

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению лабораторных работ
по дисциплине «Программная инженерия встроенных систем для автономных
платформ» для студентов направления подготовки 09.03.04 Программная инженерия
Направленность (профиль)
«Инженерия сложных программных систем»

Ставрополь 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
Лабораторная работа 1. DMA + АЦП в циркулярном режиме для многоканального сбора.....	4
Лабораторная работа 2. Quadrature Encoder: измерение скорости и направления вращения.....	7
Лабораторная работа 3. Одометрия для дифференциального привода.....	10
Лабораторная работа 4. Комплементарный фильтр для слияния акселерометра и гироскопа.....	13
Лабораторная работа 5. Расширенный фильтр Калмана (EKF) для ориентации (IMU + энкодеры).....	16
Лабораторная работа 6. Передача данных с STM32 на Raspberry Pi через UART.....	19
Лабораторная работа 7. Приём команд управления с Raspberry Pi на STM32 (cmd_vel).....	22
Лабораторная работа 8. Установка ROS 2 на Raspberry Pi и настройка micro-ROS агента.....	25
Лабораторная работа 9. micro-ROS Publisher: публикация одометрии на STM32.....	28
Лабораторная работа 10. Подключение лидара к Raspberry Pi и публикация /scan.....	31
Лабораторная работа 11. Обработка сканов лидара: избегание препятствий.....	34
Лабораторная работа 12. Подключение камеры и базовая обработка изображений (Canny).....	37
Лабораторная работа 13. Детекция объектов: YOLOv8n на Raspberry Pi.....	40
Лабораторная работа 14. Слияние лидара и камеры: комбинированное избегание препятствий.....	43
Лабораторная работа 15. PID-регулятор скорости на STM32 с обратной связью от энкодеров.....	46
Лабораторная работа 16. Полный стек навигации: Nav2 + micro-ROS.....	49
Лабораторная работа 17. SLAM: картографирование с использованием лидара.....	52
Лабораторная работа 18. Итоговый проект: полностью автономный робот.....	55

ВВЕДЕНИЕ

Методические указания предназначены для студентов направления подготовки 09.03.04 «Программная инженерия» (направленность «Разработка и сопровождение программного обеспечения»). Дисциплина «Программная инженерия встроенных систем для автономных платформ» изучается в рамках подготовки специалистов в области разработки программного обеспечения для встроенных и автономных систем.

Дисциплина формирует у студентов практические навыки разработки бортового программного обеспечения: от продвинутой периферии микроконтроллера STM32 (DMA, ADC, энкодеры) через слияние сенсорных данных (комплементарный фильтр, EKF), взаимодействие с одноплатным компьютером (UART, micro-ROS) до интеграции лидара, камеры, YOLOv8 и полноценной автономной навигации (Nav2, SLAM).

Цель лабораторного практикума — закрепить теоретические знания и сформировать устойчивые практические навыки разработки встроенных систем для автономных платформ.

Практикум построен по принципу последовательного усложнения: от периферии микроконтроллера к связи МК — ОПК, далее к обработке сенсорных данных, управлению и полной интеграции в автономный стек навигации.

Каждая лабораторная работа включает цель, теоретическое обоснование, пример выполнения, варианты индивидуальных заданий, требования к отчёту и вопросы для защиты.

Лабораторная работа 1.

DMA + АЦП В ЦИРКУЛЯРНОМ РЕЖИМЕ ДЛЯ МНОГОКАНАЛЬНОГО СБОРА

Цель и содержание работы: Настройка аналого-цифрового преобразователя (АЦП) с прямым доступом к памяти (DMA) в циркулярном режиме для непрерывного сбора данных с трёх аналоговых датчиков. Освоение методов снижения нагрузки на центральный процессор (CPU) при многоканальном сборе и обработке сигналов.

Теоретическое обоснование

Аналого-цифровой преобразователь (АЦП) микроконтроллера преобразует непрерывное аналоговое напряжение в дискретный цифровой код. При многоканальном сборе используется режим сканирования: АЦП последовательно опрашивает заданные каналы, сохраняя результаты в общий регистр данных. Для непрерывного преобразования включается режим непрерывного преобразования (continuous conversion), при котором после завершения преобразования одного канала автоматически запускается следующее.

Прямой доступ к памяти (DMA) позволяет пересылать данные из регистра данных АЦП в память (буфер) без участия CPU. Циркулярный режим (circular mode) обеспечивает автоматическое заикливание пересылок: после заполнения буфера указатель DMA возвращается в начало, и перезапись происходит поверх старых данных. Это идеально подходит для потокового сбора данных.

Прерывание по заполнению половины буфера (half-transfer interrupt) сигнализирует CPU о том, что первая половина буфера заполнена и готова к обработке, в то время как DMA продолжает заполнять вторую половину.

Такой подход (двойная буферизация) позволяет обрабатывать данные без остановки потока и без конфликта доступа.

Усреднение значений уменьшает влияние шумов. Для трёх каналов буфер организуется как массив структур или плоский массив, где каждые три последовательных элемента соответствуют трём каналам одного такта сбора. Усреднение выполняется по блоку из N отсчётов (например, за половину буфера). Результаты выводятся по интерфейсу UART в виде строк с разделителями.

Пример

Рассмотрим типовую настройку для микроконтроллера STM32F407 с использованием библиотеки HAL. Конфигурация: АЦП1, каналы 0, 1, 2 (соответствуют выводам PA0, PA1, PA2). Режим сканирования, непрерывное преобразование. DMA2, поток 0, циркулярный режим, размер буфера 256 значений (на три канала – $256/3$ не целое, поэтому берём 255, но для простоты используем 256, при этом последний отсчёт будет лишним, корректнее использовать размер, кратный числу каналов). В примере размер буфера 240 (80 выборок на канал). Прерывание по половинному заполнению. Усреднение каждых 20 отсчётов на канал. Вывод по UART2.

Код инициализации:

```
void MX_ADC1_Init(void)
{
    ADC_ChannelConfTypeDef sConfig = {0};
    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode = ENABLE;
    hadc1.Init.ContinuousConvMode = ENABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 3;
    hadc1.Init.DMAContinuousRequests = ENABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    HAL_ADC_Init(&hadc1);

    sConfig.Channel = ADC_CHANNEL_0;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
```

```

    HAL_ADC_ConfigChannel(&hadc1, &sConfig);

    sConfig.Channel = ADC_CHANNEL_1;
    sConfig.Rank = 2;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);

    sConfig.Channel = ADC_CHANNEL_2;
    sConfig.Rank = 3;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);
}

void MX_DMA_Init(void)
{
    __HAL_RCC_DMA2_CLK_ENABLE();
    hdma_adc1.Instance = DMA2_Stream0;
    hdma_adc1.Init.Channel = DMA_CHANNEL_0;
    hdma_adc1.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_adc1.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc1.Init.MemInc = DMA_MINC_ENABLE;
    hdma_adc1.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc1.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc1.Init.Mode = DMA_CIRCULAR;
    hdma_adc1.Init.Priority = DMA_PRIORITY_HIGH;
    HAL_DMA_Init(&hdma_adc1);
    __HAL_LINKDMA(&hadc1, DMA_Handle, hdma_adc1);

    HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
}

```

Буфер и переменные:

```

#define BUFFER_SIZE 240
uint16_t adc_buffer[BUFFER_SIZE];
volatile uint8_t half_done_flag = 0;

Обработчик прерывания DMA:
void DMA2_Stream0_IRQHandler(void)
{
    if (__HAL_DMA_GET_FLAG(&hdma_adc1, DMA_FLAG_HTIF0_4))
    {
        __HAL_DMA_CLEAR_FLAG(&hdma_adc1, DMA_FLAG_HTIF0_4);
        half_done_flag = 1;
    }
    HAL_DMA_IRQHandler(&hdma_adc1);
}

```

Функция усреднения и вывода:

```

void process_half_buffer(uint16_t* buffer, uint16_t start, uint16_t length)
{
    #define AVG_WINDOW 20
    uint32_t sum[3] = {0,0,0};
    uint16_t sample_count = length / 3;
    uint16_t avg_count = sample_count / AVG_WINDOW;

    for (uint16_t block = 0; block < avg_count; block++)
    {
        for (uint16_t i = 0; i < AVG_WINDOW; i++)
        {
            uint16_t idx = start + (block * AVG_WINDOW + i) * 3;
            sum[0] += buffer[idx];
            sum[1] += buffer[idx+1];
            sum[2] += buffer[idx+2];
        }
    }
}

```

```

    }
    uint16_t avg0 = sum[0] / AVG_WINDOW;
    uint16_t avg1 = sum[1] / AVG_WINDOW;
    uint16_t avg2 = sum[2] / AVG_WINDOW;
    char msg[64];
    sprintf(msg, "CH0=%u CH1=%u CH2=%u\r\n", avg0, avg1, avg2);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 100);
    sum[0]=0; sum[1]=0; sum[2]=0;
}
}

```

Основной цикл:

```

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_ADC1_Init();
    MX_USART2_UART_Init();

    HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, BUFFER_SIZE);

    while (1)
    {
        if (half_done_flag)
        {
            half_done_flag = 0;
            process_half_buffer(adc_buffer, 0, BUFFER_SIZE/2);
        }
        __WFI();
    }
}

```

В данном примере CPU не участвует в пересылке данных. Прерывание возникает каждые 120 отсчётов (половина буфера). Усреднение производится по 20 последовательным выборкам каждого канала, результат выводится в UART.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Использовать АЦП каналы 0,1,2. Размер буфера DMA = 300. Прерывание по полному заполнению буфера (complete transfer). Усреднение проводить по 25 отсчётам каждого канала. Выводить средние значения каждые 100 мс по UART.
2	Использовать АЦП каналы 1,2,3. Размер буфера DMA = 192. Прерывание по половинному заполнению. Усреднение проводить по 16 отсчётам каждого канала. Дополнительно выводить максимальное значение среди трёх каналов за интервал усреднения.
3	Использовать АЦП каналы 4,5,6. Размер буфера DMA = 480. Прерывание по полному заполнению. В обработчике прерывания не усреднять, а копировать вторую половину буфера в отдельный массив и устанавливать флаг. В основном цикле выполнять медианную фильтрацию (окно 5) для каждого канала, затем вывод через UART.
4	Использовать АЦП каналы 0,3,5. Размер буфера DMA = 256. Прерывание по половинному заполнению. Вместо простого усреднения применить экспоненциальное сглаживание (коэффициент 0.1) для каждого канала. Выводить отфильтрованные значения.
5	Использовать АЦП каналы 2,4,7. Размер буфера DMA = 360. Прерывание по полному заполнению. Усреднение проводить по 30 отсчётам. Выводить значения в формате CSV с меткой времени (счётчик прерываний).
6	Использовать АЦП каналы 0,1,2. Размер буфера DMA = 120.

	Прерывание по половинному заполнению. После усреднения (окно 10) передавать данные по UART только если разница между текущим и предыдущим средним превышает порог 20 LSB.
7	Использовать АЦП каналы 1,4,6. Размер буфера DMA = 600. Прерывание по полному заполнению. Усреднение проводить по 50 отсчётам. Выводить не только средние, но и среднеквадратическое отклонение для каждого канала за интервал.
8	Использовать АЦП каналы 0,2,5. Размер буфера DMA = 240. Прерывание по половинному заполнению. Выполнять калибровку: в первых 20 выборках после старта определить смещение (среднее) и вычитать его из всех последующих усреднённых значений.
9	Использовать АЦП каналы 3,4,5. Размер буфера DMA = 400. Прерывание по полному заполнению. Усреднение по 40 отсчётам. Параллельно подсчитывать частоту прерываний (скорость заполнения буфера) и выводить её раз в секунду.
10	Использовать АЦП каналы 0,6,7. Размер буфера DMA = 180. Прерывание по половинному заполнению. Применить линейную интерполяцию: если разность между двумя последовательными средними значениями какого-либо канала больше 100, вставить дополнительное значение (среднее арифметическое) в вывод.
11	Использовать АЦП каналы 1,2,7. Размер буфера DMA = 500. Прерывание по полному заполнению. Усреднение по 25 отсчётам. Дополнительно выводить признак переполнения буфера (если процессор не успевает обрабатывать данные).
12	Использовать АЦП каналы 0,4,8. Размер буфера DMA = 320.

	Прерывание по половинному заполнению. Реализовать простое дифференцирование: выводить не абсолютные значения, а их разность с предыдущим усреднённым значением.
13	Использовать АЦП каналы 2,3,6. Размер буфера DMA = 200. Прерывание по полному заполнению. Вместо усреднения применить децимацию: из каждых 10 последовательных отсчётов каждого канала сохранять только первый. Выводить децимированные отсчёты без дополнительной обработки.
14	Использовать АЦП каналы 1,5,8. Размер буфера DMA = 280. Прерывание по половинному заполнению. Усреднение по 14 отсчётам. Перед выводом масштабировать значения к диапазону 0-100% (напряжение опоры 3.3В, 12 бит).
15	Использовать АЦП каналы 0,3,7. Размер буфера DMA = 450. Прерывание по полному заполнению. Усреднение по 15 отсчётам. Реализовать передачу по UART в двоичном виде (три слова uint16_t) вместо текстового, добавив стартовый байт 0xAA и контрольную сумму XOR.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое DMA и зачем он используется для АЦП?
2. В чём разница между circular mode и normal mode DMA?
3. Что такое half-transfer interrupt и зачем он нужен?

4. Как вычислить размер DMA-буфера для 3 каналов АЦП?
5. Какова роль функции HAL_ADC_Start_DMA?
6. Как усредняются данные для уменьшения шума?
7. Как вывести результат по UART без блокировки?
8. Как убедиться, что DMA работает в режиме circular?

Лабораторная работа 2.

QUADRATURE ENCODER: ИЗМЕРЕНИЕ СКОРОСТИ И НАПРАВЛЕНИЯ ВРАЩЕНИЯ

Цель и содержание работы: Подключение инкрементального квадратурного энкодера к микроконтроллеру STM32 и разработка алгоритма измерения скорости и направления вращения вала с использованием аппаратного таймера в режиме Encoder Mode. Освоение методов точного подсчёта импульсов и пересчёта в обороты в секунду.

Теоретическое обоснование

Инкрементальный энкодер выдает два прямоугольных сигнала А и В, сдвинутых по фазе на 90 градусов. Направление вращения определяется по тому, какой сигнал опережает другой: если А опережает В – вращение вперёд, если В опережает А – вращение назад. Скорость пропорциональна частоте изменения состояния сигналов.

Таймеры STM32 поддерживают специальный режим Encoder Mode (TI1 и TI2 в режиме encoder). В этом режиме таймер автоматически инкрементирует или декрементирует свой счётный регистр (TIMx->CNT) при каждом изменении уровня на входах TI1 и TI2, учитывая направление вращения. Это позволяет получить точное приращение угла без нагрузки на CPU.

Для измерения скорости необходимо в течение фиксированного интервала времени (например, 100 мс) считывать значение счётчика, вычислять разность (количество импульсов) и делить на время. Чтобы получить обороты в секунду, нужно знать количество импульсов энкодера на один оборот (PPR – pulses per revolution). Полученная разность импульсов преобразуется в обороты: $\text{обороты_в_сек} = (\text{разность_импульсов} / \text{PPR}) / \text{интервал_в_сек}$.

Для повышения точности можно использовать два подхода: фиксированный интервал времени и измерение количества импульсов (более простой) или измерение периода между импульсами при низких скоростях. В работе применяется первый подход. Направление определяется знаком разности счётчика за интервал (положительная разность – одно направление, отрицательная – противоположное). Требуется учесть переполнение счётчика – для этого таймер настраивается на максимальное значение (например, 16-битный режим с автоперезагрузкой 0xFFFF). При вычислении разности используется арифметика без знака с приведением к знаковому типу.

Результаты скорости и направления выводятся по UART в виде текстовой строки.

Пример

Рассмотрим типовую настройку для микроконтроллера STM32F103C8T6 (Blue Pill). Энкодер с 400 импульсами на оборот (PPR=400). Сигналы А и В подключены к выводам PA0 (TIM2_CH1) и PA1 (TIM2_CH2). Таймер TIM2 настраивается в режим Encoder Mode TI1 и TI2. Измерение скорости выполняется каждые 100 мс с помощью системного таймера (SysTick). Вывод по UART1.

Код инициализации таймера в режиме энкодера:

```
void MX_TIM2_Encoder_Init(void)
{
    TIM_Encoder_InitTypeDef sEncoderConfig = {0};
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 0;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 0xFFFF;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    sEncoderConfig.EncoderMode = TIM_ENCODERMODE_TI1TI2;
    sEncoderConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
    sEncoderConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
    sEncoderConfig.IC1Prescaler = TIM_ICPSC_DIV1;
    sEncoderConfig.IC1Filter = 0;
    sEncoderConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
    sEncoderConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
    sEncoderConfig.IC2Prescaler = TIM_ICPSC_DIV1;
    sEncoderConfig.IC2Filter = 0;
    HAL_TIM_Encoder_Init(&htim2, &sEncoderConfig);
}
```

```

    HAL_TIM_Encoder_Start(&htim2, TIM_CHANNEL_ALL);
}

```

```

Глобальные переменные:
volatile uint32_t last_count = 0;
volatile uint32_t current_count = 0;
volatile int32_t delta = 0;
volatile uint8_t speed_ready = 0;
uint16_t ppr = 400;
float speed_rps = 0.0;
int8_t direction = 0; // 1 - вперёд, -1 - назад

```

Настройка прерывания по таймеру (например, TIM6 для периодического опроса) или использование SysTick. Допустим, SysTick настроен на 1 мс. В обработчике SysTick ведётся счётчик миллисекунд, и каждые 100 мс вызывается функция измерения.

```

void HAL_SYSTICK_Callback(void)
{
    static uint32_t ms_counter = 0;
    ms_counter++;
    if (ms_counter >= 100)
    {
        ms_counter = 0;
        current_count = __HAL_TIM_GET_COUNTER(&htim2);
        delta = (int32_t)(current_count - last_count);
        // Обработка переполнения счётчика (если delta больше половины периода -
        // знак меняется)
        if (delta > 32767) delta = delta - 65536;
        else if (delta < -32768) delta = delta + 65536;

        last_count = current_count;
        speed_ready = 1;
    }
}

```

В основном цикле:

```

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART1_UART_Init();
    MX_TIM2_Encoder_Init();

    // Запись начального значения счётчика
    last_count = __HAL_TIM_GET_COUNTER(&htim2);

    while (1)
    {
        if (speed_ready)
        {
            speed_ready = 0;
            // Скорость в оборотах в секунду: импульсы за 0.1с * 10 / PPR
            speed_rps = (float)delta * 10.0f / (float)ppr;
            if (delta > 0) direction = 1;
            else if (delta < 0) direction = -1;
            else direction = 0;
        }
    }
}

```

```

        char msg[64];
        sprintf(msg, "Speed: %.2f rps, Dir: %d, Delta: %ld\r\n", speed_rps,
direction, delta);
        HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 100);
    }
}

```

В этом примере таймер аппаратно считает импульсы энкодера. CPU только раз в 100 мс считывает разность и вычисляет скорость. Направление определяется знаком разности. Коррекция переполнения обеспечивает корректное значение даже при большом количестве импульсов за интервал.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Энкодер с PPR = 200. Измерительный интервал = 50 мс. Выводить скорость в оборотах в минуту (RPM). Направление отображать текстом "CW" или "CCW". Использовать таймер TIM3.
2	Энкодер с PPR = 360. Интервал = 200 мс. Дополнительно выводить абсолютный угол (количество импульсов с момента старта, пересчитанное в градусы). Для хранения угла использовать 32-битную переменную, учитывая переполнение

	счётчика TIM2.
3	Энкодер с PPR = 500. Интервал = 100 мс. Реализовать программный фильтр скорости: медианное значение из последних 5 измерений перед выводом. Использовать таймер TIM4.
4	Энкодер с PPR = 1000. Интервал = 10 мс. Скорость выводить только если она изменилась более чем на 0.5 об/сек по сравнению с предыдущим выводом. Таймер TIM5 (32-битный).
5	Энкодер с PPR = 250. Интервал = 500 мс. Добавить индикацию направления через светодиод: один светодиод для CW, другой для CCW. При остановке гасить оба. Таймер TIM1.
6	Энкодер с PPR = 400. Интервал = 150 мс. Выводить не только скорость, но и количество оборотов (накопленное целое число оборотов). Сброс оборотов по команде с UART (при получении символа 'R'). Таймер TIM2.
7	Энкодер с PPR = 2048. Интервал = 20 мс. Измерять также мгновенную частоту импульсов в герцах (импульсы в секунду) и выводить её вместе со скоростью. Таймер TIM3.
8	Энкодер с PPR = 600. Интервал = 40 мс. Реализовать два режима вывода: нормальный (каждые 40 мс) и усреднённый (среднее за 5 интервалов). Переключение режима по внешней кнопке. Таймер TIM4.
9	Энкодер с PPR = 1200. Интервал = 250 мс. Детектировать застой вала: если скорость меньше 0.2 об/сек в течение 2 секунд, отправить сообщение "STOP" один раз. Таймер TIM5.
10	Энкодер с PPR = 100. Интервал = 100 мс. Выводить скорость в двух форматах: в об/сек и в градусах в секунду. Использовать таймер TIM1. Направление кодировать знаком числа.

