до указателя конечной цели, может выполнить заранее запрограммированные действия у каждого указателя и самостоятельно вернуться обратно к месту старта. При этом оператор может просматривать на своем мониторе видеоизображения и телеметрию, передаваемые с ровера.

«Открытая библиотека управления движения ровером на низком уровне» предназначена для преобразования и исполнения высокоуровневых команд библиотеки навигации в управление приводами ровера на низком уровне и передачи значений датчиков ровера системе навигации и компьютерного зрения, с целью оптимизации маршрута в процессе автономного движения ровера.

«Открытая библиотека управления движения ровером на низком уровне» разработана на языке программирования C для STM32F4xxx и Python 3, для платформы ROS2 Humble (Robot Operating System 2 версии Humble).

## **СОДЕРЖАНИЕ.**

<b>АННОТАЦИЯ.</b> .....	2
<b>СОДЕРЖАНИЕ.</b> .....	3
<b>Общие сведения о программе.</b> .....	4
<b>Структура программы.</b> .....	5
<b>STM32 C модуль eureka_dc_lib.h</b> .....	5
<b>1. typedef struct stepper</b> .....	6
<b>2.typedef struct dc_motor</b> .....	9
<b>ROS2 Python3 Модуль eureka_movement_lib.py</b> .....	13
<b>1. RoverSteeringLookupTable</b> .....	13
<b>2. ackermann</b> .....	13
<b>Интеграция с высокоуровневым модулем навигации.</b> .....	20
<b>Пример использования библиотеки.</b> .....	20

## **Общие сведения о программе.**

«Открытая библиотека управления движения ровером на низком уровне» предназначена для управления на низком уровне шести, или четырехколесным ровером с четырьмя поворотными колесами. Если управляемый ровер четырехколесный, то библиотека позволяет реализовать его движение в любом направлении (омни-движение).

Эта же библиотека может использоваться при ручном дистанционном управлении ровером.

## **ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ.**

Библиотека состоит из двух модулей

1. STM32 C модуль `eureka_dc_lib.h`

- Язык программирования: C.
- Аппаратное обеспечение: STM32F4xxx.
- Уровень абстракции: HAL для STM32 версии F4.
- Назначение. Управление приводами ровера на низком уровне.

2. ROS2 Python3 Модуль `eureka_movement_lib.py`

- Язык программирования: Python3.
- Назначение.
  - 1 Снижение нагрузки с контроллеров.
  - 2 Реализация полной последовательности шагов (пайплайн) при управлении ровером.

**ВАЖНОЕ ЗАМЕЧАНИЕ.** Для работы библиотеки требуется корректная настройка аппаратно-программного интерфейса и корректная работа радиоэлектроники ровера. При изменении серии микроконтроллера работа библиотеки не может гарантироваться из-за серьезных отличий в функциях HAL и архитектурах чипов в разных поколениях STM32.

## **СПИСОК ОБЪЕКТОВ ДЛЯ ДОКУМЕНТИРОВАНИЯ.**

1. `eureka_askermann.py` - Высокоуровневый класс управления движением
2. `Neureka_dc_lib.h` - низкоуровневая библиотека управления движением

## Структура программы.

### STM32 C модуль eureka\_dc\_lib.h

- Модуль: eureka\_dc\_lib.h
- Язык программирования: C
- Аппаратное обеспечение: STM32F4xxx
- Уровень абстракции: HAL для STM32 версии F4

#### Используемые структуры данных:

1. stepper (typedef struct)
2. dc\_motor (typedef struct)

#### Функции для работы со структурой stepper:

```
int stepper_init(struct stepper *stp);           // initialize peripherals
int stepper_set(struct stepper *stp);           // apply en, dir, step to motor
int stepper_calculate_angle(struct stepper *stp); // calculate current stepper angle
                                                // from encoder
int stepper_logic(struct stepper *stp);         // interprets angles and sets target
                                                // velocity to arrive at target angle
                                                // and stay there
```

#### Функции для работы со структурой dc\_motor:

```
int dc_init(struct dc_motor *motor);           // initialize peripherals
int dc_set_pulse(struct dc_motor *motor);       // set pusle for a twin-channel DC driver
int dc_set_pulse_A1A2(struct dc_motor *motor); // set pulse for a single-channel DC driver
int dc_calculate_period(struct dc_motor *motor); // calculate time period that elapsed
int dc_calculate_velocity(struct dc_motor *motor); // calculate velocity after a period is
                                                // known
int dc_pid(struct dc_motor *motor);            // calculate motor inputs via PID
```