11	Энкодер с PPR = 800. Интервал = 30 мс. По нажатию кнопки сохранять текущее значение счётчика (угол) в EEPROM (эмулировать через внутренний Flash). При старте восстанавливать сохранённый угол. Таймер TIM2.
12	Энкодер с PPR = 3600. Интервал = 10 мс. Поскольку при высокой скорости возможны пропуски, использовать аппаратный входной фильтр таймера (значение фильтра = 2). Выводить предупреждение, если разность импульсов превышает половину периода счётчика. Таймер TIM3.
13	Энкодер с PPR = 512. Интервал = 60 мс. Выводить скорость и направление в виде строки для дисплея LCD (симуляция через UART). Дополнительно выводить максимальную достигнутую скорость за всё время работы. Таймер TIM4.
14	Энкодер с PPR = 2000. Интервал = 120 мс. Измерять ускорение: (текущая скорость - предыдущая скорость) / интервал. Выводить скорость, направление и ускорение (об/сек ²). Таймер TIM5.
15	Энкодер с PPR = 2500. Интервал = 80 мс. Реализовать протокол обмена: запрос скорости по команде 'S', запрос направления по команде 'D', запрос полной информации по команде 'A'. В остальное время данные не отправляются. Таймер TIM1.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое quadrature encoder и как он работает?
2. Какова роль сигналов А и В и что означают их фазы?
3. Как настроить таймер STM32 в режим Encoder Mode?
4. Как определить направление вращения из сигналов А и В?
5. Как вычислить скорость вращения из импульсов?
6. Что такое разрешение энкодера (CPR)?
7. Как обработать переполнение счётчика TIMx->CNT?
8. Как убедиться в корректности показаний энкодера?

Лабораторная работа 3.

ОДОМЕТРИЯ ДЛЯ ДИФФЕРЕНЦИАЛЬНОГО ПРИВОДА

Цель и содержание работы: Реализация алгоритма одометрии для колёсной платформы с дифференциальным приводом на основе данных с двух инкрементальных энкодеров. Расчёт текущих координат (x, y) и угла поворота (theta) робота относительно начальной позиции. Периодическая публикация оценённого положения по интерфейсу UART с частотой 20 Гц.

Теоретическое обоснование

Дифференциальный привод состоит из двух независимых колёс, каждое из которых оснащено энкодером. Перемещение робота за малый интервал времени dt определяется двумя величинами: пройденным линейным путём dS и изменением ориентации $d\theta$. Эти величины связаны с показаниями энкодеров.

Пусть левое колесо прошло расстояние dS_left , правое – dS_right . Для дифференциальной платформы справедливо:

$$dS = (dS_left + dS_right) / 2$$

$d\theta = (dS_right - dS_left) / B$, где B – ширина колеи (расстояние между колёсами).

Для перехода от импульсов энкодера к расстоянию необходимо знать цену импульса: $distance_per_pulse = (\pi * D) / (PPR * gear_ratio)$, где D – диаметр колеса, PPR – число импульсов энкодера на оборот, $gear_ratio$ – передаточное отношение редуктора (если энкодер стоит на валу двигателя). Упрощённо можно ввести масштабные коэффициенты: $linear_scale = (\pi * D) / (PPR * gear_ratio)$ – метры на импульс; $angle_scale = linear_scale / B$ – радианы на импульс разности.

Тогда за интервал между измерениями:

$$\Delta pulses_left = current_left - last_left$$

```
delta_pulses_right = current_right - last_right
dS = (delta_pulses_left + delta_pulses_right) * linear_scale / 2
dtheta = (delta_pulses_right - delta_pulses_left) * angle_scale
```

Накопление глобальных координат выполняется по формулам:

```
x += dS * cos(theta + dtheta/2)    (используется средний угол)
y += dS * sin(theta + dtheta/2)
theta += dtheta
```

Для устойчивости при малых поворотах можно использовать упрощённое обновление:

```
x += dS * cos(theta)
y += dS * sin(theta)
theta += dtheta
```

что допустимо при малых $d\theta$.

Измерения проводятся периодически с фиксированной частотой (например, 20 Гц), чтобы избежать накопления ошибки из-за нелинейности. Данные о положении выводятся по UART в виде строки "X:%.3f Y:%.3f TH:%.3f". При старте координаты обнуляются.

Пример

Рассмотрим типовую реализацию для платформы с двумя энкодерами, подключёнными к таймерам TIM2 (левый) и TIM3 (правый) в режиме Encoder Mode. Параметры: диаметр колеса $D = 0.065$ м (6.5 см), ширина колеи $B = 0.25$ м, $PPR = 400$, $gear_ratio = 1$ (энкодер на оси колеса). Частота обновления одометрии – 20 Гц (период 0.05 с). Используется SysTick или аппаратный таймер для формирования прерывания по времени.

Вычисление масштабных коэффициентов:

$linear_scale = (\pi * D) / PPR = (3.14159 * 0.065) / 400 \approx 0.0005105$
м/импульс

$angle_scale = linear_scale / B = 0.0005105 / 0.25 = 0.002042$ рад/импульс
разности (примерно 0.117 градуса на импульс разности)

Код инициализации энкодеров (аналогично ЛР2, для каждого таймера):

```

void MX_TIM2_Encoder_Init(void)
{
    TIM_Encoder_InitTypeDef sEncoderConfig = {0};
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 0;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 0xFFFF;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    sEncoderConfig.EncoderMode = TIM_ENCODERMODE_TI1TI2;
    sEncoderConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
    sEncoderConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
    HAL_TIM_Encoder_Init(&htim2, &sEncoderConfig);
    HAL_TIM_Encoder_Start(&htim2, TIM_CHANNEL_ALL);
}

```

Аналогично для TIM3.

Глобальные переменные:

```

volatile int16_t left_pulses, right_pulses;
volatile int16_t last_left, last_right;
float x = 0.0f, y = 0.0f, theta = 0.0f;
const float linear_scale = 0.0005105f;
const float angle_scale = 0.002042f;
volatile uint8_t odometry_ready = 0;

```

Таймер прерываний (например, TIM6 на 20 Гц):

```

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM6)
    {
        left_pulses = __HAL_TIM_GET_COUNTER(&htim2);
        right_pulses = __HAL_TIM_GET_COUNTER(&htim3);

        int16_t delta_left = left_pulses - last_left;
        int16_t delta_right = right_pulses - last_right;

        // Коррекция переполнения счётчика 16 бит (знаковые)
        if (delta_left > 32767) delta_left -= 65536;
        else if (delta_left < -32768) delta_left += 65536;
        if (delta_right > 32767) delta_right -= 65536;
        else if (delta_right < -32768) delta_right += 65536;

        last_left = left_pulses;
        last_right = right_pulses;

        float dS = (delta_left + delta_right) * linear_scale * 0.5f;
        float dtheta = (delta_right - delta_left) * angle_scale;

        // Обновление положения по средней ориентации
        float avg_theta = theta + dtheta * 0.5f;
        x += dS * cosf(avg_theta);
        y += dS * sinf(avg_theta);
        theta += dtheta;

        odometry_ready = 1;
    }
}

```

Основной цикл:

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART1_UART_Init();
    MX_TIM2_Encoder_Init();
    MX_TIM3_Encoder_Init();
    MX_TIM6_Init(); // прерывания с периодом 50 мс

    // Запись начальных значений счётчиков
    last_left = __HAL_TIM_GET_COUNTER(&htim2);
    last_right = __HAL_TIM_GET_COUNTER(&htim3);

    HAL_TIM_Base_Start_IT(&htim6);

    while (1)
    {
        if (odometry_ready)
        {
            odometry_ready = 0;
            char msg[80];
            sprintf(msg, "X:%.3f Y:%.3f TH:%.3f\r\n", x, y, theta);
            HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 100);
        }
    }
}
```

В этом примере каждые 50 мс вычисляются приращения по импульсам, обновляются координаты и угол. Оценка положения публикуется через UART. CPU не тратит время на опрос энкодеров, таймеры аппаратно считают импульсы.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№	Содержание индивидуального задания
---	------------------------------------

варианта	
2	D=0.065 м, B=0.28 м, PPR=400, gear_ratio=1. Частота 25 Гц. Выводить также линейную скорость и угловую скорость (dS/dt и $d\theta/dt$). Использовать плавающую точку с двумя знаками.
3	D=0.10 м, B=0.35 м, PPR=512, gear_ratio=1. Частота 20 Гц. Реализовать сброс положения ($x=0, y=0, \theta=0$) по внешней кнопке. При сбросе выводить сообщение "RESET".
4	D=0.06 м, B=0.22 м, PPR=1000, gear_ratio=1.5. Частота 50 Гц. Выводить данные в формате "X Y TH" с разделителем пробел, каждое значение с тремя десятичными знаками. Использовать фиксированный размер строки.
5	D=0.08 м, B=0.32 м, PPR=360, gear_ratio=1. Частота 15 Гц. Добавить простую модель дрейфа: если сумма абсолютных приращений импульсов за последнюю секунду меньше порога (50 импульсов), не обновлять угол ($d\theta=0$). Это помогает при остановке.
6	D=0.05 м, B=0.20 м, PPR=250, gear_ratio=2. Частота 30 Гц. Выводить не только положение, но и пройденное расстояние (total_distance) как сумму модулей dS . При достижении total_distance=10 метров отправлять сообщение "TARGET REACHED".
7	D=0.075 м, B=0.27 м, PPR=480, gear_ratio=1.2. Частота 20 Гц. Вычислять квадрат ошибки одометрии: сравнивать текущее расчётное положение с идеальной прямой (движение только вперёд по команде 'F'). Выводить отклонение от прямой (нормальное расстояние) через UART.
8	D=0.09 м, B=0.40 м, PPR=2048, gear_ratio=1. Частота 10 Гц. Использовать 32-битные счётчики таймеров (TIM5 и TIM6) для

	устранения переполнения. Выводить положение с двойной точностью (double).
9	D=0.055 м, B=0.24 м, PPR=400, gear_ratio=1. Частота 40 Гц. Реализовать буферизацию: накапливать до 5 измерений одометрии и отправлять одним пакетом через UART в бинарном виде (три числа float). Добавить стартовый байт 0xBB.
10	D=0.12 м, B=0.50 м, PPR=300, gear_ratio=1. Частота 20 Гц. Дополнительно к положению выводить ориентацию в кватернионной форме (q0,q1,q2,q3) для использования в фильтре. Кватернион вычислять из угла theta.
11	D=0.066 м, B=0.29 м, PPR=600, gear_ratio=1. Частота 25 Гц. При движении назад (dS отрицательное) инвертировать знак dtheta для коррекции? В задании требуется реализовать корректную одометрию при движении назад: знак dtheta должен оставаться в соответствии с разностью импульсов. Выполнить проверку и выводить дополнительный флаг направления движения.
12	D=0.085 м, B=0.33 м, PPR=800, gear_ratio=1.5. Частота 12 Гц. Из-за низкой частоты обновления возможны большие dtheta за один шаг. Использовать полную формулу обновления с cos(средний угол) и sin(средний угол). В отчёте показать разницу с упрощённой формулой.
13	D=0.07 м, B=0.26 м, PPR=1000, gear_ratio=1. Частота 20 Гц. Выводить значения x,y в дециметрах (умноженные на 10) целыми числами для экономии места. Угол theta в сотых долях градуса. Проверять переполнение целочисленного накопления.
14	D=0.095 м, B=0.38 м, PPR=256, gear_ratio=2. Частота 5 Гц. Низкая частота приводит к ступенчатому выводу. Реализовать линейную интерполяцию между измерениями: в промежутках

	отправлять предсказанные положения, используя последние вычисленные скорости. Выводить флаг реального или интерполированного значения.
15	D=0.06 м, B=0.21 м, PPR=400, gear_ratio=1. Частота 20 Гц. Добавить калибровку: проехать 2 метра по прямой и определить фактический linear_scale (если проехали 2.05 м, то скорректировать коэффициент). Выполнять калибровку по команде 'C' и сохранять новый коэффициент в энергонезависимую память.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое одометрия и для чего она используется?
2. Как описывается модель дифференциального привода?
3. Как накапливаются ошибки одометрии со временем?
4. Как вычислить (x, y, theta) из двух энкодеров?
5. Что такое разрешение (метров на импульс) для пересчёта?
6. Как задать частоту обновления одометрии (20 Гц)?
7. Как публиковать одометрию по UART в структурированном виде?
8. Какова роль параметра расстояния между колёсами (wheelbase)?

Лабораторная работа 4.

КОМПЛЕМЕНТАРНЫЙ ФИЛЬТР ДЛЯ СЛИЯНИЯ АКСЕЛЕРОМЕТРА И ГИРОСКОПА

Цель и содержание работы: Реализация комплементарного фильтра на микроконтроллере STM32 для слияния данных акселерометра и гироскопа с датчика MPU6050. Получение стабильной и точной оценки угла наклона (крена и тангажа) с устранением высокочастотного шума акселерометра и низкочастотного дрейфа гироскопа. Вывод отфильтрованного угла по интерфейсу UART.

Теоретическое обоснование

Акселерометр измеряет проекции ускорения, включая гравитационную составляющую. При неподвижном объекте вектор ускорения свободного падения ($1g$) указывает направление «вниз». По соотношению проекций A_x , A_y , A_z можно вычислить углы наклона относительно горизонта. Для угла крена (roll) относительно оси X используется формула $\text{roll_acc} = \text{atan2}(A_y, A_z)$. Для угла тангажа (pitch) – $\text{pitch_acc} = \text{atan2}(-A_x, \sqrt{A_y^2 + A_z^2})$. Однако акселерометр чувствителен к линейным ускорениям и шумам, что даёт зашумлённый сигнал при движении.

Гироскоп измеряет угловую скорость (град/с или рад/с). Интегрирование угловой скорости даёт угол: $\text{angle_gyro} += \text{gyro_rate} * dt$. Гироскоп имеет высокую точность на коротких интервалах, но его интеграл накапливает ошибку смещения (дрейф), что приводит к неограниченному росту ошибки со временем.

Комплементарный фильтр объединяет преимущества обоих датчиков:

$$\text{angle} = \alpha * (\text{angle_prev} + \text{gyro_rate} * dt) + (1 - \alpha) * \text{accel_angle}$$

где α – коэффициент доверия к гироскопу (обычно $0.95 \dots 0.99$).

Фильтр пропускает низкие частоты от акселерометра (корректируя дрейф) и

высокие частоты от гироскопа (сглаживая шум). Сумма коэффициентов равна 1. Постоянная времени фильтра $\tau = (\alpha * dt) / (1 - \alpha)$. Рекомендуемое $\tau = 0.5 \dots 2$ секунды.

Для корректной работы необходимо:

- Настроить I2C для связи с MPU6050 по адресу 0x68.
- Сконфигурировать гироскоп (например, диапазон $\pm 250^\circ/\text{с}$) и акселерометр ($\pm 2g$).
- Считывать сырые данные и преобразовывать их в физические величины с учётом чувствительности.
- Измерять dt между выборками (используя аппаратный таймер или SysTick).
- Применять фильтр отдельно для угла крена и тангажа.

Результат выводится по UART с частотой 50-100 Гц.

Пример

Рассмотрим реализацию для STM32F103 и датчика MPU6050. I2C1 (PB6 – SCL, PB7 – SDA). Частота опроса 100 Гц ($dt = 0.01$ с). Акселерометр: диапазон $\pm 2g$, чувствительность 16384 LSB/g. Гироскоп: диапазон $\pm 250^\circ/\text{с}$, чувствительность 131 LSB/ $(^\circ/\text{с})$. Коэффициент $\alpha = 0.98$. Вычисляется угол тангажа (pitch) и крена (roll). Вывод по UART1.

Инициализация I2C и MPU6050:

```
void MX_I2C1_Init(void)
{
    hi2c1.Instance = I2C1;
    hi2c1.Init.ClockSpeed = 400000;
    hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
    hi2c1.Init.OwnAddress1 = 0;
    hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
    hi2c1.Init.OwnAddress2 = 0;
    hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
    hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
    HAL_I2C_Init(&hi2c1);
}
```

```
uint8_t MPU6050_Init(void)
{
    uint8_t check;
    HAL_I2C_Mem_Read(&hi2c1, MPU6050_ADDR, 0x75, 1, &check, 1, 100);
    if (check != 0x68) return 0;
    uint8_t data = 0;
    data = 0x00; // выключение сна
    HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, 0x6B, 1, &data, 1, 100);
    data = 0x00; // гироскоп ±250°/с
    HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, 0x1B, 1, &data, 1, 100);
    data = 0x00; // акселерометр ±2g
    HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, 0x1C, 1, &data, 1, 100);
    return 1;
}
```

Чтение сырых данных:

```
void MPU6050_Read_Raw(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx,
int16_t* gy, int16_t* gz)
{
    uint8_t buf[14];
    HAL_I2C_Mem_Read(&hi2c1, MPU6050_ADDR, 0x3B, 1, buf, 14, 100);
    *ax = (buf[0] << 8) | buf[1];
    *ay = (buf[2] << 8) | buf[3];
    *az = (buf[4] << 8) | buf[5];
    *gx = (buf[8] << 8) | buf[9];
    *gy = (buf[10] << 8) | buf[11];
    *gz = (buf[12] << 8) | buf[13];
}
```

Глобальные переменные и фильтр:

```
volatile float pitch = 0.0f, roll = 0.0f;
volatile uint8_t data_ready = 0;
uint32_t last_time = 0;
float dt = 0.01f; // фиксированное значение, но можно измерять
const float alpha = 0.98f;
const float gyro_scale = 1.0f / 131.0f; // преобразование в °/с
const float accel_scale = 1.0f / 16384.0f; // преобразование в g
```

Функция обработки (вызывается по таймеру 100 Гц или в основном цикле с измерением dt):

```
void Complementary_Filter_Update(void)
{
    int16_t ax_raw, ay_raw, az_raw, gx_raw, gy_raw, gz_raw;
    MPU6050_Read_Raw(&ax_raw, &ay_raw, &az_raw, &gx_raw, &gy_raw, &gz_raw);

    float ax = ax_raw * accel_scale;
    float ay = ay_raw * accel_scale;
    float az = az_raw * accel_scale;
    float gyro_x = gx_raw * gyro_scale; // °/с
    float gyro_y = gy_raw * gyro_scale;
    float gyro_z = gz_raw * gyro_scale;

    // Углы из акселерометра (тангаж и крен)
    float pitch_acc = atan2f(-ax, sqrtf(ay*ay + az*az)) * 180.0f / 3.14159f;
    float roll_acc = atan2f(ay, az) * 180.0f / 3.14159f;
```

```
// Интегрирование гироскопа (простое)
pitch = alpha * (pitch + gyro_y * dt) + (1.0f - alpha) * pitch_acc;
roll = alpha * (roll + gyro_x * dt) + (1.0f - alpha) * roll_acc;

data_ready = 1;
}
```

Основной цикл с точным dt:

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_I2C1_Init();
    MX_USART1_UART_Init();
    if (!MPU6050_Init()) while(1);

    last_time = HAL_GetTick();

    while (1)
    {
        uint32_t now = HAL_GetTick();
        dt = (now - last_time) / 1000.0f;
        if (dt >= 0.005f) // ограничение для стабильности
        {
            last_time = now;
            Complementary_Filter_Update();
        }
        if (data_ready)
        {
            data_ready = 0;
            char msg[64];
            sprintf(msg, "Pitch:%.1f Roll:%.1f\r\n", pitch, roll);
            HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 100);
        }
    }
}
```

В этом примере акселерометр корректен в статике, гироскоп обеспечивает динамическую точность. Коэффициент 0.98 даёт хорошее подавление дрейфа.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;

5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Использовать только один угол – тангаж. $\alpha = 0.95$, $dt = 20$ мс (50 Гц). Выводить угол с точностью до 0.1 градуса. Дополнительно выводить сырой угол гироскопа для сравнения.
2	Использовать только крен. $\alpha = 0.99$, $dt = 5$ мс (200 Гц). Реализовать предварительную фильтрацию акселерометра: скользящее среднее по 4 выборкам перед вычислением угла.
3	Для обоих углов (pitch и roll). $\alpha = 0.97$, $dt = 25$ мс (40 Гц). Выводить также интегрированный угол гироскопа (без комплементарной коррекции) и угол акселерометра отдельно. Формат вывода: три значения через пробел.
4	$\alpha = 0.92$, $dt = 10$ мс. Изменять коэффициент комплементарного фильтра в реальном времени по команде с UART: символ 'A' увеличивает α на 0.01 (до 0.99), символ 'B' уменьшает до 0.90. Выводить текущий α .
5	Использовать вычисление угла по акселерометру без $\arctan 2$, а через линеаризацию (малые углы: $\text{pitch_acc} \approx -ax$, при условии, что ось Z вертикальна). Сравнить с полной формулой и вывести разницу. $DT = 15$ мс.
6	Реализовать комплементарный фильтр не для эйлеровых углов, а для кватернионов (более сложная задача). Альтернативно: добавить датчик температуры MPU6050 и выводить её вместе с углами. $\alpha = 0.98$, $dt = 10$ мс.
7	Ввести автоматическую коррекцию смещения гироскопа: в

	первые 2 секунды после старта считать среднее значение гироскопа при неподвижном датчике и вычитать это смещение из всех последующих измерений. $\text{Alpha} = 0.98$, $\text{dt} = 20$ мс.
8	$\text{Alpha} = 0.96$, $\text{dt} = 30$ мс. Выводить углы в градусах, но также пересчитывать их в радианы и передавать как два числа типа float в бинарном виде (8 байт). Добавить стартовый байт 0x55.
9	Использовать только один угол (тангаж), но добавить наложение низкочастотного фильтра на выходной сигнал (простой RC-фильтр) для дополнительного сглаживания. Выводить исходный и сглаженный углы. $\text{Alpha} = 0.98$, $\text{dt} = 10$ мс.
10	$\text{Alpha} = 0.995$, $\text{dt} = 2$ мс (500 Гц). Высокая частота обновления. Оценить дрейф гироскопа за 1 минуту без акселерометра (временно выставить $\text{alpha}=1$) и записать максимальное отклонение. Выводить результат.
11	Для платформы, где ось Y направлена вперёд, а X вбок. Переопределить формулы расчёта углов акселерометра и учесть это. $\text{Alpha} = 0.97$, $\text{dt} = 20$ мс. Выводить крен и тангаж с пояснениями.
12	Использовать внешний прерывание по готовности данных MPU6050 (вывод INT). По прерыванию читать данные и выполнять фильтр. dt вычислять как разницу времен между прерываниями. $\text{Alpha} = 0.98$. Выводить частоту прерываний (скорость сбора).
13	Реализовать комплементарный фильтр для угла рыскания (yaw) с использованием магнитометра MPU6050 (на самом деле MPU6050 не имеет магнитометра, это MPU9250 – но в задании разрешено эмулировать или использовать компас). Для простоты: добавить третий угол – yaw, вычисляемый из

	гироскопа, без акселерометра. $\alpha = 0.98$, $dt = 20$ мс, выводить φ_{aw} отдельно.
14	$\alpha = 0.9$, $dt = 50$ мс (низкая частота). Показать, как увеличивается влияние акселерометра. Выводить углы и дополнительно абсолютную разницу между фильтрованным углом и углом акселерометра в процентах. Сделать вывод о влиянии α .
15	Использовать отдельные коэффициенты α для крена и тангажа: $\alpha_{roll} = 0.97$, $\alpha_{pitch} = 0.99$. $dt = 10$ мс. Объяснить, почему они могут различаться. Выводить оба угла с пометками R и P.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. В чём проблема использования только акселерометра для угла наклона?
2. В чём проблема использования только гироскопа?
3. Как работает комплементарный фильтр?
4. Что такое коэффициент α (0.98) и как он выбирается?
5. Что такое MPU6050 и как подключить его по I2C?
6. Как вычислить угол из данных акселерометра ($\arctan 2$)?
7. Как интегрировать гироскоп для получения угла?
8. Как настроить dt (период дискретизации)?

Лабораторная работа 5.

РАСШИРЕННЫЙ ФИЛЬТР КАЛМАНА (ЕКФ) ДЛЯ ОРИЕНТАЦИИ (IMU + ЭНКОДЕРЫ)

Цель и содержание работы: Реализация расширенного фильтра Калмана (Extended Kalman Filter, EKF) на микроконтроллере STM32 для слияния данных гироскопа (IMU) и одометрии колёсных энкодеров. Получение оптимальной оценки угла поворота (θ) робота с учётом шумов различных датчиков. Сравнение эффективности ЕКФ с классическим комплементарным фильтром (Лабораторная работа 4). Интеграция легковесной библиотеки TinyEKF.

Теоретическое обоснование

Классический комплементарный фильтр (ЛР4) сочетает долгосрочную точность акселерометра с краткосрочной стабильностью гироскопа с помощью фиксированных коэффициентов. Расширенный фильтр Калмана предлагает более общий и теоретически обоснованный подход к слиянию данных. Это рекурсивный алгоритм, который оценивает состояние системы, минимизируя среднеквадратичную ошибку. В отличие от линейного фильтра Калмана, ЕКФ применяется к нелинейным системам через линеаризацию с помощью матриц Якоби.

В данной работе вектор состояния выбран двумерным:

$x = [\theta, \omega_{\text{bias}}]^T$, где θ — угол поворота робота (рад), ω_{bias} — смещение нуля (дрейф) гироскопа (рад/с), которое также оценивается для компенсации ошибок интегрирования.

Модель процесса (предсказание). На каждом шаге dt состояние предсказывается по данным гироскопа с учётом смещения:

$$\theta_{\text{pred}} = \theta_{\text{prev}} + (\text{gyro_rate} - \omega_{\text{bias_prev}}) * dt$$

$\text{omega_bias_}\{\text{pred}\} = \text{omega_bias_prev}$ (модель блуждания со случайной добавкой)

Предсказание пересчитывает ковариационную матрицу ошибок P . В библиотеке TinyEKF модель задаётся через функцию $f(x, dt)$, возвращающую новое состояние, и её Якобиан F . Шум процесса представлен матрицей Q , характерной для дрейфа гироскопа и ускорений (выбирается экспериментально).

Модель измерения. EKF корректирует предсказание, используя измерения:

— theta_odom : угол, вычисленный по разности импульсов левого и правого энкодера (одометрия, ЛРЗ).

— theta_accel : угол, полученный из акселерометра (ЛР4).

Измерения подключаются с помощью функции $h(x)$, которая в простейшем случае является прямой проекцией: $h(x) = \text{theta}$ (угол из состояния). Матрица Якоби H в этой модели равна $[1, 0]$. Шум измерений представлен матрицей R , значения которой отражают доверие к одометрии и акселерометру.

Цикл работы. EKF работает в два этапа:

1. Прогноз (шаг по времени): новая оценка угла и смещения на основе показаний гироскопа.

2. Коррекция (шаг измерения): поступление угла из одометрии или акселерометра.

Частота прогноза может быть выше, чем коррекции, если датчики имеют разную частоту обновления.

Библиотека TinyEKF. Для упрощения реализации используется TinyEKF — легковесная, заголовочная (header-only) C/C++ библиотека с поддержкой чисел с плавающей точкой одинарной или двойной точности. Она предназначена для работы на микроконтроллерах Arduino, STM32 и других, используя статическое выделение памяти. Для работы достаточно

определить одну модельную функцию (model), в которой заполняются значения функции перехода состояния $f(x)$, её матрица Якоби F , функция измерения $h(x)$ и её Якобиан H . Прогноз и коррекция автоматически выполняются вызовом метода `step` с передачей вектора наблюдения z .

Пример

Рассмотрим реализацию EKF на STM32F103 для робота с двумя энкодерами и IMU MPU6050. Гироскоп читается по I2C с частотой 100 Гц, одометрия обновляется с частотой 20 Гц. $dt = 0.01$ с для прогноза. Вектор состояния: [угол (рад), смещение_гироскопа (рад/с)]. Шумы: $Q = [0.001, 0; 0, 0.0001]$, $R_{odom} = 0.01$ (рад²), $R_{accel} = 0.1$.

Подключение TinyEKF. Заголовочный файл библиотеки (например, `tinyekf.h`) помещается в проект. Определяется модель:

```
#include "tinyekf.h"
#include <math.h>

#define STATE_DIM 2
#define MEASURE_DIM 1

float gyro_raw = 0.0; // из MPU6050, преобр. в рад/с
float angle_odom = 0.0; // из энкодеров (рад)
float angle_accel = 0.0; // из акселерометра (рад)

// Матрицы шумов
static float Q[STATE_DIM*STATE_DIM] = {0.001f, 0.0f, 0.0f, 0.0001f};
static float R[MEASURE_DIM*MEASURE_DIM] = {0.01f}; // для одометрии

// Начальная оценка: угол 0, смещение 0
static float x[STATE_DIM] = {0.0f, 0.0f};
// Начальная ковариация
static float P[STATE_DIM*STATE_DIM] = {1.0f, 0.0f, 0.0f, 1.0f};

// Модельная функция, вызываемая TinyEKF
int model(float *x, float *F, float *h, float *H, int n, int m, float dt)
{
    // f(x): предсказание состояния по гироскопу
    float theta = x[0];
    float bias = x[1];
    float gyro_corrected = gyro_raw - bias;
    float theta_pred = theta + gyro_corrected * dt;
    float bias_pred = bias; // блуждание учитывается Q
    x[0] = theta_pred;
    x[1] = bias_pred;

    // Матрица Якоби F (линеаризация f)
    if (F != NULL)
```

```

    {
        F[0] = 1.0f;           // dtheta/dtheta
        F[1] = -dt;           // dtheta/dbias
        F[2] = 0.0f;           // dbias/dtheta
        F[3] = 1.0f;           // dbias/dbias
    }

    // h(x): измеряем угол
    if (h != NULL) h[0] = theta_pred;

    // Матрица Якоби H (проекция)
    if (H != NULL)
    {
        H[0] = 1.0f; // dh/dtheta
        H[1] = 0.0f; // dh/dbias
    }

    return 0;
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_I2C1_Init();
    MX_USART1_UART_Init();
    MX_TIM2_Encoder_Init(); // левый энкодер
    MX_TIM3_Encoder_Init(); // правый энкодер
    MPU6050_Init();

    // Инициализация EKF
    EKF ekf = ekf_init(STATE_DIM, MEASURE_DIM, 0, model, x, P, Q, R);
    float z[1] = {0.0f};
    uint32_t last_time = HAL_GetTick();
    float dt = 0.0f;

    while (1)
    {
        uint32_t now = HAL_GetTick();
        dt = (now - last_time) / 1000.0f;
        if (dt >= 0.01f) // 100 Гц прогноз
        {
            last_time = now;

            // Чтение гироскопа (рад/с)
            int16_t gz_raw;
            MPU6050_Read_Gyro(&gz_raw);
            gyro_raw = gz_raw / 131.0f * 3.14159f / 180.0f; // в рад/с

            // Выполнение шага прогноза
            // В TinyEKF шаг step автоматически делает прогноз и коррекцию,
            // но передача z = NULL выполняет только прогноз
            // Однако для раздельного управления проще пересобрать логику
            // Упрощённо: один вызов step с измерением, когда оно доступно

            // В данном примере: каждые 50 мс (20 Гц) коррекция по одометрии
            if ( (now % 50) < dt*1000 ) // грубо, лучше по флагу
            {
                // Вычисление угла одометрии за интервал (ЛРЗ)
                angle_odom = compute_odom_angle();
                z[0] = angle_odom;
                ekf.step(&ekf, z);
                // Отправка результата
                char msg[64];
                sprintf(msg, "EKF Angle: %.3f rad, Bias: %.3f rad/s\r\n",
ekf.x[0], ekf.x[1]);
            }
        }
    }
}

```

```

        HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 100);
    }
    else
    {
        // Шаг только прогноза (передаётся z = NULL)
        ekf.step(&ekf, NULL);
    }
}
}
}

```

В этом примере EKF прогнозирует угол на основе гироскопа (с учётом оцениваемого смещения) и корректируется по углу из одометрии. Аналогично можно добавить коррекцию по акселерометру, переключая матрицу шумов R или используя последовательные вызовы step с разными измерениями.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Вектор состояния: [угол, угловая_скорость]. Прогноз: угол += скорость * dt, скорость += шум процесса. Измерения: угол из одометрии. Частота прогноза 100 Гц, коррекции 10 Гц. Вывести оценённую угловую скорость.
2	Вектор состояния: [угол, смещение_гироскопа]. Измерения: угол из акселерометра и угол из одометрии. Использовать два последовательных шага коррекции с разными R. Сравнить

	результат с комплементарным фильтром (альфа=0.98).
3	Вектор состояния: [угол, смещение_гироскопа, смещение_одометрии]. Добавить оценку систематической ошибки одометрии (например, из-за разного диаметра колёс). Измерение: только угол из акселерометра. Оценить сходимость.
4	Вектор состояния: [угол, угловая_скорость, смещение_гироскопа] (3 состояния). Прогноз: $\text{угол} += (\text{скорость} - \text{смещение}) * dt$, $\text{скорость} += \text{шум}$, смещение постоянно. Измерения: угол из одометрии и акселерометра. Реализовать выбор источника измерения по команде UART.
5	Использовать кватернионную постановку EKF (4 состояния: q_0, q_1, q_2, q_3). Прогноз по гироскопу (кинематическое уравнение). Измерение: ориентация из акселерометра (крен и тангаж). Пересчёт кватерниона в углы Эйлера для вывода.
6	Вектор состояния: [угол, смещение_гироскопа]. Частота прогноза 200 Гц. Измерения: только от одометрии с частотой 20 Гц, но с адаптивным R: $R = 0.01 + 0.1 * \omega $, т.е. доверие падает при быстром вращении.
7	Вектор состояния: [угол, смещение_гироскопа, радиус_колеса] для компенсации масштабной ошибки одометрии. Прогноз угла через энкодеры (с использованием радиуса). Измерение: угол из акселерометра. Вывести оценённый радиус.
8	Использовать EKF не для угла, а для декартовых координат (x, y, theta, v) на основе данных энкодеров (ЛРЗ) и гироскопа. Модель процесса: движение с постоянной скоростью, поворот. Измерение: приращения энкодеров. Оценить траекторию.
9	Вектор состояния: [угол, смещение_гироскопа]. Шумы процесса Q и измерения R настраиваются интерактивно через UART.

	Выводить для каждого шага новшество (инновацию) — разность между измерением и предсказанием. Оптимизировать Q и R.
10	Использовать Error-State EKF (ES-EKF): номинальное состояние интегрируется гироскопом, EKF оценивает ошибку. Состояние ошибки: [delta_theta, смещение_гироскопа]. Измерение: угол одометрии. Сравнить с прямым EKF.
11	Вектор состояния: [угол, смещение_гироскопа]. Вместо библиотеки TinyEKF реализовать EKF вручную, используя матричные операции 2x2. Запрограммировать прогноз и коррекцию. Проверить идентичность результата.
12	Вектор состояния: [угол, смещение_гироскопа]. Сделать два EKF параллельно: один работает с частотой прогноза 50 Гц, другой 200 Гц. Сравнить точность оценки при резких движениях (рывках). Результат выводить через UART.
13	Вектор состояния: [угол, угловая_скорость, смещение_гироскопа, ускорение] (4 состояния). Прогноз: угол += скорость*dt, скорость += ускорение*dt, ускорение += шум. Измерения: угол из одометрии и акселерометра. Оценить, улучшает ли это предсказание.
14	Интегрировать EKF с магнитометром (HMC5883L). Вектор состояния: [угол_рыскания, смещение_гироскопа_z]. Измерения: угол из одометрии (при движении) и из магнитометра (при малой скорости). Выводить оценённый курс.
15	Вектор состояния: [угол, смещение_гироскопа]. Реализовать автоматическую инициализацию: первые 100 шагов усреднять гироскоп для начальной оценки смещения. Учитывать это в начальном x и P. После инициализации запустить EKF.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое расширенный фильтр Калмана (EKF)?
2. Из каких шагов состоит алгоритм EKF?
3. Что такое матрица ковариации шума процесса Q ?
4. Что такое матрица ковариации шума измерений R ?
5. Как выбирается состояние системы для оценки ориентации?
6. Что такое библиотека TinyEKF и как её интегрировать?
7. Как сравнить EKF с комплементарным фильтром?
8. В каких ситуациях EKF даёт лучший результат?

Лабораторная работа 6.

ПЕРЕДАЧА ДАННЫХ С STM32 НА RASPBERRY PI ЧЕРЕЗ UART

Цель и содержание работы: Настроить надёжный канал передачи данных между микроконтроллером STM32 и одноплатным компьютером Raspberry Pi по интерфейсу UART, упаковывая данные одометрии и углов в бинарную структуру с заголовком и CRC.

Теоретическое обоснование

UART — последовательный интерфейс передачи данных, широко применяемый для связи между микроконтроллером (STM32) и одноплатным компьютером (Raspberry Pi). Для надёжной передачи используются бинарные фреймы с заголовком, полями данных и контрольной суммой (CRC8/CRC16).

Структура фрейма: стартовые байты (0xAA 0x55), длина, тип пакета, данные (float32-поля: x, y, theta, speed_left, speed_right), CRC16. На стороне STM32 применяется HAL_UART_Transmit_DMA для передачи без блокировки основного цикла (частота 20–50 Гц).

На Raspberry Pi Python-скрипт с библиотекой pySerial читает байты из /dev/ttyAMA0, ищет заголовок, парсит поля с помощью struct.unpack и проверяет CRC.

Пример

Пример структуры фрейма на C (STM32):

```
typedef struct { uint16_t header; float x; float y; float theta; uint16_t crc; }  
OdomFrame_t;
```

Отправка (в основном цикле, 20 Гц):

```

frame.header = 0xAA55; frame.x = odom_x; frame.y = odom_y; frame.theta
= odom_theta;

frame.crc = crc16((uint8_t*)&frame, sizeof(frame)-2);
HAL_UART_Transmit_DMA(&huart2, (uint8_t*)&frame, sizeof(frame));

```

Python-парсер на Raspberry Pi:

```
import serial, struct
```

```
ser = serial.Serial("/dev/ttyAMA0", 115200)
```

```
while True:
```

```

    if ser.read(2) == b"\xaa\x55":
        data = ser.read(14)
        x, y, theta, crc = struct.unpack("<fffH", data)
        print(f"x={x:.3f} y={y:.3f} theta={theta:.3f}")

```

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Добавить в фрейм поле speed (float) скорости вращения мотора 2. Частота передачи: 12 Гц. Вывести данные в консоль.
2	Добавить в фрейм поле speed (float) скорости вращения мотора 1. Частота передачи: 14 Гц. Вывести данные в консоль.
3	Добавить в фрейм поле speed (float) скорости вращения мотора 2. Частота передачи: 16 Гц. Вывести данные в консоль.

4	Добавить в фрейм поле speed (float) скорости вращения мотора 1. Частота передачи: 18 Гц. Вывести данные в консоль.
5	Добавить в фрейм поле speed (float) скорости вращения мотора 2. Частота передачи: 20 Гц. Вывести данные в консоль.
6	Добавить в фрейм поле speed (float) скорости вращения мотора 1. Частота передачи: 22 Гц. Вывести данные в консоль.
7	Добавить в фрейм поле speed (float) скорости вращения мотора 2. Частота передачи: 24 Гц. Вывести данные в консоль.
8	Добавить в фрейм поле speed (float) скорости вращения мотора 1. Частота передачи: 26 Гц. Вывести данные в консоль.
9	Добавить в фрейм поле speed (float) скорости вращения мотора 2. Частота передачи: 28 Гц. Вывести данные в консоль.
10	Добавить в фрейм поле speed (float) скорости вращения мотора 1. Частота передачи: 30 Гц. Вывести данные в консоль.
11	Добавить в фрейм поле speed (float) скорости вращения мотора 2. Частота передачи: 32 Гц. Вывести данные в консоль.
12	Добавить в фрейм поле speed (float) скорости вращения мотора 1. Частота передачи: 34 Гц. Вывести данные в консоль.
13	Добавить в фрейм поле speed (float) скорости вращения мотора 2. Частота передачи: 36 Гц. Вывести данные в консоль.
14	Добавить в фрейм поле speed (float) скорости вращения мотора 1. Частота передачи: 38 Гц. Вывести данные в консоль.
15	Добавить в фрейм поле speed (float) скорости вращения мотора 2. Частота передачи: 40 Гц. Вывести данные в консоль.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики,

ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое бинарный фрейм и зачем он нужен?
2. Как вычисляется CRC16 и зачем нужна контрольная сумма?
3. Как настроить UART с DMA на STM32?
4. Как использовать struct.unpack в Python?
5. Как избежать потери данных при высокой частоте передачи?
6. Что такое заголовок фрейма и как его задать?
7. Как обработать неполный фрейм на приёмной стороне?
8. Как измерить задержку передачи данных (latency)?

Лабораторная работа 7.

ПРИЁМ КОМАНД УПРАВЛЕНИЯ С RASPBERRY PI НА STM32 (CMD_VEL)

Цель и содержание работы: Реализовать приём команд линейной (v) и угловой (ω) скорости, передаваемых с Raspberry Pi по UART, и преобразование их в ШИМ-сигналы для управления моторами через драйвер L298N с контролем таймаута.

Теоретическое обоснование

Команда управления (`cmd_vel`) содержит линейную скорость v (м/с) и угловую скорость ω (рад/с). Для дифференциального привода скорости колёс вычисляются как: $v_{\text{left}} = (v - \omega * L/2)$, $v_{\text{right}} = (v + \omega * L/2)$, где L — база шасси. Полученные значения масштабируются в ШИМ (0–100%).

Приём по UART организуется через прерывания или DMA с кольцевым буфером. Таймаут безопасности: если команда не обновляется более 500 мс, моторы останавливаются.

Драйвер L298N управляется двумя сигналами на канал: направление (GPIO) и скорость (TIM PWM). ШИМ-частота 20 кГц обеспечивает плавное вращение.

Пример

Формат бинарной команды (8 байт): 0xCC 0xDD + float v + float ω

Приём на STM32 (прерывание UART):

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *h) {
```

```
    if (rx_buf[0]==0xCC && rx_buf[1]==0xDD) {  
        memcpy(&cmd_v, &rx_buf[2], 4);  
        memcpy(&cmd_omega, &rx_buf[6], 4);  
    }
```

```

        last_cmd_tick = HAL_GetTick();
    }
    HAL_UART_Receive_IT(h, rx_buf, 8);
}

```

В основном цикле (1 мс тик таймера):

```

if (HAL_GetTick() - last_cmd_tick > 500) { cmd_v = 0; cmd_omega = 0; }
float vl = (cmd_v - cmd_omega*WHEEL_BASE/2.0f) / WHEEL_RADIUS;
float   vr   =   (cmd_v   +   cmd_omega*WHEEL_BASE/2.0f)   /
WHEEL_RADIUS;
set_motor_pwm(LEFT, vl); set_motor_pwm(RIGHT, vr);

```

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Реализовать таймаут безопасности 320 мс. База шасси L=0.16 м. Радиус колеса R=0.035 м. Вывести v_left и v_right по UART с частотой 21 Гц.
2	Реализовать таймаут безопасности 340 мс. База шасси L=0.17 м. Радиус колеса R=0.040 м. Вывести v_left и v_right по UART с частотой 22 Гц.
3	Реализовать таймаут безопасности 360 мс. База шасси L=0.18 м. Радиус колеса R=0.045 м. Вывести v_left и v_right по UART с

	частотой 23 Гц.
4	Реализовать таймаут безопасности 380 мс. База шасси $L=0.19$ м. Радиус колеса $R=0.050$ м. Вывести v_left и v_right по UART с частотой 24 Гц.
5	Реализовать таймаут безопасности 400 мс. База шасси $L=0.20$ м. Радиус колеса $R=0.030$ м. Вывести v_left и v_right по UART с частотой 25 Гц.
6	Реализовать таймаут безопасности 420 мс. База шасси $L=0.21$ м. Радиус колеса $R=0.035$ м. Вывести v_left и v_right по UART с частотой 26 Гц.
7	Реализовать таймаут безопасности 440 мс. База шасси $L=0.22$ м. Радиус колеса $R=0.040$ м. Вывести v_left и v_right по UART с частотой 27 Гц.
8	Реализовать таймаут безопасности 460 мс. База шасси $L=0.23$ м. Радиус колеса $R=0.045$ м. Вывести v_left и v_right по UART с частотой 28 Гц.
9	Реализовать таймаут безопасности 480 мс. База шасси $L=0.24$ м. Радиус колеса $R=0.050$ м. Вывести v_left и v_right по UART с частотой 29 Гц.
10	Реализовать таймаут безопасности 500 мс. База шасси $L=0.25$ м. Радиус колеса $R=0.030$ м. Вывести v_left и v_right по UART с частотой 30 Гц.
11	Реализовать таймаут безопасности 520 мс. База шасси $L=0.26$ м. Радиус колеса $R=0.035$ м. Вывести v_left и v_right по UART с частотой 31 Гц.
12	Реализовать таймаут безопасности 540 мс. База шасси $L=0.27$ м. Радиус колеса $R=0.040$ м. Вывести v_left и v_right по UART с частотой 32 Гц.

13	Реализовать таймаут безопасности 560 мс. База шасси $L=0.28$ м. Радиус колеса $R=0.045$ м. Вывести v_left и v_right по UART с частотой 33 Гц.
14	Реализовать таймаут безопасности 580 мс. База шасси $L=0.29$ м. Радиус колеса $R=0.050$ м. Вывести v_left и v_right по UART с частотой 34 Гц.
15	Реализовать таймаут безопасности 600 мс. База шасси $L=0.30$ м. Радиус колеса $R=0.030$ м. Вывести v_left и v_right по UART с частотой 35 Гц.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое `cmd_vel` и какие поля он содержит?
2. Как преобразовать v и ω в скорости левого и правого колёс?
3. Что такое таймаут безопасности и как его реализовать?
4. Как управлять L298N через ШИМ?
5. Как организовать приём UART через прерывания?
6. Как масштабировать линейную скорость в ШИМ?
7. Что происходит при потере связи с Raspberry Pi?
8. Как тестировать управление без физического робота?

Лабораторная работа 8.

УСТАНОВКА ROS 2 НА RASPBERRY PI И НАСТРОЙКА MICRO-ROS АГЕНТА

Цель и содержание работы: Установить ROS 2 Humble на Raspberry Pi, собрать micro-ROS агент и проверить двустороннюю связь по UART между агентом на ОПК и micro-ROS-клиентом на STM32 (или симуляцией).

Теоретическое обоснование

micro-ROS — реализация ROS 2 API для микроконтроллеров с ограниченными ресурсами (STM32, ESP32). Архитектура: микроконтроллер запускает micro-ROS-клиент (rclc), который общается через транспорт (UART/UDP) с micro-ROS агентом на одноплатном компьютере. Агент — мост между миром DDS/ROS 2 и клиентом.

Установка ROS 2 Humble на Raspberry Pi OS 64-bit выполняется из официального репозитория: добавляются ключ и источник, устанавливается ros-humble-ros-base, зависимости: python3-colcon-common-extensions, libssl-dev.

micro-ROS агент собирается из исходников (micro-ROS-Agent) или запускается через Docker. Запуск: `ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyAMA0 -b 115200`. Агент автоматически обнаруживает клиентов.

Пример

```
# Установка ROS 2 Humble на Raspberry Pi (Ubuntu 22.04 или RPi OS 64-bit):  
sudo apt install -y software-properties-common curl  
  
sudo                                curl                                -sSL  
  
https://raw.githubusercontent.com/ros/rosdistro/master/ros.key \  
-o /usr/share/keyrings/ros-archive-keyring.gpg
```

```
echo "deb [arch=arm64 signed-by=...] http://packages.ros.org/ros2/ubuntu  
jammy main" \
```

```
| sudo tee /etc/apt/sources.list.d/ros2.list  
sudo apt update && sudo apt install -y ros-humble-ros-base
```

```
# Сборка micro-ROS агента:  
mkdir -p ~/microros_ws/src && cd ~/microros_ws  
git clone https://github.com/micro-ROS/micro-ROS-Agent.git  
src/micro_ros_agent  
source /opt/ros/humble/setup.bash  
rosdep install --from-paths src -y  
colcon build && source install/setup.bash
```

```
# Запуск агента:  
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyAMA0 -b  
115200
```

```
# Проверка:  
ros2 topic list # должны появиться топики от STM32
```

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
-----------------------	---

1	Вариант 1: Настроить агент на baudrate 67200. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
2	Вариант 2: Настроить агент на baudrate 76800. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
3	Вариант 3: Настроить агент на baudrate 86400. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
4	Вариант 4: Настроить агент на baudrate 96000. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
5	Вариант 5: Настроить агент на baudrate 105600. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
6	Вариант 6: Настроить агент на baudrate 115200. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
7	Вариант 7: Настроить агент на baudrate 124800. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.

8	Вариант 8: Настроить агент на baudrate 134400. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
9	Вариант 9: Настроить агент на baudrate 144000. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
10	Вариант 10: Настроить агент на baudrate 153600. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
11	Вариант 11: Настроить агент на baudrate 163200. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
12	Вариант 12: Настроить агент на baudrate 172800. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
13	Вариант 13: Настроить агент на baudrate 182400. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
14	Вариант 14: Настроить агент на baudrate 192000. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.

15	Вариант 15: Настроить агент на baudrate 201600. Проверить задержку (latency) передачи сообщения: timestamp на STM32 vs timestamp на RPi. Написать Python-скрипт мониторинга топики /odom.
----	---

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое micro-ROS и чем он отличается от ROS 2?
2. Какова роль micro-ROS агента?
3. Как установить ROS 2 Humble на Raspberry Pi?
4. Как собрать micro-ROS агент из исходников?
5. Как запустить агент на UART-порту?
6. Как проверить связь через ros2 topic list?
7. Что такое транспортный уровень micro-ROS?
8. Как настроить /dev/ttyAMA0 на Raspberry Pi?

Лабораторная работа 9.

MICRO-ROS PUBLISHER: ПУБЛИКАЦИЯ ОДОМЕТРИИ НА STM32

Цель и содержание работы: Реализовать micro-ROS узел на STM32, публикующий топик /odom (nav_msgs/Odometry) с данными о положении и скорости робота, полученными из функции одометрии (ЛР 3) и ЕKF (ЛР 5).

Теоретическое обоснование

rcl (ROS Client Library for C) — основная библиотека micro-ROS для C. Создание узла: rcl_node_init_default, publisher: rcl_publisher_init_default. Тип сообщения nav_msgs/msg/Odometry содержит поля: header (stamp, frame_id), child_frame_id, pose.pose (position.x/y, orientation.z/w), twist.twist (linear.x, angular.z).

Цикл micro-ROS: инициализация транспорта → инициализация поддержки DDS → создание узла → создание publisher → цикл публикации → rcl_spin_node_some. Таймер публикации: 20 Гц (50 мс) через rcl_timer_init_default.

На Raspberry Pi данные /odom отображаются в RViz (Add by topic → /odom → Odometry display). Ориентация передаётся в виде кватерниона (для 2D: $q.w = \cos(\theta/2)$, $q.z = \sin(\theta/2)$).

Пример

```
// Инициализация micro-ROS (STM32, main.c):  
rcl_allocator_t allocator = rcl_get_default_allocator();  
rcl_support_t support;  
rcl_support_init(&support, 0, NULL, &allocator);  
rcl_node_t node;  
rcl_node_init_default(&node, "stm32_odom_node", "", &support);  
rcl_publisher_t publisher;
```