#### Функции общего назначения:

```
float abs_float(float val);                    // calculates absolute value of float
int sign_float(float val);                     // returns sign of given float
float clamp_float(float val, float min, float max); // clamps float to range
```

## 1. typedef struct stepper

Назначение. Управление сервоприводом, состоящим из шагового двигателя (возможно, редукторного), драйвера шагового двигателя с интерфейсом step/dir, и абсолютным аналоговым энкодером на выходном валу шагового двигателя.

Описание. Структура данных для работы с одним сервоприводом, состоящим из шагового двигателя (возможно, редукторного), драйвера шагового двигателя с интерфейсом step/dir, и абсолютным аналоговым энкодером на выходном валу шагового двигателя.

## Переменные.

- управляющие - через них отдаются команды сервоприводу. Можно изменять по ходу работы программы.

[illegible]

- измеряемые - в них содержится наиболее актуальное значение измеряемой переменной

[illegible]

- Конфигурационные - настройки привода. Задаются один раз до инициализации HAL и не меняются после.

```
int direction_fix; // Устанавливать в +-1 так, чтобы при положительной
// скорости вращения шаговика значение угла энкодера
// увеличивалось.

int raw_angle_offset; // Смещение нуля энкодера на уровне АЦП. Устанавливать
// так чтобы измеренный угол никогда не перескакивал
// между 360 и 0.

float hold_position_tolerance; // Допустимая ошибка угла в режиме удержания позиции.
// Если она превышена - привод переходит в режим движения
// чтобы скорректировать ошибку.

float drive_position_tolerance; // Допустимая ошибка угла в режиме движения. Должна быть
// меньше допустимой ошибки удержания. Рекомендуется
// hold_position_tolerance/2.

float gear_ratio; // Передаточное число редуктора шаговика.

int is_backdriveable; // ставить в нуль для самоблокирующихся (червячных)
// редукторов, иначе 1.

TIM_HandleTypeDef *timer; // таймер который будет генерировать ШИМ для STEP.
// Один таймер на один шаговик.
```

## 7 MSUROVERTEAM-DC-V1.0.0

```
uint16_t channel;           // используемый канал ШИМ таймера
GPIO_TypeDef* en_port;      // GPIO порт EN пина на драйвере
uint16_t en_pin;           // GPIO пин EN пина на драйвере
GPIO_TypeDef* dir_port;    // GPIO порт DIR пина на драйвере
uint16_t dir_pin;          // GPIO порт DIR пина на драйвере
ADC_HandleTypeDef* adc;     // АЦП который используется для энкодера
uint32_t adc_buffer[adc_buffer_len]; // Буфер значений АЦП. Нужно сделать "#define
                                     // adc_buffer_len (x)" перед подключением
                                     // библиотеки.
```

- прочие - используются для отладки и внутренних вычислений (не редактировать!!!)

```
int precalc_period;
float current_velocity;
long ticks_last;
long ticks_total;
```

### Функции для работы со структурой.

```
int stepper_init(struct stepper *stp); // initialize peripherals
```

Назначение. Инициализация периферии микроконтроллера в соответствии с конфигурационными параметрами.

Входные переменные. Указатель на нужную структуру struct stepper

Возвращаемое значение: всегда 1

Применение. После HAL\_Init() и определения всех конфигурационных параметров в объекте stepper, до всех остальных функций, работающих с соответствующим объектом stepper.

```
int stepper_set(struct stepper *stp); // apply en, dir, step to motor
```

Назначение. применяет текущие значения current velocity и enable на привод

Входные переменные. Указатель на нужную структуру struct stepper

Возвращаемое значение: всегда 1

Применение. Для отдачи команды на привод.

```
int stepper_calculate_angle(struct stepper *stp); // calculate current stepper angle
                                                  // from encoder
```

Назначение. Обновляет значение current\_position по показаниям энкодера

Входные переменные. Указатель на нужную структуру struct stepper

Возвращаемое значение: всегда 1

Применение. Для получения актуальных показаний датчика



```
int stepper_logic(struct stepper *stp); // interprets angles and sets target velocity to  
                                         // arrive at target angle and stay there
```