```
rcl_publisher_init_default(&publisher, &node,
```

```
ROSIDL_GET_MSG_TYPE_SUPPORT(nav_msgs, msg, Odometry), "/odom");
```

```
// Заполнение сообщения (в прерывании таймера 20 Гц):
```

```
nav_msgs__msg__Odometry odom_msg;
```

```
odom_msg.pose.pose.position.x = odom_x;
```

```
odom_msg.pose.pose.position.y = odom_y;
```

```
odom_msg.pose.pose.orientation.w = cosf(odom_theta / 2.0f);
```

```
odom_msg.pose.pose.orientation.z = sinf(odom_theta / 2.0f);
```

```
odom_msg.twist.twist.linear.x = linear_vel;
```

```
odom_msg.twist.twist.angular.z = angular_vel;
```

```
rcl_publish(&publisher, &odom_msg, NULL);
```

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Публиковать /odom с частотой 12 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 1: child_frame_id = "base_link_1". Визуализировать в RViz, сделать скриншот траектории.
2	Публиковать /odom с частотой 14 Гц. Добавить в header.stamp

	текущее время (UNIX timestamp). Вариант 2: child_frame_id = "base_link_2". Визуализировать в RViz, сделать скриншот траектории.
3	Публиковать /odom с частотой 16 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 3: child_frame_id = "base_link_3". Визуализировать в RViz, сделать скриншот траектории.
4	Публиковать /odom с частотой 18 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 4: child_frame_id = "base_link_4". Визуализировать в RViz, сделать скриншот траектории.
5	Публиковать /odom с частотой 20 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 5: child_frame_id = "base_link_5". Визуализировать в RViz, сделать скриншот траектории.
6	Публиковать /odom с частотой 22 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 6: child_frame_id = "base_link_6". Визуализировать в RViz, сделать скриншот траектории.
7	Публиковать /odom с частотой 24 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 7: child_frame_id = "base_link_7". Визуализировать в RViz, сделать скриншот траектории.
8	Публиковать /odom с частотой 26 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 8: child_frame_id = "base_link_8". Визуализировать в RViz, сделать скриншот траектории.
9	Публиковать /odom с частотой 28 Гц. Добавить в header.stamp

	текущее время (UNIX timestamp). Вариант 9: child_frame_id = "base_link_9". Визуализировать в RViz, сделать скриншот траектории.
10	Публиковать /odom с частотой 30 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 10: child_frame_id = "base_link_10". Визуализировать в RViz, сделать скриншот траектории.
11	Публиковать /odom с частотой 32 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 11: child_frame_id = "base_link_11". Визуализировать в RViz, сделать скриншот траектории.
12	Публиковать /odom с частотой 34 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 12: child_frame_id = "base_link_12". Визуализировать в RViz, сделать скриншот траектории.
13	Публиковать /odom с частотой 36 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 13: child_frame_id = "base_link_13". Визуализировать в RViz, сделать скриншот траектории.
14	Публиковать /odom с частотой 38 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 14: child_frame_id = "base_link_14". Визуализировать в RViz, сделать скриншот траектории.
15	Публиковать /odom с частотой 40 Гц. Добавить в header.stamp текущее время (UNIX timestamp). Вариант 15: child_frame_id = "base_link_15". Визуализировать в RViz, сделать скриншот траектории.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое nav_msgs/Odometry и его основные поля?
2. Как инициализировать rclcpp и создать узел?
3. Как создать publisher в rclcpp?
4. Что такое кватернион и как перевести угол theta в кватернион?
5. Как организовать цикл публикации с частотой 20 Гц?
6. Как убедиться, что топик /odom появился в ros2 topic list?
7. Как визуализировать одометрию в RViz?
8. Как публиковать временную метку (timestamp) в header.stamp?

Лабораторная работа 10.

ПОДКЛЮЧЕНИЕ ЛИДАРА К RASPBERRY PI И ПУБЛИКАЦИЯ /SCAN

Цель и содержание работы: Подключение лидара RPLIDAR A1 к одноплатному компьютеру Raspberry Pi, настройка драйвера для работы с операционной системой ROS 2 (дистрибутив Humble), публикация данных лазерного сканирования в стандартный топик /scan. Изучение структуры сообщения sensor_msgs/LaserScan, визуализация данных в инструменте RViz2 и анализ производительности системы (частота обновления, загрузка процессора). Получение практических навыков интеграции датчика лазерной дальнометрии в робототехническую ROS-экосистему.

Теоретическое обоснование

Лидар (LIDAR, Light Detection and Ranging) — это оптический датчик, который измеряет расстояния до окружающих объектов путем облучения их лазерными импульсами и анализа отражённого сигнала. Инкрементальные лазерные сканеры, такие как RPLIDAR A1 производства компании SLAMTEC, представляют собой 2D-дальномеры, вращающиеся вокруг своей оси, которые формируют круговую карту препятствий в горизонтальной плоскости. Время пролёта импульса не измеряется напрямую, используется более дешёвый триангуляционный метод. Характеристики RPLIDAR A1: дальность обнаружения до 12 метров, частота обновления скан-кадра 5.5–10 Гц (можно программно переключать), угловое разрешение 1 градус, что даёт 360 точек на один полный оборот.

Для интеграции с ROS 2 используется официальный пакет rplidar_ros, который содержит узел-драйвер, работающий через USB/UART-интерфейс. Этот узел публикует данные в стандартном сообщении

sensor_msgs/LaserScan. Данное сообщение включает следующие ключевые поля:

- header: временная метка и система координат (frame_id), в которой измерены расстояния.
- angle_min, angle_max: начальный и конечный угол сканирования (радианы).
- angle_increment: угловое разрешение (rad).
- range_min, range_max: минимальная и максимальная измеряемая дистанция (метры).
- ranges: массив расстояний до препятствий (float[]), где для “бесконечности” (объект за пределами max_range) записывается значение Inf.
- intensities: массив интенсивности отражённого сигнала, для RPLIDAR A1 обычно не используется и заполнен нулями.

ROS 2 Humble — это дистрибутив, который активно поддерживается на Ubuntu 22.04 (Jammy) и является рекомендуемой базовой системой для Raspberry Pi (64-битная версия). Подключение лидара к Raspberry Pi выполняется по USB (микроСабля). В системе устройство появляется как /dev/ttyUSB0. Для работы без прав суперпользователя рекомендуется создать udev-правило, чтобы автоматически выдавать права доступа.

Пример

Рассмотрим настройку лидара RPLIDAR A1 на Raspberry Pi 4 с установленной операционной системой Ubuntu 22.04 Server и ROS 2 Humble.

1. Подготовка системы и установка пакета rplidar_ros.

Откройте терминал на Raspberry Pi. Наиболее простой способ — установка готового deb-пакета (если он доступен в репозиториях):

```
sudo apt update
sudo apt install ros-humble-rplidar-ros
```

Если deb-пакет отсутствует, выполняется сборка из исходного кода в рабочем пространстве colcon:

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws/src
git clone -b ros2 https://github.com/Slamtec/rplidar_ros.git
cd ~/ros2_ws
colcon build --packages-select rplidar_ros
source ~/ros2_ws/install/setup.bash
```

2. Назначение прав доступа к последовательному порту.

Подключите лидар к USB-порту Raspberry Pi. Убедитесь, что система его обнаружила:

```
ls -l /dev/ttyUSB0
```

Выдайте права доступа:

```
sudo chmod 666 /dev/ttyUSB0
```

Для постоянного применения прав создайте udev-правило. Перейдите в папку исходного кода пакета и выполните скрипт:

```
cd ~/ros2_ws/src/rplidar_ros
sudo cp scripts/rplidar.rules /etc/udev/rules.d/
sudo udevadm control --reload-rules
sudo udevadm trigger
```

3. Проверка публикации топика /scan.

Запустите узел лидара для модели A1:

```
ros2 launch rplidar_ros rplidar_a1_launch.py
```

В другом терминале (или через SSH) проверьте список активных топиков:

```
ros2 topic list
```

Вы должны увидеть топик /scan. Просмотрите содержимое одного сообщения:

```
ros2 topic echo /scan --once
```

4. Анализ частоты обновления.

Для измерения частоты публикации данных используйте команду:

```
ros2 topic hz /scan
```

Средняя частота для RPLIDAR A1 должна составлять приблизительно 8–10 Гц.

5. Визуализация в RViz2.

Запустите RViz2:

```
rviz2
```

В графическом интерфейсе выполните настройки:

- В левой панели Global Options установите Fixed Frame: laser_frame.
- Нажмите кнопку Add (внизу слева).
- В открывшемся диалоговом окне перейдите на вкладку By topic.
- Выберите топик /scan и тип отображения LaserScan.
- Нажмите ОК.

На экране должна появиться карта лазерных точек, отображающая контуры окружающих объектов.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
---------------	------------------------------------

1	Установите пакет <code>rplidar_ros</code> на Raspberry Pi, используя сборку из исходного кода. Измените в файле запуска параметр <code>serial_baudrate</code> с 115200 на 230400. Проверьте, изменилась ли частота обновления топика <code>/scan</code> . Объясните результат.
2	Для RPLIDAR A1 создайте собственный <code>launch</code> -файл, который запускает узел драйвера и RViz2 одновременно. В качестве <code>frame_id</code> для лидара используйте значение <code>"laser_link"</code> . Добавьте статическое преобразование координат между <code>base_link</code> и <code>laser_link</code> с помощью узла <code>tf2_ros</code> .
3	Напишите на Python простой подписчик на топик <code>/scan</code> . В функции обратного вызова вычислите минимальное расстояние до препятствия из массива <code>ranges</code> . Выводите это значение в терминал каждую секунду.
4	Используйте команду <code>ros2 topic echo</code> для перенаправления данных из топика <code>/scan</code> в текстовый файл в течение 10 секунд. Разберите структуру полученного файла. Найдите в нём значения расстояний, соответствующие углам 0, 90, 180 и 270 градусов.
5	Подключите к Raspberry Pi лидар YDLIDAR X4. Установите пакет <code>ydlidar_ros2_driver</code> из исходного кода. Настройте запуск для публикации топика <code>/scan</code> . Сравните процедуру настройки с установкой RPLIDAR.
6	Реализуйте фильтр, который публикует новый топик <code>/scan_filtered</code> . Новый топик должен содержать только те измерения, где расстояние находится в пределах от 0.2 до 3.5 метров, отбрасывая слишком близкие и слишком далёкие точки. Напишите узел на C++.
7	Измерьте нагрузку на центральный процессор Raspberry Pi при работающем драйвере лидара (команда <code>top</code> или <code>htop</code>). Запустите

	одновременно RViz2 и снова измерьте нагрузку. Сделайте вывод о потреблении ресурсов системой визуализации.
8	Измените частоту вращения двигателя лидара программно через изменение параметра <code>frequency</code> в <code>launch</code> -файле (модели A2 и выше). Сравните визуальное качество точечного облака в RViz2 на частотах 5 Гц и 10 Гц.
9	Подключите лидар к Raspberry Pi не напрямую, а через USB-концентратор. Запустите драйвер и проанализируйте, появляются ли дополнительные ошибки чтения или пропуски кадров по сравнению с прямым подключением. Задокументируйте результаты.
10	Установите пакет <code>rplidar_ros</code> на рабочую станцию x86_64 с Ubuntu 22.04 и ROS 2 Humble. Подключите лидар к этому компьютеру. Убедитесь, что данные публикуются. Сравните производительность (частоту и задержку) с Raspberry Pi.
11	Создайте сервис, который останавливает вращение мотора лидара с помощью вызова сервиса <code>/stop_motor</code> . Разработайте второй сервис для включения мотора (<code>/start_motor</code>). Напишите простую командную строку на Python для управления лидаром.
12	Запустите лидар и одновременно включите запись всех топиков в файл мешка (ROS bag) на одну минуту. Воспроизведите этот мешок на другом компьютере без физического лидара. Проверьте, что RViz2 корректно отображает записанные данные.
13	Установите пакет <code>slam_toolbox</code> на Raspberry Pi. Запустите узел лазерной одометрии (<code>rf2o_laser_odometry</code>), используя топик <code>/scan</code> . Визуализируйте в RViz2 одновременно лазерные точки и оценённую траекторию движения робота.
14	В пакете <code>rplidar_ros</code> измените файл параметров так, чтобы лидар

	игнорировал показания интенсивности (intensities). Убедитесь, что в топике /scan массив intensities заполнен нулями. Перекомпилируйте пакет и проверьте.
15	Разработайте систему оповещения о препятствиях: узел ROS 2 подписывается на /scan, и если в любом направлении расстояние до препятствия становится меньше заданного порога (например, 0.8 м), узел публикует предупреждение в отдельный топик /warnings и выводит сообщение в консоль.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое лидар и какой топик используется для его данных?
2. Что такое RPLIDAR A1 и как его подключить к Raspberry Pi?
3. Какова структура сообщения sensor_msgs/LaserScan?
4. Как запустить узел rplidar_ros?
5. Как измерить частоту публикации (ros2 topic hz)?
6. Как визуализировать данные лидара в RViz?
7. Что означают значения ranges[] в LaserScan?
8. Как настроить frame_id для TF?

Лабораторная работа 11.

ОБРАБОТКА СКАНОВ ЛИДАРА: ИЗБЕГАНИЕ ПРЕПЯТСТВИЙ

Цель и содержание работы: Разработка простого ROS 2 узла на языке Python или C++, который выполняет обработку данных лазерного сканирования (топик /scan) для обнаружения препятствий перед роботом. По результатам анализа узел публикует команды управления в топик /cmd_vel: при отсутствии препятствий робот движется вперёд с заданной скоростью, а при обнаружении препятствия на расстоянии менее порогового значения (0.5 метра) – немедленная остановка. Проверка работоспособности в симуляционной среде (например, Gazebo с готовой моделью робота, имеющего лидар).

Теоретическое обоснование

Для автономной навигации мобильного робота критически важна способность обнаруживать препятствия и реагировать на них. Самым распространённым датчиком для этой задачи является 2D лидар, предоставляющий круговой обзор в горизонтальной плоскости в виде массива расстояний (ranges). ROS 2 предоставляет стандартизированный интерфейс – сообщение sensor_msgs/LaserScan, которое публикуется драйверами лидаров и симуляторами.

Сообщение LaserScan содержит:

- angle_min, angle_max: начальный и конечный углы сканирования (радианы).
- angle_increment: угловой шаг между последовательными измерениями.
- ranges: массив расстояний (float), где значение Inf или NaN означает отсутствие препятствия или невалидное измерение.

Для простого избегания препятствий достаточно проанализировать сектор перед роботом. Обычно это углы от $-\alpha$ до $+\alpha$ относительно направления движения (например, от -30 до $+30$ градусов). В этом секторе находится минимальное расстояние `min_distance`. Если `min_distance < threshold` (например, 0.5 м), робот должен остановиться, иначе – продолжать движение (например, двигаться вперёд с фиксированной линейной скоростью). Такой алгоритм называется реактивным (или рефлексивным) поведением. Он не требует построения карты и планирования пути, но достаточен для простых сценариев и для знакомства с обработкой сенсорных данных в ROS.

Для публикации команд управления используется сообщение `geometry_msgs/Twist`, содержащее линейную скорость (`linear.x`) и угловую скорость (`angular.z`). Управление в симуляции или на реальном роботе осуществляется через драйвер (например, узел, преобразующий `cmd_vel` в ШИМ на STM32).

В рамках работы узел может быть реализован как на C++ (более высокая производительность) с использованием компонентной архитектуры или как на Python (быстрая разработка). В примере используется Python для наглядности.

Пример

Рассмотрим создание узла на Python с именем `obstacle_avoider`. Рабочее пространство ROS 2 на Raspberry Pi (или на PC) предполагается уже настроенным, лидар публикует топик `/scan` (LP10). Симуляция может быть заменена реальными данными.

Создание пакета и узла:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python obstacle_avoider
cd obstacle_avoider/obstacle_avoider
nano obstacle_avoider_node.py
```

Листинг узла:

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
import math

class ObstacleAvoider(Node):
    def __init__(self):
        super().__init__('obstacle_avoider')
        # Подписка на топик лидара
        self.scan_sub = self.create_subscription(LaserScan, 'scan',
self.scan_callback, 10)
        # Публикация команд скорости
        self.cmd_pub = self.create_publisher(Twist, 'cmd_vel', 10)
        # Параметры узла
        self.declare_parameter('forward_speed', 0.2) # м/с
        self.declare_parameter('safe_distance', 0.5) # м
        self.declare_parameter('front_angle', 30.0) # градусы (половина
сектора)

        self.forward_speed = self.get_parameter('forward_speed').value
        self.safe_distance = self.get_parameter('safe_distance').value
        self.front_angle_rad =
math.radians(self.get_parameter('front_angle').value)
        self.get_logger().info('Obstacle avoider node started')

    def scan_callback(self, msg: LaserScan):
        # Определяем индекс начала и конца сектора перед роботом
        # Угол 0 соответствует направлению вперёд (ось X робота)
        # Сообщение лидара: angle_min, angle_increment, ranges
        # Находим индекс, соответствующий углу -front_angle_rad и
+front_angle_rad
        angle_min = msg.angle_min
        angle_inc = msg.angle_increment
        start_index = int((-self.front_angle_rad - angle_min) / angle_inc)
        end_index = int((self.front_angle_rad - angle_min) / angle_inc)

        # Ограничиваем индексы допустимыми пределами
        start_index = max(0, start_index)
        end_index = min(len(msg.ranges) - 1, end_index)

        # Поиск минимального расстояния в секторе
        min_distance = float('inf')
        for i in range(start_index, end_index + 1):
            dist = msg.ranges[i]
            if math.isnan(dist) or math.isinf(dist):
                continue
            if dist < min_distance:
                min_distance = dist

        # Принимаем решение
        cmd = Twist()
        if min_distance < self.safe_distance:
            # Останавливаемся
            cmd.linear.x = 0.0
            cmd.angular.z = 0.0
            self.get_logger().warn(f'Obstacle detected at {min_distance:.2f}m,
stopping!')
        else:
            # Движение вперёд
            cmd.linear.x = self.forward_speed
```

```

        cmd.angular.z = 0.0
        self.get_logger().debug(f'Free space, min distance
{min_distance:.2f}m')

        self.cmd_pub.publish(cmd)

def main(args=None):
    rclpy.init(args=args)
    node = ObstacleAvoider()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()

```

Настройка файла setup.py (добавление точки входа):

```

entry_points={
    'console_scripts': [
        'obstacle_avoider = obstacle_avoider.obstacle_avoider_node:main',
    ],
}

```

Сборка и запуск:

```

cd ~/ros2_ws
colcon build --packages-select obstacle_avoider
source install/setup.bash
ros2 run obstacle_avoider obstacle_avoider --ros-args -p forward_speed:=0.3 -p
safe_distance:=0.6

```

Проверка в симуляции (например, Gazebo с роботом turtlebot3):

- Запустить симуляцию: `ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py`
- Запустить лидар (если не запущен автоматически): он уже есть у TurtleBot3.
- Запустить узел `obstacle_avoider`.
- В симуляции расставить препятствия (ящики) — робот должен останавливаться перед ними.

Если реального робота с лидаром нет, можно использовать записанный мешок (bag) с топиком `/scan` и воспроизводить его, а узел запускать офлайн.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Реализовать узел на C++ вместо Python. Использовать <code>rclcpp::Node</code> , подписку на <code>/scan</code> , публикацию в <code>/cmd_vel</code> . Порог остановки 0.4 м, скорость вперёд 0.25 м/с. Угол обзора перед роботом: ± 45 градусов.
2	Добавить в алгоритм поворот на месте при обнаружении препятствия. Если минимальное расстояние меньше порога, робот не останавливается, а поворачивается на месте с угловой скоростью 0.5 рад/с в течение 2 секунд, затем продолжает движение вперёд.
3	Вместо фиксированной скорости вперёд, сделать её зависящей от минимального расстояния: $\text{скорость} = \min(\text{forward_speed}, (\text{min_distance} - 0.2) / \text{safe_distance} * \text{forward_speed})$. При $\text{min_distance} < 0.2$ м скорость обнуляется.
4	Анализировать сектор не только перед роботом, но и слева и справа (три сектора). При обнаружении препятствия спереди, робот поворачивает в сторону, где расстояние больше. Реализовать выбор направления поворота.
5	Узел должен игнорировать измерения, где расстояние больше 3

	метров (эффективная дальность лидара). В секторе перед роботом заменить такие значения на Inf. Проверить корректность минимума.
6	Добавить отладочную публикацию маркеров в RViz: опубликовать маркер в виде точки или линии, показывающей минимальное расстояние в секторе перед роботом. Использовать топик <code>visualization_marker</code> .
7	Реализовать два порога: дальний (0.8 м) – снижать скорость до 0.1 м/с, ближний (0.3 м) – полная остановка. Скорость переключается плавно (как в варианте 3, но с двумя зонами).
8	Вместо фиксированной скорости движения вперёд, узел должен подписываться на топик <code>/desired_speed</code> , который задаёт желаемую линейную скорость из другого узла (например, телеоперации). Ограничение скорости применяется только при наличии препятствия.
9	Узел должен быть компонентом (<code>composable node</code>) для использования в составе одной системы с драйвером лидара. Запускать через <code>launch</code> -файл, загружая компонент в контейнер.
10	Анализировать не только переднюю зону, но и всю окружность. Если где-либо (кроме задней зоны) расстояние меньше порога, остановить робота. При этом заднюю зону (например, -150...150 градусов) игнорировать, чтобы робот мог двигаться задним ходом.
11	Добавить фильтрацию выбросов: для каждого измерения в секторе перед роботом применить медианный фильтр по 5 соседним точкам (соседние по углу). Только после этого искать минимум.
12	Использовать качество сигнала (<code>intensities</code>) для отбрасывания

	“шумных” измерений. Если интенсивность ниже заданного порога (например, 10), не учитывать это расстояние при поиске минимума.
13	Реализовать плавную остановку: при обнаружении препятствия не обнулять скорость мгновенно, а уменьшать её с заданным замедлением (0.2 м/с^2). Для этого использовать таймер и переменную состояния.
14	Узел должен также публиковать диагностический топик /obstacle_distance (тип std_msgs/Float32) с текущим минимальным расстоянием. Запустить другой узел-подписчик, который при значении меньше 0.2 м выводит звуковой сигнал на Raspberry Pi (через beep).
15	Использовать для обработки сканов метод, основанный на разбиении сектора на три подсектора (левый, центральный, правый). Если центральный подсектор свободен – ехать вперёд. Если занят – поворачивать в сторону свободного подсектора. Скорость поворота пропорциональна степени занятости.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Как подписаться на /scan и обработать данные?
2. Как выбрать зону перед роботом (от -30° до $+30^\circ$)?
3. Что такое ranges[i] и как найти минимальное расстояние?

4. Как публиковать `cmd_vel` для остановки?
5. Как настроить порог остановки (`threshold`)?
6. Как тестировать узел в симуляции?
7. Как обработать значения `inf` или `nan` в `ranges`?
8. Как реализовать торможение (замедление перед порогом)?

Лабораторная работа 12.

ПОДКЛЮЧЕНИЕ КАМЕРЫ И БАЗОВАЯ ОБРАБОТКА ИЗОБРАЖЕНИЙ (CANNY)

Цель и содержание работы: Подключение камеры (CSI или USB) к одноплатному компьютеру Raspberry Pi, захват видеопотока с использованием библиотеки OpenCV, выполнение базовой обработки изображения – выделение границ (edges) с помощью алгоритма Канни (Canny). Публикация исходного и обработанного изображений в топики ROS 2 (sensor_msgs/Image) с применением утилиты cv_bridge для преобразования между форматами OpenCV и ROS. Получение навыков построения пайплайна компьютерного зрения в составе робототехнической системы.

Теоретическое обоснование

Для задач навигации, обнаружения объектов и слежения роботы обычно оснащаются камерами. Raspberry Pi может работать с двумя типами камер: официальная камера Raspberry Pi (CSI-интерфейс) и стандартные USB-камеры (веб-камеры). CSI-камера подключается через специальный разъем и обеспечивает более высокую производительность, так как использует аппаратное кодирование. USB-камеры проще в подключении, но могут создавать дополнительную нагрузку на процессор.

Библиотека OpenCV (Open Source Computer Vision Library) предоставляет широкий набор алгоритмов обработки изображений. В данной работе используется детектор границ Канни – многоступенчатый алгоритм, включающий:

- Преобразование в оттенки серого для уменьшения размерности данных.
- Гауссово размытие (GaussianBlur) для подавления шума.

- Вычисление градиента (величина и направление) с помощью операторов Собеля.
- Подавление немаксимумов для уточнения границ.
- Двойная пороговая фильтрация для выделения сильных и слабых границ.
- Трассировка связности для формирования окончательных контуров.

Результатом являются бинарные изображения, где белыми пикселями отмечены границы объектов.

В экосистеме ROS 2 для передачи изображений используется сообщение `sensor_msgs/Image`. Для преобразования между изображениями OpenCV (numpy-массивы) и ROS-сообщениями применяется библиотека `cv_bridge`, которая входит в состав пакета `vision_opencv`. Это позволяет публиковать обработанные кадры для дальнейшего использования другими узлами (например, для детектирования линий или препятствий).

В рамках работы создается Python-узел ROS 2, который:

- 1. Захватывает кадры с камеры через `cv2.VideoCapture`.**
- 2. Для каждого кадра выполняет преобразование в серый, размытие и детектор Канни.**
- 3. Публикует исходный и обработанный кадры в отдельные топики.**
- 4. Также может отображать кадры в окне OpenCV (`cv2.imshow`) для локальной отладки.**

Пример

Рассмотрим реализацию узла ROS 2 на Python для Raspberry Pi с подключенной USB-камерой. Предполагается, что ROS 2 Humble установлен, рабочее пространство создано. Требуется установить пакеты: `python3-opencv`, `ros-humble-cv-bridge`, `ros-humble-vision-msgs`.

Создание пакета и скрипта:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python camera_processing
cd camera_processing/camera_processing
nano camera_canny_node.py
```

Листинг узла:

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import numpy as np

class CameraCannyNode(Node):
    def __init__(self):
        super().__init__('camera_canny_node')
        # Параметры
        self.declare_parameter('camera_id', 0)          # ID устройства (0 для
/dev/video0)
        self.declare_parameter('canny_low', 50)
        self.declare_parameter('canny_high', 150)
        self.declare_parameter('blur_kernel', 5)        # Размер ядра размытия
(нечетное)
        self.declare_parameter('publish_rate', 10.0)    # Частота публикации
(Гц)
        self.declare_parameter('show_window', True)     # Показывать ли окно
OpenCV

        self.camera_id = self.get_parameter('camera_id').value
        self.canny_low = self.get_parameter('canny_low').value
        self.canny_high = self.get_parameter('canny_high').value
        self.blur_kernel = self.get_parameter('blur_kernel').value
        self.publish_rate = self.get_parameter('publish_rate').value
        self.show_window = self.get_parameter('show_window').value

        # Инициализация камеры
        self.cap = cv2.VideoCapture(self.camera_id)
        if not self.cap.isOpened():
            self.get_logger().error('Failed to open camera')
            raise RuntimeError('Camera not accessible')

        # Мост OpenCV<->ROS
        self.bridge = CvBridge()

        # Публикаторы
        self.raw_pub = self.create_publisher(Image, 'camera/raw', 10)
        self.canny_pub = self.create_publisher(Image, 'camera/canny', 10)

        # Таймер для периодического захвата
        timer_period = 1.0 / self.publish_rate if self.publish_rate > 0 else 0.1
        self.timer = self.create_timer(timer_period, self.timer_callback)

        self.get_logger().info('Camera Canny node started')

    def timer_callback(self):
        ret, frame = self.cap.read()
        if not ret:
```

```

        self.get_logger().warn('Failed to grab frame')
        return

    # Преобразование в оттенки серого
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Размытие для уменьшения шума
    blurred = cv2.GaussianBlur(gray, (self.blur_kernel, self.blur_kernel),
0)

    # Детектор границ Канни
    edges = cv2.Canny(blurred, self.canny_low, self.canny_high)

    # Публикация исходного изображения (BGR)
    raw_msg = self.bridge.cv2_to_imgmsg(frame, encoding='bgr8')
    self.raw_pub.publish(raw_msg)

    # Публикация изображения границ (монохромное)
    canny_msg = self.bridge.cv2_to_imgmsg(edges, encoding='mono8')
    self.canny_pub.publish(canny_msg)

    # Отображение в окне (опционально)
    if self.show_window:
        cv2.imshow('Camera View', frame)
        cv2.imshow('Canny Edges', edges)
        cv2.waitKey(1) # Необходимо для обновления окон

def __del__(self):
    if self.cap.isOpened():
        self.cap.release()
    if self.show_window:
        cv2.destroyAllWindows()

def main(args=None):
    rclpy.init(args=args)
    node = CameraCannyNode()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()

```

Настройка файла setup.py (добавление точки входа):

```

entry_points={
    'console_scripts': [
        'camera_canny = camera_processing.camera_canny_node:main',
    ],
},

```

Сборка и запуск:

```

cd ~/ros2_ws
colcon build --packages-select camera_processing
source install/setup.bash
ros2 run camera_processing camera_canny --ros-args -p camera_id:=0 -p
canny_low:=50 -p canny_high:=150 -p publish_rate:=15

```

Для просмотра публикуемых топиков в другом терминале:

```
ros2 topic list
ros2 topic echo /camera/canny --once | less
ros2 run rqt_image_view rqt_image_view
```

Если используется Raspberry Pi Camera (CSI), необходимо заменить VideoCapture на пикамеру. Для этого можно использовать библиотеку picamera2. Пример адаптации:

```
from picamera2 import Picamera2
picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"format": 'RGB888'}))
picam2.start()
frame = picam2.capture_array() # получаем numpy массив RGB
```

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Использовать USB-камеру с разрешением 640x480. Публиковать только обработанное изображение (Canny) в топик /canny_image. Частота публикации 5 Гц. Добавить параметр порога для canny_low и canny_high, передаваемый через командную строку.
2	Подключить официальную камеру Raspberry Pi (CSI) через библиотеку picamera2. Выполнить те же операции: серый, размытие, Канни. Публиковать в топик /camera/csi/raw и

	/camera/csi/edges. Убедиться, что частота не падает ниже 10 Гц.
3	Добавить в узел возможность выбора алгоритма выделения границ: Canny или Sobel (cv2.Sobel). Переключать через параметр edge_method ('canny' или 'sobel'). При выборе Sobel выводить изображение с градиентом по оси X (cv2.convertScaleAbs).
4	Публиковать также изображение после размытия (GaussianBlur) в отдельный топик /camera/blurred. Размер ядра размытия сделать параметром (нечётное число от 3 до 15). Проанализировать влияние размера ядра на качество границ.
5	Вместо публикации в топик, сохранять каждое 100-е обработанное изображение на диск (в формате .png) в папку ~/captures. Имя файла: timestamp_edges.png. Одновременно продолжать публикацию в топик для визуализации.
6	Написать второй узел (Python), который подписывается на топик /camera/canny и выполняет поиск контуров (cv2.findContours) на бинарном изображении. Выводить количество найденных контуров в консоль.
7	Реализовать динамическое изменение параметров детектора Канни через сервис: объявить сервис с полями low_threshold, high_threshold. При вызове сервиса параметры должны обновляться на лету без перезапуска узла.
8	Добавить в узел захват видео с удаленной IP-камеры (RTSP). Использовать cv2.VideoCapture("rtsp://user:pass@ip:port/stream"). Параметры подключения передавать через параметры узла. Проверить работоспособность.
9	Создать launch-файл, который запускает узел обработки камеры и одновременно rqt_image_view с предварительно настроенными подписками на топики /camera/raw и /camera/canny.

10	Перед детектором Канни добавить адаптивную пороговую обработку (<code>cv2.adaptiveThreshold</code>) в качестве альтернативы размытию. Сравнить качество выделения границ при разном освещении.
11	Развернуть обработку изображений в отдельном потоке (<code>Thread</code>) для увеличения частоты кадров. Таймер должен триггерить только захват, а публикация выполняться в потоке. Измерить прирост производительности.
12	Публиковать вместе с изображениями временную метку из ROS (<code>header.stamp</code>), синхронизированную с системным временем. Использовать <code>this->get_clock()->now()</code> и заполнять поле <code>stamp</code> сообщения <code>Image</code> .
13	Реализовать обнаружение движения: вычитать предыдущий кадр из текущего (<code>cv2.absdiff</code>), затем пороговую обработку и поиск контуров. Публиковать изображение разности и флаг движения (<code>true/false</code>) в отдельный топик.
14	Использовать GPU-ускорение OpenCV на Raspberry Pi (если доступно) для ускорения детектора Канни. Настроить сборку OpenCV с поддержкой OpenCL и сравнить производительность с программной реализацией.
15	Интегрировать узел обработки камеры с ранее созданным узлом избегания препятствий (JP11). Вместо лидара использовать данные с камеры: выделить область перед роботом (нижняя половина кадра) и оценить занятость пространства по количеству граничных пикселей. При превышении порога останавливать робота.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Как захватить изображение с камеры через OpenCV?
2. Как выполнить детекцию границ Canny?
3. Как опубликовать изображение в ROS 2 через cv_bridge?
4. Что такое sensor_msgs/Image?
5. Как преобразовать цветное изображение в оттенки серого?
6. Какие параметры имеет алгоритм Canny?
7. Как настроить разрешение камеры для экономии ресурсов?
8. Как проверить задержку публикации изображений?

Лабораторная работа 13.

ДЕТЕКЦИЯ ОБЪЕКТОВ: YOLOV8N НА RASPBERRY PI

Цель и содержание работы: Разработка ROS 2 узла на языке Python для обнаружения объектов (мяч, человек, конус) с помощью нейросетевой модели YOLOv8n на одноплатном компьютере Raspberry Pi. Интеграция пайплайна компьютерного зрения: захват кадров с камеры через топик /camera/raw, выполнение инференса модели, фильтрация детекций по порогу достоверности (0.5), публикация координат центров обнаруженных объектов в пользовательский топик /detection. Ограничение частоты обработки до 5–10 кадров в секунду для обеспечения стабильной работы в реальном времени.

Теоретическое обоснование

Современные системы компьютерного зрения на мобильных роботах часто строятся по распределённой архитектуре, где задачи низкоуровневого управления и сбора данных выполняются микроконтроллером (STM32), а задачи обработки изображений и принятия решений – одноплатным компьютером (Raspberry Pi). ROS 2 предоставляет удобную среду для организации такого взаимодействия.

YOLO (You Only Look Once) – это семейство алгоритмов обнаружения объектов в реальном времени, основанных на единой свёрточной нейронной сети. YOLOv8n (nano) – наиболее лёгкая версия модели YOLOv8, предназначенная для работы на устройствах с ограниченными вычислительными ресурсами. Модель YOLOv8n имеет около 3.2 миллиона параметров и обеспечивает баланс между точностью и скоростью инференса.

На Raspberry Pi производительность YOLOv8n зависит от версии платы. Для Raspberry Pi 4 ожидаемая частота кадров составляет 5–15 FPS при входном разрешении 320x240 пикселей [0†L36-L37]. Для Raspberry Pi 5 с более мощным процессором (2.4 ГГц) и улучшенным GPU,

производительность может быть выше, но всё же ограничена [13†L38-L40]. Для обеспечения стабильной работы и снижения нагрузки на CPU следует уменьшать разрешение входного изображения (например, до 320x240) и ограничивать частоту обработки до 5–10 Гц.

Библиотека Ultralytics предоставляет простой API для загрузки и выполнения инференса модели YOLO, а также инструменты для обучения на пользовательских наборах данных. Интеграция с ROS 2 выполняется через `cv_bridge`, который преобразует сообщения типа `sensor_msgs/Image` в изображения OpenCV (numpy-массивы), и наоборот.

Пользовательское сообщение для публикации координат обнаруженного объекта должно содержать:

- `header`: временная метка и система координат (`frame_id`).
- `object_class`: строка с именем класса (например, "person", "ball").
- `confidence`: значение достоверности (`float`).
- `x`, `y`: координаты центра ограничивающего прямоугольника в пикселях (`int`).
- `width`, `height`: размеры ограничивающего прямоугольника (`int`).

Для сокращения объёма передаваемых данных можно публиковать только координаты наиболее уверенного объекта, либо агрегировать несколько объектов в одном сообщении (массив). В примере используется публикация одного сообщения на каждый обнаруженный объект, но с ограничением максимального количества объектов (например, не более 5).

Пример

Рассмотрим создание ROS 2 узла на Python для детекции объектов YOLOv8n на Raspberry Pi 4/5. Предполагается, что ROS 2 Humble установлен, камера публикует топик `/camera/raw` (LP12). Требуется установить библиотеку `ultralytics`.

Создание пользовательского сообщения:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_cmake detection_msgs
cd detection_msgs
mkdir msg
nano msg/Detection.msg
```

Содержание Detection.msg:

```
std_msgs/Header header
string object_class
float32 confidence
int32 x
int32 y
int32 width
int32 height
```

Настройка CMakeLists.txt и package.xml:

- В CMakeLists.txt добавить:

```
find_package(rosidl_default_generators REQUIRED)
find_package(std_msgs REQUIRED)
rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/Detection.msg"
  DEPENDENCIES std_msgs
)
```

- В package.xml добавить:

```
<build_depend>rosidl_default_generators</build_depend>
<exec_depend>rosidl_default_runtime</exec_depend>
<depend>std_msgs</depend>
```

Сборка пользовательского сообщения:

```
cd ~/ros2_ws
colcon build --packages-select detection_msgs
source install/setup.bash
```

Создание пакета детекции:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python yolo_detector --dependencies rclpy
sensor_msgs cv_bridge detection_msgs
cd yolo_detector/yolo_detector
nano detector_node.py
```

Листинг узла detector_node.py:

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from detection_msgs.msg import Detection
from ultralytics import YOLO
import cv2
```

```

import numpy as np
from rclpy.qos import QoSProfile, ReliabilityPolicy

class YoloDetector(Node):
    def __init__(self):
        super().__init__('yolo_detector')
        # Параметры узла
        self.declare_parameter('model_path', 'yolov8n.pt')
        self.declare_parameter('confidence_threshold', 0.5)
        self.declare_parameter('processing_width', 320)
        self.declare_parameter('processing_height', 240)
        self.declare_parameter('max_detections', 5)
        self.declare_parameter('publish_rate', 5.0) # Гц

        model_path = self.get_parameter('model_path').value
        self.conf_threshold = self.get_parameter('confidence_threshold').value
        self.proc_width = self.get_parameter('processing_width').value
        self.proc_height = self.get_parameter('processing_height').value
        self.max_detections = self.get_parameter('max_detections').value
        self.publish_rate = self.get_parameter('publish_rate').value

        # Загрузка модели YOLO
        self.model = YOLO(model_path)
        self.get_logger().info('YOLO model loaded')

        # Мост OpenCV-ROS
        self.bridge = CvBridge()

        # QoS профиль для надёжной доставки
        qos = QoSProfile(depth=10, reliability=ReliabilityPolicy.BEST_EFFORT)

        # Подписка на сырое изображение
        self.sub = self.create_subscription(Image, '/camera/raw',
self.image_callback, qos)

        # Публикатор детекций
        self.detection_pub = self.create_publisher(Detection, '/detection', 10)

        # Публикатор аннотированного изображения (опционально)
        self.annotated_pub = self.create_publisher(Image, '/camera/detections',
10)

        # Таймер для ограничения частоты обработки
        self.last_process_time = self.get_clock().now()
        self.timer = self.create_timer(1.0 / self.publish_rate,
self.timer_callback)

        # Буфер последнего кадра
        self.last_frame = None
        self.frame_available = False
        self.get_logger().info('YOLO Detector node started')

    def image_callback(self, msg):
        # Принимаем кадр, сохраняем в буфер
        try:
            self.last_frame = self.bridge.imgmsg_to_cv2(msg, 'bgr8')
            self.frame_available = True
        except Exception as e:
            self.get_logger().error(f'Conversion error: {e}')

    def timer_callback(self):
        if not self.frame_available:
            return

```

```

# Защита от слишком частой обработки
now = self.get_clock().now()
dt = (now - self.last_process_time).nanoseconds / 1e9
if dt < (1.0 / self.publish_rate):
    return
self.last_process_time = now

frame = self.last_frame
self.frame_available = False

# Масштабирование для ускорения инференса
h, w = frame.shape[:2]
scale_w = w / self.proc_width
scale_h = h / self.proc_height
resized = cv2.resize(frame, (self.proc_width, self.proc_height))

# Выполнение детекции
results = self.model(resized, conf=self.conf_threshold, verbose=False)
result = results[0]

# Аннотация кадра (для публикации/отладки)
annotated = frame.copy()

# Обработка обнаруженных объектов
boxes = result.boxes
if boxes is not None and len(boxes) > 0:
    for i in range(min(len(boxes), self.max_detections)):
        # Координаты в масштабе resized
        x1, y1, x2, y2 = map(int, boxes.xyxy[i].cpu().numpy())
        conf = float(boxes.conf[i].cpu().numpy())
        cls_id = int(boxes.cls[i].cpu().numpy())
        cls_name = self.model.names[cls_id]

        # Пересчёт в масштаб оригинального кадра
        orig_x1 = int(x1 * scale_w)
        orig_y1 = int(y1 * scale_h)
        orig_x2 = int(x2 * scale_w)
        orig_y2 = int(y2 * scale_h)