Назначение. Логика управления двигателем. Исходя из текущего угла `current_position` и ограничения скорости `target_velocity` считает `current_velocity`

Входные переменные. Указатель на нужную структуру `struct stepper`

Возвращаемое значение: всегда 1

Применение. После `stepper_calculate_angle`, до `stepper_set`

### **Пример применения структуры данных для работы с шаговым сервоприводом.**

```
//объявить глобально  
#define stp_len 2  
struct stepper stp[stp_len];  
  
//инициализация в main()  
for (c = 0; c < stp_len; ++c) {  
    stp[c].enable = 1;  
    stp[c].hold_position_tolerance = 1.0;  
    stp[c].drive_position_tolerance = 0.5;  
    stp[c].target_position = 0.0;  
    stp[c].target_velocity = 0.0;  
}  
stp[0].dir_port = Stepper1_DIR_GPIO_Port;  
stp[0].dir_pin = Stepper1_DIR_Pin;  
stp[0].en_port = Stepper1_EN_GPIO_Port;  
stp[0].en_pin = Stepper1_EN_Pin;  
stp[0].timer = &htim9;  
stp[0].raw_angle_offset = 300;  
stp[0].adc = &hadc2;  
stp[0].channel = TIM_CHANNEL_1;  
stp[0].gear_ratio = 100.0;  
stp[0].direction_fix = -1;  
  
stp[1].dir_port = Stepper2_DIR_GPIO_Port;  
stp[1].dir_pin = Stepper2_DIR_Pin;  
stp[1].en_port = Stepper2_EN_GPIO_Port;  
stp[1].en_pin = Stepper2_EN_Pin;  
stp[1].timer = &htim12;  
stp[1].raw_angle_offset = -350;  
stp[1].adc = &hadc1;  
stp[1].channel = TIM_CHANNEL_1;
```

```
stp[1].gear_ratio = 100.0;
stp[1].direction_fix = -1;

for (c = 0; c < stp_len; ++c) {
    stepper_init(&stp[c]);
}

//управление приводом в периодически-исполняемой функции (привязанной к таймеру,
например):
for (c = 0; c < stp_len; ++c) {
    stepper_calculate_angle(&stp[c]);
    stepper_logic(&stp[c]);
    stepper_set(&stp[c]);
}
```

## 2.typedef struct dc\_motor

Назначение. Управление сервоприводом, состоящим из двигателя постоянного тока (возможно, с редуктором), драйвера двигателя постоянного тока с интерфейсом двухканального ШИМ или с интерфейсом одноканального ШИМ и пинами A1A2 задающими направление вращения, и двухканальным инкрементальным энкодером на валу электромотора.

Описание. Структура данных для работы с одним сервоприводом, состоящим из двигателя постоянного тока (возможно, с редуктором), драйвера двигателя постоянного тока с интерфейсом двухканального ШИМ или с интерфейсом одноканального ШИМ и пинами A1A2 задающими направление вращения, и двухканальным инкрементальным энкодером на валу электромотора.

### Переменные.

- управляющие - через них отдаются команды сервоприводу. Можно изменять по ходу работы программы.

```
float target_velocity;           // требуемая скорость вращения выходного вала
                                 // двигателя в градусах в секунду
```

- измеряемые - в них содержится наиболее актуальное значение измеряемой переменной

```
float raw_velocity;              // неотфильтрованная измеренная скорость в градусах
                                 // в секунду
float filtered_velocity;         // отфильтрованная измеренная скорость в градусах
                                 // в секунду
float filtered_velocity_prev;    // отфильтрованная измеренная в предыдущий момент
                                 // времени скорость
float acceleration_filtered;     // отфильтрованное ускорение, град/с^2
float acceleration;              // неотфильтрованное ускорение, град/с^2
```

- Конфигурационные - настройки привода. Задаются один раз до инициализации HAL и не меняются после.

```
float voltage_limit;           // от 0 до 1, задает верхнее ограничение скажности ШИМ
float gear_ratio;              // передаточное число редуктора
float encoder_resolution;      // разрешение энкодера, количество "щелчков" на оборот
                                // вала электродвигателя
float low_pass_gain;           // коэффициент фильтра низких частот
float velocity_correction_gain; // множитель скорости, подбирается экспериментально
                                // для наиболее точной работы одометра,
                                // по умолчанию задавать 1.
float gain_p, gain_i, gain_d;   // коэффициенты ПИД регулятора
float integral_fade_gain;       // интегральная ошибка умножается на это число на
                                // каждом такте контроллера. Рекомендуется 0.95-0.99
TIM_HandleTypeDef *pwm_timer;   // Таймер ШИМ, может быть один на несколько приводов
uint16_t pwm_channel;           // Канал ШИМ для одноканальных A1A2 драйверов.
                                // Для двухканальных используются каналы 1 и 2.
TIM_HandleTypeDef *encoder_timer; // Таймер в режиме энкодера, свой на каждый двигатель
TIM_HandleTypeDef *clock_timer; // системный таймер для измерения времени.
                                // 100КГц после прескейлера.
GPIO_TypeDef* A1_port;         // порты и пины A1A2 для соответствующего драйвера
uint16_t A1_pin;
GPIO_TypeDef* A2_port;
uint16_t A2_pin;
```

- прочие - используются для отладки и внутренних вычислений (не редактировать!!!)

```
float pid_value;
float pulse;
int clock_prev;
int encoder_prev;
float integral_error;
float period_seconds;
int encoder_ticks_debug;
```

### Функции для работы со структурой.

```
int dc_init(struct dc_motor *motor);
```

Назначение. Инициализация периферии микроконтроллера в соответствии с конфигурационными параметрами.

Входные переменные. Указатель на нужную структуру struct dc\_motor

Возвращаемое значение: всегда 1

Применение. После HAL\_Init() и определения всех конфигурационных параметров в объекте dc\_motor, до всех остальных функций работающих с соответствующим объектом dc\_motor.

```
int dc_set_pulse(struct dc_motor *motor);          // set pulse for a twin-channel DC driver
```

Назначение. Применяет текущие значения pulse для двухканального ШИМ драйвера

Входные переменные. Указатель на нужную структуру struct dc\_motor

Возвращаемое значение: всегда 1

Применение. Для отдачи команды на привод.

```
int dc_set_pulse_A1A2(struct dc_motor *motor); // set pulse for a single-channel DC driver
```

Назначение. Применяет текущие значения pulse для одноканального ШИМ драйвера с A1A2

Входные переменные. Указатель на нужную структуру struct dc\_motor

Возвращаемое значение: всегда 1

Применение. Для отдачи команды на привод

```
int dc_calculate_period(struct dc_motor *motor); // calculate time period that elapsed
```

Назначение. Обновляет значение period\_seconds, это прошедшее время в секундах с предыдущего вызова этой функции

Входные переменные. Указатель на нужную структуру struct dc\_motor

Возвращаемое значение: всегда 1

Применение. Значение period\_seconds нужно для расчета скорости

```
int dc_calculate_velocity(struct dc_motor *motor); // calculate velocity after a period is  
                                                    // known
```

Назначение. Считает raw\_velocity и filtered\_velocity, требует корректный period\_seconds

Входные переменные. Указатель на нужную структуру struct dc\_motor

Возвращаемое значение: всегда 1

Применение. Для вычисления скорости, сразу после dc\_calculate\_period;

```
int dc_pid(struct dc_motor *motor);                // calculate motor inputs via PID
```

Назначение. Считает pid\_value с помощью ПИД регулятора.

Входные переменные: Указатель на нужную структуру struct stepper.

Возвращаемое значение: всегда 1

Применение. Для вычисления подаваемого на привод напряжения.

### Функции для работы со структурой.

```
//объявить глобально
#define dc_len 2
struct dc_motor dc[dc_len];

//инициализация в main()
for (c = 0; c < dc_len; ++c) {
    dc[c].target_velocity = 0.0;
    dc[c].raw_velocity = 0.0;
    dc[c].filtered_velocity = 0.0;
    dc[c].filtered_velocity_prev = 0.0;
    dc[c].clock_prev = 0;
    dc[c].acceleration = 0.0;
    dc[c].acceleration_filtered = 0.0;
    dc[c].gear_ratio = 100;
    dc[c].encoder_resolution = 34;
    dc[c].low_pass_gain = 0.9;
    dc[c].gain_p = gain_p;
    dc[c].gain_i = gain_i;
    dc[c].gain_d = gain_d;
    dc[c].integral_fade_gain = 0.99;
    dc[c].integral_error = 0.0;
    dc[c].pulse = 0.0;
    dc[c].clock_timer = &htim10;
    dc[c].voltage_limit = 0.95;
}

dc[0].pwm_timer = &htim4;
dc[0].pwm_channel = TIM_CHANNEL_2;
dc[0].encoder_timer = &htim2;
dc[0].velocity_correction_gain = -0.5;
dc[0].A1_port = A1_GPIO_Port;
dc[0].A1_pin = A1_Pin;
dc[0].A2_port = A2_GPIO_Port;
dc[0].A2_pin = A2_Pin;

dc[1].pwm_timer = &htim4;
dc[1].pwm_channel = TIM_CHANNEL_1;
dc[1].encoder_timer = &htim1;
dc[1].velocity_correction_gain = -0.5;
dc[1].A1_port = B1_GPIO_Port;
dc[1].A1_pin = B1_Pin;
dc[1].A2_port = B2_GPIO_Port;
dc[1].A2_pin = B2_Pin;
```

```
for (c = 0; c < dc_len; ++c) {
    dc_init(&dc[c]);
    dc_calculate_period(&dc[c]);
}

//управление приводом в периодически-исполняемой функции (привязанной к таймеру,
например):
for (c = 0; c < dc_len; ++c) {
    dc_calculate_period(&dc[c]);
    dc_calculate_velocity(&dc[c]);
    dc_pid(&dc[c]);
    dc[c].pulse = clamp_float(dc[c].pid_value, -voltage_limit, voltage_limit);
    dc_set_pulse_A1A2(&dc[c]);
}
```

## ROS2 Python3 Модуль eureka\_movement\_lib.py

- Модуль: eureka\_movement\_lib.py
- Язык программирования: Python3
- Используемые классы:
  1. RoverSteeringLookupTable
  2. Ackermann

### 1. RoverSteeringLookupTable

Описание. Это вспомогательный класс, его основная задача - снизить вычислительную нагрузку, заранее генерируя таблицы углов и скоростей поворота колес для каждого радиуса поворота ровера. Пользователю библиотеки взаимодействовать с этим классом нет необходимости. Объект класса RoverSteeringLookupTable создается при инициализации основного класса ackermann.

### 2. ackermann

Описание. Основной класс данного модуля. В нем реализован полный пайплайн управления ровером. Чтобы запустить ROS2 Python3 Модуль eureka\_movement\_lib.py необходимо создать объект класса Ackermann. Все необходимые процедуры запускаются автоматически в конструкторе класса. Реализовать управление движением реального устройства как простой набор функций не представляется возможным, так как любой аппаратный драйвер должен постоянно держать связь с низкоуровневой электроникой. Команды должны отправляться с определенной частотой, обратная связь должна считываться своевременно, и так далее.

Ниже приведен отрывок программного кода, обеспечивающего связь высокоуровневого вычислителя NVIDIA JETSON ORIN NX и низкоуровневого контроллера STM32F407DISCOVERY на интерфейсу USB\_FS. Полный код можно найти в файлах библиотеки.

### ROS2 Python3 нода на NVIDIA JETSON ORIN NX

```
#!/usr/bin/env python3

#Developed by Andrei Smirnov. 2024
#MSU Rover Team.
from sensor_msgs.msg import JointState
import numpy as np
import rclpy
from rclpy.node import Node
import serial
from neuron import h

print("abracadabra")

class usb_movement(Node):
    def __init__(self):
        print("init")
        super().__init__('usb_movement')
        self.sub = self.create_subscription(JointState, "dc_commands", self.dc_callback,
10)

        self.sub_2 = self.create_subscription(JointState, "stepper_commands",
self.stepper_callback, 10)
        self.sub_3 = self.create_subscription(JointState, "wheel_settings",
self.settings_callback, 10)
        self.sub_4 = self.create_subscription(JointState, "light_commands",
self.light_callback, 10)
        self.sub_5 = self.create_subscription(JointState, "arm_commands",
self.gripper_callback, 10)
        self.pub = self.create_publisher(JointState, "wheel_states", 10)
        #commands
        self.heartbeat = 1
        self.control_mode = 2
        self.power_saving = 0
        self.gain_p = 0.0040
        self.gain_i = 0.0020
        self.gain_d = 0.0
        self.voltage_limit = 0.95
        self.stepper_pos_com = [0.] * 4
        self.stepper_vel_com = [0.] * 4
```

15  
MSUROVERTEAM-DC-V1.0.0

```
self.dc_vel_com = [0.] * 4
self.steering_corrections = [0.0, 0.0, 0.0, 0.0]
self.gripper_pulse = 0.0
self.light_pulse = 0.0
#feedback
self.stepper_pos_fb = [0.] * 4
self.dc_vel_fb = [0.] * 4
#flags
self.left_connected = 0
self.right_connected = 0
self.x = [h.ref(0) for i in range(20)]
self.command_format = '''global: hrtb=%d, cm=%d, ps=%d, vl=%.2f, gp=%.4f, gi=%f,
gd=%.4f\r\n\
wheel1: stp_pos=%.2f, stp_vel=%.2f, dc_vel=%.2f\r\n\
wheel2: stp_pos=%.2f, stp_vel=%.2f, dc_vel=%.2f\r\n\
gp: pulse=%.2f\r\n\
__end__'''
    self.reply_format = '''wheel1: stepper_pos=%f, dc_vel=%f\r\n\
wheel2: stepper_pos=%f, dc_vel=%f\r\n\
__end__'''
    timer_period = 0.05 # seconds
    self.heartbeat_counter = 0
    self.timer = self.create_timer(timer_period, self.send)
    self.timer3 = self.create_timer(.1, self.heartbeat_function)
    self.send()
    self.get_logger().info("usb_movement Started!")
def __del__(self):
    self.get_logger().info("usb_movement Killed!")

def send(self):
    if(self.left_connected < 1 or not self.left.isOpen()):
        try:
            self.left = serial.Serial('/dev/dc_left', 9600, timeout=1)
            self.left_connected = 1
        except:
            self.get_logger().warning("No USB FS Connection to Left Side!")
            self.left_connected = 0
    if(self.right_connected < 1 or not self.right.isOpen()):
        try:
            self.right = serial.Serial('/dev/dc_right', 9600, timeout=1)
            self.right_connected = 1
        except:
            self.get_logger().warning("No USB FS Connection to Right Side!")
            self.right_connected = 0
```



16  
MSUROVERTEAM-DC-V1.0.0

```
        if(self.left_connected > 0 and self.right_connected > 0 and self.left.isOpen() and
self.right.isOpen()):
            try:
                message = self.command_format % (self.heartbeat, self.control_mode,
self.power_saving, self.voltage_limit, self.gain_p, self.gain_i, self.gain_d,
                                                self.stepper_pos_com[1],
self.stepper_vel_com[1], -self.dc_vel_com[1],
                                                self.stepper_pos_com[3],
self.stepper_vel_com[3], -self.dc_vel_com[3],
                                                -self.light_pulse)

                print(message)
                print(self.right.write(bytes(message, encoding='utf8')))
                reply = self.right.read_until(str.encode("__end__")).decode('utf-8')
                print(reply)

                num = h sscanf(reply, self.reply_format, self.x[0], self.x[1], self.x[2],
self.x[3])

                self.stepper_pos_fb[1] = float(self.x[0][0])
                self.stepper_pos_fb[3] = float(self.x[2][0])
                self.dc_vel_fb[1] = -float(self.x[1][0])
                self.dc_vel_fb[3] = -float(self.x[3][0])
                print(self.stepper_pos_fb)
                print(len(self.command_format))
            except:
                self.get_logger().warning("Failed to send or receive message for right")
                self.right_connected = 0
            try:
                message = self.command_format % (self.heartbeat, self.control_mode,
self.power_saving, self.voltage_limit, self.gain_p, self.gain_i, self.gain_d,
                                                self.stepper_pos_com[0],
self.stepper_vel_com[0], -self.dc_vel_com[0],
                                                self.stepper_pos_com[2],
self.stepper_vel_com[2], -self.dc_vel_com[2],
                                                -self.gripper_pulse)

                print(message)
                print(self.left.write(bytes(message, encoding='utf8')))
                reply = self.left.read_until(str.encode("__end__")).decode('utf-8')
                print(reply)
                num = h sscanf(reply, self.reply_format, self.x[0], self.x[1], self.x[2],
self.x[3])

                self.stepper_pos_fb[0] = float(self.x[0][0])
                self.stepper_pos_fb[2] = float(self.x[2][0])
                self.dc_vel_fb[0] = -float(self.x[3][0])
                self.dc_vel_fb[2] = -float(self.x[1][0])
```

```
        message = JointState()
        message.name = ['FL', 'FR', 'RL', 'RR']
        message.header.stamp = self.get_clock().now().to_msg()
        message.velocity = self.dc_vel_fb
        message.effort = [0.] * 6
        message.position = list(np.array(self.stepper_pos_fb) -
np.array(self.steering_corrections));
        #         message.position[0] -= 5.0
        #         message.position[1] += 1.0
        self.pub.publish(message)
    except:
        self.left_connected = 0

        self.get_logger().warning("Failed to send or receive messages from left")
```

## C код на STM32F407DISCOVERY

```
//usb variables
char usbd_ch[2048];
char reply_buffer[2048];
int usb_flag;

int sscanf_flag = 0;
int usb_ctr = 0;

char command_message_format[] =
    "global: hrtb=%d, cm=%d, ps=%d, vl=%f, gp=%f, gi=%f, gd=%f\r\n\
wheel1: stp_pos=%f, stp_vel=%f, dc_vel=%f\r\n\
wheel2: stp_pos=%f, stp_vel=%f, dc_vel=%f\r\n\
gp: pulse=%f\r\n\
__end__";
char reply_message_format[] =
    "wheel1: stepper_pos=%2f, dc_vel=%2f\r\n\
wheel2: stepper_pos=%2f, dc_vel=%2f\r\n\
__end__";

float gain_p = 0.0075, gain_i = 0.005, gain_d = 0.0;
float voltage_limit = 0.5;

// system config variables
uint8_t control_mode = CONTROL_MODE_RAW;
uint8_t power_saving = POWER_SAVING_OFF;
uint8_t heartbeat = HEARTBEAT_OFF;

int heartbeat_counter = 0;
struct dc_motor dc[dc_len];
```

```
struct dc_motor gp_driver;
struct stepper stp[stp_len];

int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */

    /* MCU Configuration-----*/
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */
    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */
    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_TIM4_Init();
    MX_TIM5_Init();
    MX_TIM7_Init();
    MX_ADC1_Init();
    MX_TIM9_Init();
    MX_TIM10_Init();
    MX_USB_DEVICE_Init();
    MX_TIM8_Init();
    MX_CAN1_Init();
    MX_TIM1_Init();
    MX_TIM2_Init();
    MX_TIM3_Init();
    MX_TIM12_Init();
    MX_ADC2_Init();
    MX_ADC3_Init();

    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        if (usb_flag > 0) {
            heartbeat_counter = 0;
        }
    }
}
```

```
usb_ctr = strlen(usbd_ch);
sscanf_flag = sscanf(usbd_ch, command_message_format, &heartbeat,
                    &control_mode, &power_saving, &voltage_limit, &gain_p, &gain_i,
&gain_d,
                    &(stp[0].target_position), &(stp[0].target_velocity),
&(dc[0].target_velocity),
                    &(stp[1].target_position), &(stp[1].target_velocity),
&(dc[1].target_velocity),
                    &(gp_driver.pulse));
//      vel_2_targ = vel_3_targ;
sprintf(reply_buffer, reply_message_format, stp[0].current_position,
dc[0].raw_velocity,
                    stp[1].current_position, dc[1].raw_velocity);
CDC_Transmit_FS((uint8_t*) reply_buffer, strlen(reply_buffer));

usb_flag = 0;
for (c = 0; c < dc_len; ++c) {
    dc[c].gain_p = gain_p;
    dc[c].gain_i = gain_i;
    dc[c].gain_d = gain_d;
}

}

/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}
```

**Следует понимать, что любой аппаратно-программный интерфейс требует корректной настройки и подключения. Это область радиоэлектроники. Выше приведен пример применения одного из десятков возможных интерфейсов. Это НЕ ЧАСТЬ программной библиотеки управления движения ровером на низком уровне. Выбирать конкретный аппаратный интерфейс для программной библиотеки бессмысленно, это сделает ее очень узкоспециализированный и практически неприменимой, поэтому эта часть остается на усмотрение пользователя.**

### **Интеграция с высокоуровневым модулем навигации.**

Модуль eureka\_askermann.ru получает команды в стандартном формате /cmd\_vel и передает угловые скорости колёс для дальнейшего расчета колесной одометрии в wheel\_states.

### **Пример использования библиотеки.**

Примеры использования библиотеки приведены в разделе [«Структура программы»](#).