        # Вычисление центра
        center_x = (orig_x1 + orig_x2) // 2
        center_y = (orig_y1 + orig_y2) // 2
        width = orig_x2 - orig_x1
        height = orig_y2 - orig_y1

        # Публикация сообщения Detection
        msg = Detection()
        msg.header.stamp = self.get_clock().now().to_msg()
        msg.header.frame_id = 'camera_frame'
        msg.object_class = cls_name
        msg.confidence = conf
        msg.x = center_x
        msg.y = center_y
        msg.width = width
        msg.height = height
        self.detection_pub.publish(msg)

        # Рисование на аннотации
        cv2.rectangle(annotated, (orig_x1, orig_y1), (orig_x2, orig_y2),
(0, 255, 0), 2)

        label = f'{cls_name} ({conf:.2f})'
        cv2.putText(annotated, label, (orig_x1, orig_y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
        # Отметка центра
        cv2.circle(annotated, (center_x, center_y), 3, (0, 0, 255), -1)

```

```

        # Публикация аннотированного изображения (опционально)
        try:
            annotated_msg = self.bridge.cv2_to_imgmsg(annotated, 'bgr8')
            annotated_msg.header.stamp = self.get_clock().now().to_msg()
            annotated_msg.header.frame_id = 'camera_frame'
            self.annotated_pub.publish(annotated_msg)
        except Exception as e:
            self.get_logger().error(f'Publish annotation error: {e}')

def main(args=None):
    rclpy.init(args=args)
    node = YoloDetector()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()

```

Настройка файла setup.py (добавление точки входа):

```

entry_points={
    'console_scripts': [
        'detector = yolo_detector.detector_node:main',
    ],
},

```

Установка зависимостей (на Raspberry Pi):

```

pip3 install ultralytics opencv-python-headless
sudo apt install python3-pip

```

Сборка и запуск:

```

cd ~/ros2_ws
colcon build --packages-select detection_msgs yolo_detector
source install/setup.bash
ros2 run yolo_detector detector --ros-args -p confidence_threshold:=0.5 -p
processing_width:=320

```

Для просмотра публикуемых детекций:

```

ros2 topic echo /detection
ros2 run rqt_image_view rqt_image_view # выбрать топик /camera/detections

```

Важное замечание по производительности:

- Снижение processing_width/height ускоряет инференс, но снижает точность.

- Увеличение `confidence_threshold` фильтрует ложные срабатывания, но может пропускать мелкие объекты.

- Частота публикации ограничена таймером, так как модель может не успевать обрабатывать каждый кадр.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Использовать модель YOLOv8n, предварительно обученную на пользовательском наборе данных (например, конусы и мячи). Загрузить модель из файла <code>custom.pt</code> . Изменить код узла для работы с этими классами. Выводить только объекты, соответствующие заданным.
2	Публиковать детекции не в отдельный топик, а в агрегированное сообщение <code>detection_msgs/DetectionArray</code> , содержащее массив из нескольких <code>Detection</code> . Это уменьшает количество сообщений при большом числе объектов.
3	Добавить в узел подписчика на топик <code>/camera/depth</code> (если есть глубинная камера). При получении детекции извлекать среднюю глубину в области <code>bounding box</code> и публиковать 3D-координаты (x,

	у, z) в системе координат камеры.
4	Вместо публикации всех детекций, определить только одну самую уверенную детекцию за кадр и опубликовать её координаты. Это полезно для задач слежения за одним объектом.
5	Использовать YOLOv8n-seg для сегментации экземпляров. Публиковать не bounding box, а бинарную маску сегментации в виде сообщения sensor_msgs/Image. Ограничить обработку до 5 FPS.
6	Интегрировать детектор с узлом управления (JP7, 11). При обнаружении объекта класса "person" на расстоянии менее порогового (по глубине или по ширине bounding box) публиковать команду остановки в /cmd_vel.
7	Реализовать трекинг объектов: сопоставлять детекции в последовательных кадрах с помощью алгоритма венгерского (Hungarian algorithm) или библиотеки motpy. Публиковать топик /tracking с ID объектов и их историей координат.
8	Создать сервис, который по запросу ('/detect_once') выполняет детекцию на последнем кадре и возвращает список обнаруженных объектов. Полезно для интеграции с конечными автоматами.
9	Перенести инференс на ускоритель Nailo-8 NPU (Raspberry Pi 5). Установить драйверы и библиотеки NailoRT, сконвертировать модель YOLOv8n в формат hef и запустить инференс через NailoNPU. Сравнить FPS и энергопотребление.
10	Ограничить область детекции: обрабатывать только центральную область кадра (например, 320x240). Это снижает нагрузку на CPU, но сохраняет возможность обнаружения в важной зоне перед роботом.

11	Использовать детекцию для подсчёта объектов. Узел должен вести счётчик каждого класса объектов, появляющихся в кадре, и публиковать его в топик /counter (std_msgs/Int64). Счётчик должен инкрементироваться только один раз на объект, пока он находится в поле зрения.
12	Добавить логирование детекций в CSV-файл на SD-карте Raspberry Pi. Каждая строка: временная метка, класс, достоверность, x, y, ширина, высота. Файл ротировать каждые 10 МБ.
13	Реализовать детекцию на гранях (Canny) с последующим поиском контуров, как альтернативу YOLO, для простых случаев (например, мяч на контрастном фоне). Сравнить FPS и точность.
14	Использовать YOLOv8n-pose для детекции ключевых точек (pose estimation). Публиковать топик /pose_detection с массивом координат ключевых точек. Полезно для взаимодействия с человеком.
15	Сделать веб-интерфейс для просмотра детекций на Raspberry Pi. Использовать Flask для трансляции аннотированного видео через веб-сокет. ROS 2 узел должен публиковать аннотированные кадры, а веб-сервер – отдавать их по HTTP.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Что такое YOLOv8 и как его установить?
2. Как выполнить инференс на изображении?
3. Что такое bounding box и confidence?
4. Как фильтровать результаты по порогу confidence?
5. Как ограничить FPS для экономии ресурсов RPi?
6. Как опубликовать координаты объекта в топик ROS 2?
7. Что такое mAP и как оценить качество модели?
8. Как выбрать подходящий вариант YOLOv8 для RPi?

Лабораторная работа 14.

СЛИЯНИЕ ЛИДАРА И КАМЕРЫ: КОМБИНИРОВАННОЕ ИЗБЕГАНИЕ ПРЕПЯТСТВИЙ

Цель и содержание работы: Разработка узла ROS 2, который выполняет слияние данных от лидара (топик /scan) и камеры с детекцией объектов (топик /detection) для принятия решений об управлении мобильным роботом. Реализация логики комбинированного избегания препятствий: при обнаружении препятствия лидаром на расстоянии менее 0.5 метра – полная остановка; при обнаружении камерой объекта заданного класса (например, человек) на расстоянии менее 1.5 метра (оцениваемом по размеру bounding box) – снижение скорости до 0.3 м/с. Публикация команд в топик /cmd_vel. Тестирование в симуляции (Gazebo) с различными сценариями: только лидар, только камера, оба датчика.

Теоретическое обоснование

Лидар и камера обладают комплементарными свойствами. Лидар даёт точные дальности до препятствий в горизонтальной плоскости, устойчив к освещению, но не различает типы объектов. Камера с детектором YOLO (LP13) определяет классы объектов (человек, мяч, конус), но оценка расстояния по монокулярному изображению является косвенной и менее точной. Комбинирование двух сенсоров позволяет:

- Использовать лидар для надёжного обнаружения любых препятствий в ближней зоне (аварийная остановка).
- Использовать камеру для превентивного замедления при появлении специфических объектов (например, людей) на большем расстоянии, где лидар может их не заметить из-за низкой отражающей способности или малой высоты.

В ROS 2 узел-комбинатор подписывается на два топика: `/scan` (`sensor_msgs/LaserScan`) и `/detection` (`detection_msgs/Detection`). В обработчиках сообщений обновляются внутренние переменные состояния: минимальное расстояние по лидару в переднем секторе и наличие/расстояние до целевого объекта по камере. Основной цикл (таймер с частотой 10 Гц) считывает эти переменные и вычисляет результирующую команду скорости по следующим правилам (приоритет остановки выше):

- Если `min_lidar_distance < LIDAR_STOP_DIST` (0.5 м) → скорость = 0.
- Иначе если `camera_object_detected` и `camera_distance < CAMERA_SLOW_DIST` (1.5 м) → скорость = `SLOW_SPEED` (0.3 м/с).
- Иначе → скорость = `NORMAL_SPEED` (0.5 м/с).

Оценка расстояния по камере при монокулярном зрении базируется на предположении о среднем физическом размере объекта (например, ширина плеч человека ~0.5 м). Расстояние вычисляется как $D = (\text{реальный_размер} * \text{фокусное_расстояние_пикс}) / \text{ширина_bounding_box_пикс}$. Фокусное расстояние может быть получено из калибровки камеры или приблизительно оценено. Для упрощения можно использовать эмпирическую зависимость: расстояние ~ $k / \text{ширина_bbox}$, где k подбирается экспериментально.

Такой подход позволяет роботу более интеллектуально реагировать на окружение: не останавливаться резко перед каждым человеком, а плавно снижать скорость, демонстрируя более безопасное и предсказуемое поведение.

Пример

Рассмотрим создание ROS 2 узла на Python с именем `sensor_fusion_avoider`. Предполагается наличие работающих узлов: лидар публикует `/scan` (LP10), камера с детекцией публикует `/detection` (LP13).

Также предполагается наличие пакета `detection_msgs` с сообщением `Detection`.

Узел управляет роботом через `/cmd_vel`.

Создание пакета:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python sensor_fusion --dependencies rclpy
geometry_msgs sensor_msgs detection_msgs
cd sensor_fusion/sensor_fusion
nano fusion_node.py
```

Листинг узла `fusion_node.py`:

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan
from detection_msgs.msg import Detection
from geometry_msgs.msg import Twist
import math

class SensorFusionAvoider(Node):
    def __init__(self):
        super().__init__('sensor_fusion_avoider')
        # Параметры
        self.declare_parameter('lidar_stop_dist', 0.5)      # метры
        self.declare_parameter('camera_slow_dist', 1.5)    # метры
        self.declare_parameter('normal_speed', 0.5)        # м/с
        self.declare_parameter('slow_speed', 0.3)          # м/с
        self.declare_parameter('front_angle_deg', 30.0)     # градусы (сектор
лидара)
        self.declare_parameter('target_class', 'person')    # класс для
замедления
        # Эмпирический коэффициент для оценки расстояния по ширине bbox
        self.declare_parameter('camera_k_factor', 0.8)      # (м * пикс)
предположительно

        self.lidar_stop_dist = self.get_parameter('lidar_stop_dist').value
        self.camera_slow_dist = self.get_parameter('camera_slow_dist').value
        self.normal_speed = self.get_parameter('normal_speed').value
        self.slow_speed = self.get_parameter('slow_speed').value
        self.front_angle_rad =
math.radians(self.get_parameter('front_angle_deg').value)
        self.target_class = self.get_parameter('target_class').value
        self.k_factor = self.get_parameter('camera_k_factor').value

        # Переменные состояния
        self.min_lidar_dist = float('inf')
        self.camera_detected = False
        self.camera_distance = float('inf')
        self.last_lidar_time = self.get_clock().now()
        self.last_camera_time = self.get_clock().now()

        # Подписки
        self.lidar_sub = self.create_subscription(LaserScan, 'scan',
self.lidar_callback, 10)
        self.detection_sub = self.create_subscription(Detection, 'detection',
self.detection_callback, 10)

        # Публикатор cmd_vel
        self.cmd_pub = self.create_publisher(Twist, 'cmd_vel', 10)

        # Таймер основного цикла управления (10 Гц)
        self.timer = self.create_timer(0.1, self.control_loop)
```

```

# Таймауты: если данные не приходят более 1 секунды, считаем датчик
недоступным
self.lidar_timeout = 1.0
self.camera_timeout = 1.0

self.get_logger().info('Sensor Fusion Avider started')

def lidar_callback(self, msg: LaserScan):
    self.last_lidar_time = self.get_clock().now()
    # Определяем индексы для переднего сектора
    angle_min = msg.angle_min
    angle_inc = msg.angle_increment
    start_idx = int((-self.front_angle_rad - angle_min) / angle_inc)
    end_idx = int((self.front_angle_rad - angle_min) / angle_inc)
    start_idx = max(0, start_idx)
    end_idx = min(len(msg.ranges)-1, end_idx)

    min_dist = float('inf')
    for i in range(start_idx, end_idx+1):
        d = msg.ranges[i]
        if math.isnan(d) or math.isinf(d):
            continue
        if d < min_dist:
            min_dist = d
    self.min_lidar_dist = min_dist

def detection_callback(self, msg: Detection):
    self.last_camera_time = self.get_clock().now()
    if msg.object_class == self.target_class and msg.confidence > 0.5:
        # Оценка расстояния по ширине bounding box
        # Предполагаем, что ширина человека ~0.5 метра
        REAL_WIDTH = 0.5 # метра
        # Фокусное расстояние в пикселях (приблизённо, должно быть
откалибровано)
        # Можно вычислить как k_factor * resolution? Упростим: distance =
        (REAL_WIDTH * FX) / width_px
        # Здесь FX = k_factor (подбирается)
        # В примере k_factor = 0.8 - эмпирически, зависит от камеры и
разрешения
        width_px = msg.width
        if width_px > 0:
            # distance = (REAL_WIDTH * FX) / width_px
            # Пусть FX = 800 (типично для HD камеры). Но для простоты
используем k_factor
            distance = (REAL_WIDTH * self.k_factor) / width_px
            self.camera_distance = distance
            self.camera_detected = True
            self.get_logger().debug(f'Detected {msg.object_class} at
distance {distance:.2f}m')
        else:
            self.camera_detected = False
    else:
        self.camera_detected = False

def control_loop(self):
    now = self.get_clock().now()
    # Проверка таймаутов
    if (now - self.last_lidar_time).nanoseconds * 1e-9 > self.lidar_timeout:
        self.min_lidar_dist = float('inf') # нет данных лидара
        self.get_logger().warn('Lidar timeout, ignoring lidar')
    if (now - self.last_camera_time).nanoseconds * 1e-9 >
self.camera_timeout:
        self.camera_detected = False
        self.camera_distance = float('inf')

    cmd = Twist()
    # Приоритет: остановка по лидару

```

```

        if self.min_lidar_dist < self.lidar_stop_dist:
            cmd.linear.x = 0.0
            cmd.angular.z = 0.0
            self.get_logger().info(f'Lidar obstacle: {self.min_lidar_dist:.2f}m,
STOP')
        elif self.camera_detected and self.camera_distance <
self.camera_slow_dist:
            cmd.linear.x = self.slow_speed
            cmd.angular.z = 0.0
            self.get_logger().info(f'Camera detected {self.target_class} at
{self.camera_distance:.2f}m, SLOW to {self.slow_speed}')
        else:
            cmd.linear.x = self.normal_speed
            cmd.angular.z = 0.0
            # Можно добавить небольшой поворот для исследования, но не
обязательно
            self.cmd_pub.publish(cmd)

def main(args=None):
    rclpy.init(args=args)
    node = SensorFusionAvoider()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()

```

Настройка setup.py:

```

entry_points={
    'console_scripts': [
        'fusion_avoider = sensor_fusion.fusion_node:main',
    ],
},

```

Сборка и запуск:

```

cd ~/ros2_ws
colcon build --packages-select sensor_fusion
source install/setup.bash
ros2 run sensor_fusion fusion_avoider --ros-args -p lidar_stop_dist:=0.5 -p
camera_slow_dist:=1.5 -p normal_speed:=0.5 -p slow_speed:=0.3 -p target_class:=person

```

Тестирование в симуляции (Gazebo с TurtleBot3):

- Запустить симуляцию с лидаром и камерой (модель burger с камерой).
- Запустить узел детекции YOLO (LP13), подписанный на /camera/raw.
- Запустить fusion_avoider.
- Разместить в мире препятствие (ящик) – робот остановится по лидару.
- Разместить модель человека – робот замедлится на расстоянии ~1.5 м.

Для проверки расстояния по камере может потребоваться калибровка коэффициента `k_factor`. Рекомендуется поместить человека на известном

расстоянии (1 м) и измерить ширину bbox, затем вычислить $k = (\text{расстояние} * \text{ширина}) / \text{реальная_ширина}$.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Использовать только лидар и камеру, но добавить поворот в сторону свободного пространства, если лидар показывает препятствие спереди, но есть свободное место слева или справа (анализ секторов). Камера при этом только замедляет, но не меняет направление.
2	Ввести два порога для лидара: дальний (0.8 м) – замедление до 0.2 м/с, ближний (0.4 м) – остановка. Для камеры аналогично: обнаружение объекта на >2 м – нормальная скорость, <2 м – замедление, <1 м – остановка. Реализовать комбинированную логику с приоритетом наименьшей скорости.
3	Вместо обнаружения человека, реагировать на обнаружение мяча (класс "sports ball"). При обнаружении мяча на расстоянии менее 1 м робот должен остановиться и опубликовать сообщение "Ball found". Движение возобновляется, если мяч исчез из кадра.

4	Использовать не только класс, но и положение объекта в кадре (координаты x, y центра). Если объект смещён влево или вправо, робот должен поворачивать в сторону объекта (как при слежении). Лидар сохраняет приоритет остановки.
5	Оценивать расстояние по камере не по ширине, а по высоте bounding box (для людей и объектов с известной высотой). Реализовать оба метода и сравнить стабильность оценки при изменении ракурса.
6	Добавить в узел логирование всех сенсорных событий в CSV-файл (время, min_lidar, camera_detected, camera_dist, команда). По окончании работы построить график зависимости скорости от времени.
7	Создать launch-файл, который запускает одновременно лидар, камеру, детекцию YOLO и fusion_avoider с нужными параметрами. Использовать механизм remapping топиков, если имена отличаются.
8	Реализовать узел на C++ с использованием rclcpp. Обработчики сообщений обновляют атомарные переменные, control_loop работает в отдельном потоке. Сравнить задержки с Python-версией.
9	Добавить в узел подписку на топик одометрии (/odom) для определения собственной скорости. Если робот уже движется медленно из-за команды замедления, дополнительно не снижать скорость при появлении человека (гистерезис).
10	Использовать данные глубины с RGB-D камеры (Intel RealSense) вместо монокулярной оценки. Подписаться на топик /depth и для каждого детектированного объекта извлекать среднюю глубину в области bounding box. Сравнить точность.

11	Реализовать динамическое изменение порогов через сервис: сервис /set_params принимает lidar_stop_dist, camera_slow_dist, normal_speed, slow_speed. При вызове параметры обновляются без перезапуска узла.
12	Добавить визуализацию в RViz: публиковать маркеры (visualization_msgs/Marker), показывающие зону обнаружения лидара (полукруг) и позицию обнаруженного камерой объекта. Использовать топик /visualization_marker.
13	Создать модель поведения с конечным автоматом: состояния NORMAL, SLOW_BY_CAMERA, STOP_BY_LIDAR. Переходы с таймаутами. В каждом состоянии публиковать соответствующую команду скорости и дополнительный диагностический топик /state.
14	Интегрировать с ранее разработанной одометрией (ЛР9) и картой. При остановке по лидару робот должен поменять целевую точку (по заглушке) или начать вращение для поиска пути. Реализовать простой планировщик.
15	Провести эксперимент с реальным роботом на Raspberry Pi. Записать видео с камеры и логи лидара, синхронизировать. Оценить, насколько часто камера даёт ложные срабатывания (далёкий человек на 2 метрах оценивается как близкий из-за ошибки масштаба). Предложить улучшения алгоритма.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Как синхронизировать подписки на /scan и /detection в ROS 2?
2. Как объединить логику лидара и камеры?
3. Как реализовать приоритет реакции (лидар vs камера)?
4. Как задать зону детекции по камере (центр кадра)?
5. Как протестировать на разных сценариях?
6. Что такое message_filters и TimeSynchronizer?
7. Как логировать события обнаружения?
8. Каковы ограничения комбинированного подхода?

Лабораторная работа 15.

PID-РЕГУЛЯТОР СКОРОСТИ НА STM32 С ОБРАТНОЙ СВЯЗЬЮ ОТ ЭНКОДЕРОВ

Цель и содержание работы: Реализация на микроконтроллере STM32 замкнутого контура управления скоростью вращения каждого колеса мобильного робота с дифференциальным приводом. В качестве заданной скорости используются значения линейной и угловой скорости, полученные из топика ROS 2 /cmd_vel через micro-ROS (LP9). Обратная связь по скорости формируется на основе данных инкрементальных энкодеров (LP2). Управление осуществляется с помощью трёхкомпонентного ПИД-регулятора (пропорциональная, интегральная, дифференциальная составляющие), который вычисляет требуемую скважность ШИМ для драйвера двигателей (LP7). Выполняется настройка коэффициентов K_p , K_i , K_d , а также сравнительный анализ точности поддержания скорости при разомкнутом и замкнутом контуре управления. Результаты отображаются через отладочную печать по UART.

Теоретическое обоснование

Разомкнутое управление скоростью, при котором заданный ШИМ подаётся на двигатель без учёта реальной скорости, не компенсирует влияние нагрузки, разряда батареи или трения. В результате фактическая скорость колеса может значительно отличаться от желаемой. Для устранения этого недостатка применяется обратная связь по скорости с помощью ПИД-регулятора.

ПИД-регулятор вычисляет управляющий сигнал (ШИМ) на основе ошибки $e(t) = v_{\text{target}} - v_{\text{current}}$, где v_{target} – желаемая скорость (м/с), v_{current} – текущая скорость, измеренная энкодером. Выход регулятора формируется как сумма трёх слагаемых:

$$u(t) = K_p * e(t) + K_i * \text{Integral}(e(t) dt) + K_d * de(t)/dt.$$

- Пропорциональная составляющая (K_p) даёт основной вклад в управление, пропорциональный текущей ошибке. При увеличении K_p уменьшается время реакции, но возрастает риск перерегулирования и колебаний.

- Интегральная составляющая (K_i) накапливает ошибку во времени и позволяет устранить статическую ошибку (например, при постоянном моменте нагрузки). Интеграл необходимо ограничивать (anti-windup), чтобы избежать насыщения регулятора.

- Дифференциальная составляющая (K_d) реагирует на скорость изменения ошибки, улучшая демпфирование и снижая перерегулирование.

Для цифровой реализации на STM32 удобно использовать дискретные формы ПИД-регулятора с периодом дискретизации T (например, 10 мс, частота 100 Гц). Распространена скоростная форма (incremental form), где вычисляется приращение управляющего сигнала, что упрощает защиту от насыщения интегратора. Однако в учебных целях можно применять позиционную форму с ограничением интегральной суммы.

В работе для каждого колеса вычисляется своя целевая скорость на основе команд v_linear и $v_angular$, полученных из `/cmd_vel`, и кинематики дифференциального привода:

$$v_left_target = v_linear - (v_angular * B/2)$$

$$v_right_target = v_linear + (v_angular * B/2)$$

где B – ширина колеи.

Текущие линейные скорости колёс (м/с) вычисляются по приращениям энкодеров за интервал dt :

$$v_current = (\text{delta_pulses} / \text{PPR}) * (\pi * D) / dt,$$

где PPR – количество импульсов энкодера на оборот, D – диаметр колеса.

ПИД-регулятор выполняется в прерывании таймера (100 Гц). Для каждого колеса:

```
error = target_speed - current_speed;
```

```
integral += error * dt (с насыщением);
```

```
derivative = (error - prev_error) / dt;
```

```
output_pwm = Kp*error + Ki*integral + Kd*derivative;
```

Output_pwm ограничивается диапазоном [-PWM_MAX, PWM_MAX].

Знак определяет направление вращения (для реверсивного драйвера).

После вычисления выходного ШИМ для обоих колёс обновляются регистры сравнения таймеров. Для отладки целесообразно отправлять по UART значения заданной и текущей скоростей, а также выходной ШИМ.

Коэффициенты Kp, Ki, Kd подбираются экспериментально: сначала устанавливают Ki=Kd=0 и увеличивают Kp до появления колебаний, затем вводят Ki для устранения статической ошибки, после чего добавляют Kd для демпфирования.

Пример

Рассмотрим реализацию PID-регулятора скорости на STM32F103 (или F4) для двух колёс. Исходные данные: PPR = 400 (импульсов на оборот), D = 0.065 м, B = 0.26 м, PWM_MAX = 1000 (таймер с периодом 1000). Задание скорости поступает через micro-ROS подписчика на топик /cmd_vel. Частота регулятора – 100 Гц (таймер TIM6). Энкодеры подключены к TIM2 и TIM3 в режиме Encoder Mode (LP2). Драйвер двигателей – L298N с ШИМ на TIM3_CH1 (левое) и TIM4_CH1 (правое). Скорость UART для логирования – 115200.

Глобальные переменные и структуры:

```
#define PWM_MAX 1000
#define DT 0.01f // 100 Гц
#define WHEEL_DIAM 0.065f
#define PPR 400
#define WHEEL_BASE 0.26f
#define PI 3.14159f
#define M_PER_PULSE (PI * WHEEL_DIAM / PPR)

typedef struct {
    float Kp, Ki, Kd;
    float target_speed;
    float current_speed;
    float integral;
    float prev_error;
    int16_t pwm_out;
} PID_Handle;

PID_Handle pid_left, pid_right;
volatile float v_linear_cmd = 0.0f, v_angular_cmd = 0.0f;
volatile int16_t left_pulses, right_pulses, last_left, last_right;
volatile uint8_t new_cmd = 0;
```

Функция инициализации PID:

```
void PID_Init(PID_Handle *pid, float Kp, float Ki, float Kd) {
    pid->Kp = Kp;
    pid->Ki = Ki;
    pid->Kd = Kd;
    pid->target_speed = 0.0f;
    pid->current_speed = 0.0f;
    pid->integral = 0.0f;
    pid->prev_error = 0.0f;
    pid->pwm_out = 0;
}
```

Функция вычисления PID для одного колеса (вызывается каждые DT):

```
void PID_Update(PID_Handle *pid, float dt) {
    float error = pid->target_speed - pid->current_speed;
    // Интегральная составляющая с ограничением anti-windup
    pid->integral += error * dt;
    if (pid->integral > PWM_MAX / pid->Ki) pid->integral = PWM_MAX / pid->Ki;
    if (pid->integral < -PWM_MAX / pid->Ki) pid->integral = -PWM_MAX / pid->Ki;
    float derivative = (error - pid->prev_error) / dt;
    float output = pid->Kp * error + pid->Ki * pid->integral + pid->Kd *
derivative;
    // Ограничение выходного ШИМ
    if (output > PWM_MAX) output = PWM_MAX;
    if (output < -PWM_MAX) output = -PWM_MAX;
    pid->pwm_out = (int16_t)output;
    pid->prev_error = error;
}
```

Функция обновления скоростей колёс по энкодерам (вызывается в таймере регулятора):

```
void UpdateWheelSpeeds(void) {
    int16_t delta_left = left_pulses - last_left;
    int16_t delta_right = right_pulses - last_right;
    last_left = left_pulses;
    last_right = right_pulses;
    // Коррекция переполнения 16-битного счётчика
    if (delta_left > 32767) delta_left -= 65536;
    if (delta_left < -32768) delta_left += 65536;
    if (delta_right > 32767) delta_right -= 65536;
    if (delta_right < -32768) delta_right += 65536;

    float left_dist = delta_left * M_PER_PULSE;
    float right_dist = delta_right * M_PER_PULSE;
    pid_left.current_speed = left_dist / DT;
    pid_right.current_speed = right_dist / DT;
}
```

Функция установки заданий для PID из команд `v_linear`, `v_angular`:

```
void SetTargetSpeeds(float v_lin, float v_ang) {
    float left_target = v_lin - (v_ang * WHEEL_BASE / 2.0f);
    float right_target = v_lin + (v_ang * WHEEL_BASE / 2.0f);
    pid_left.target_speed = left_target;
    pid_right.target_speed = right_target;
}
```

Прерывание таймера TIM6 (100 Гц):

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM6) {
        // Получение текущих значений счётчиков энкодеров
        left_pulses = __HAL_TIM_GET_COUNTER(&htim2);
        right_pulses = __HAL_TIM_GET_COUNTER(&htim3);
        UpdateWheelSpeeds();
        PID_Update(&pid_left, DT);
        PID_Update(&pid_right, DT);
        // Применение ШИМ (с учётом направления)
        set_motor_pwm(pid_left.pwm_out, pid_right.pwm_out);
        // Логирование (раз в секунду можно отсылать по UART)
        static uint32_t log_cnt = 0;
        if (++log_cnt >= 100) { // 100 * 10 мс = 1 сек
            log_cnt = 0;
            char buf[128];
            sprintf(buf, "L_t=%.2f L_a=%.2f L_p=%d | R_t=%.2f R_a=%.2f R_p=%d\r\n",
                    pid_left.target_speed, pid_left.current_speed, pid_left.pwm_out,
                    pid_right.target_speed, pid_right.current_speed,
                    pid_right.pwm_out);
            HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 10);
        }
    }
}
```

Интеграция micro-ROS: создаётся подписчик на топик /cmd_vel. В callback-функции сохраняются v_linear_cmd и v_angular_cmd, устанавливается флаг new_cmd. В основном цикле по флагу вызывается SetTargetSpeeds.

Настройка PID-коэффициентов (подбираются экспериментально). Начальные значения: $K_p = 50$, $K_i = 10$, $K_d = 5$. После тестов на реальном роботе корректируются.

Тестирование: при подаче постоянной скорости 0.3 м/с робот должен поддерживать её с минимальной ошибкой. При создании нагрузки (например, притормаживании колеса рукой) регулятор должен увеличивать ШИМ для компенсации.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Реализовать PID-регулятор только для левого колеса, правое управляется разомкнуто. Сравнить точность поддержания скорости при разгоне и торможении. Выводить оба значения скорости по UART.
2	Вычислять интегральную составляющую только при условии,

	что выходной ШИМ не достиг насыщения (anti-windup через условие). Сравнить поведение регулятора с простым ограничением интегратора.
3	Добавить в регулятор каскадную структуру: внутренний контур – ШИМ → скорость, внешний контур – скорость → положение. Реализовать П-регулятор положения с ограничением скорости. Протестировать перемещение на заданное количество оборотов колеса.
4	Реализовать PID-регулятор в скоростной (инкрементальной) форме:
5	Вместо фиксированных K_p , K_i , K_d сделать их настраиваемыми через UART. Написать простой протокол: например, "p 1.5" для установки K_p . Выводить текущие коэффициенты по запросу "g".
6	Использовать ПИД-регулятор для стабилизации угловой скорости (омега). Вычислять омегу по разности скоростей колёс: $\omega = (v_{\text{right}} - v_{\text{left}}) / B$. Целевая омега поступает из cmd_vel. Регулятор напрямую вычисляет разность ШИМ. Сравнить с управлением через индивидуальные PID колёс.
7	Добавить фильтр низких частот на измеренную скорость (простое экспоненциальное сглаживание) перед подачей в PID. Оценить влияние на шум и запаздывание.
8	Реализовать ограничение ускорения для задания скорости: желаемая скорость изменяется не скачком, а с постоянным ускорением (0.5 м/с^2). Это снижает рывки при старте и остановке.
9	Логировать работу регулятора в память STM32 (массив структур за 10 секунд), а затем по команде сбрасывать по UART для построения графиков в Python.

10	Сравнить разомкнутое и замкнутое управление: подать ступенчатое задание 0.3 м/с и записать переходный процесс. Измерить время установления, перерегулирование, статическую ошибку. Сделать вывод.
11	Адаптивный PID: при большой ошибке (например, >0.2 м/с) временно увеличивать K_p , при малой ошибке уменьшать K_i . Реализовать переключение коэффициентов.
12	Добавить в контур управления компенсацию статического трения: если заданная скорость не равна нулю, а колесо не вращается (ошибка нарастает), то принудительно добавить минимальный ШИМ (20 единиц) для страгивания.
13	Реализовать PID-регулятор с использованием аппаратного таймера для высокочастотного прерывания (1 кГц) и усреднением скорости за 10 мс (передискретизация). Оценить влияние на качество регулирования.
14	Интегрировать ПИД-регулятор скорости с одометрией (ЛРЗ) и micro-ROS: получать целевую скорость через топик /cmd_vel, а публиковать реальную скорость колёс в отдельный топик для мониторинга.
15	Создать модель разомкнутой системы (ШИМ $\rightarrow$ скорость) путём эксперимента: подать ступеньки ШИМ и записать установившуюся скорость. Построить зависимость $v(PWM)$ и использовать её для линейной компенсации (feedforward) в дополнение к PID. Сравнить с чистым PID.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики,

ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Из каких составляющих состоит ПИД-регулятор?
2. Как организовать цикл ПИД в прерывании таймера?
3. Что такое интегральное насыщение (anti-windup)?
4. Как получить текущую скорость с энкодеров?
5. Как преобразовать выход ПИД в ШИМ?
6. Как подобрать коэффициенты K_p , K_i , K_d ?
7. Почему замкнутый контур лучше разомкнутого?
8. Как логировать заданную и текущую скорость?

Лабораторная работа 16.

ПОЛНЫЙ СТЕК НАВИГАЦИИ: NAV2 + MICRO-ROS

Цель и содержание работы: Интеграция микроконтроллера STM32, работающего под управлением micro-ROS, с полным стеком навигации ROS 2 (пакет Navigation2, также известный как nav2). Реализация автономной навигации мобильного робота на основе предварительно построенной карты окружающей среды. Задействование глобального планировщика (алгоритм A*) для построения пути от текущего положения до заданной цели и локального планировщика (Dynamic Window Approach, DWA) для объезда динамических препятствий. Установка навигационных целей через графический интерфейс RViz. Мониторинг траектории движения и оценка качества следования.

Теоретическое обоснование

Навигационный стек Navigation2 (nav2) является стандартным решением для автономной навигации мобильных роботов в среде ROS 2. Он включает в себя следующие основные компоненты:

- Карта (map): представление окружающей среды в виде сетки занятости (occupancy grid), где каждая ячейка содержит вероятность наличия препятствия. Карта может быть загружена из файла или построена в процессе работы с помощью алгоритмов SLAM (например, slam_toolbox).
- Глобальный планировщик (Global Planner): алгоритм, который находит оптимальный путь от текущего положения робота до заданной цели с учётом статических препятствий на карте. Наиболее распространённый алгоритм — A* (A-star). Результатом является последовательность глобальных узловых точек (global path).
- Локальный планировщик (Local Planner): алгоритм, который принимает глобальный путь и текущие данные сенсоров (лидар, одометрия) и

вычисляет команды скорости (`cmd_vel`) в реальном времени, объезжая динамические и ранее не отображённые препятствия. В `nav2` по умолчанию используется Regulated Pure Pursuit (RPP) или DWA (Dynamic Window Approach).

- Преобразования координат (`tf`): система, которая поддерживает дерево преобразований между различными системами координат, например, `map` -> `odom` -> `base_link` -> `laser_link`. Это необходимо для согласованной работы планировщиков и сенсоров.

- Поведенческие деревья (Behavior Trees): управляют переходами между состояниями навигации (инициализация, планирование, следование по пути, восстановление при застревании).

`micro-ROS` на `STM32` выполняет роль низкоуровневого драйвера:

- Публикует топик `/odom` (`nav_msgs/Odometry`) на основе данных энкодеров и, возможно, `IMU` (см. ЛР9).

- Публикует преобразование `tf odom` → `base_link`.

- Подписывается на топик `/cmd_vel` (`geometry_msgs/Twist`), генерируемый локальным планировщиком, и преобразует полученные линейную и угловую скорости в ШИМ для моторов (см. ЛР7 и ЛР15). Желательно также реализовать замкнутый контур PID-регулятора скорости на `STM32` для точного исполнения команд.

`Raspberry Pi` (или другой компьютер) запускает полный стек `nav2`, подписывается на `/odom` и `tf`, а также на топики сенсоров (например, `/scan` от лидара) и публикует `/cmd_vel`. Для визуализации используется `RViz`, где оператор может задать цель нажатием кнопки “2D Nav Goal”.

Для успешной навигации необходимы:

- Точная одометрия (малая ошибка накопления).

- Стабильная связь между `STM32` и `Raspberry Pi` через `micro-ROS` агент (`UART`, `Ethernet` или `Wi-Fi`).

- Предварительно построенная карта или запуск SLAM в режиме реального времени.
- Корректно настроенные параметры планировщиков (максимальные скорости, ускорения, радиус робота и т.п.).

Пример

Рассмотрим развёртывание полного стека навигации для робота с STM32 (микроконтроллер) и Raspberry Pi 4 (хост ROS 2). Предполагается, что выполнены предыдущие лабораторные работы:

- ЛР7: приём `/cmd_vel` и управление моторами.
- ЛР9: публикация `/odom` через micro-ROS.
- ЛР10: подключение лидара и публикация `/scan`.
- ЛР14 или ЛР15: (опционально) PID-регулятор скорости для точного исполнения.

Карта помещения построена заранее с помощью `slam_toolbox` (например, при ручном управлении) и сохранена в файлы `map.pgm` и `map.yaml`.

Конфигурация на Raspberry Pi.

Установка пакетов nav2:

```
sudo apt install ros-humble-navigation2 ros-humble-nav2-bringup ros-humble-slam-toolbox
```

Создание launch-файла для запуска навигации с micro-ROS агентом:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python my_nav_bringup
cd my_nav_bringup/my_nav_bringup
nano nav_bringup.launch.py
```

Содержимое launch-файла:

```

from launch import LaunchDescription
from launch.actions import ExecuteProcess, DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node

def generate_launch_description():
    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    map_yaml = LaunchConfiguration('map', default='/home/pi/map/map.yaml')
    params_file = LaunchConfiguration('params',
default='/home/pi/nav2_params.yaml')

    return LaunchDescription([
        DeclareLaunchArgument('use_sim_time', default_value='false'),
        DeclareLaunchArgument('map', default_value='/home/pi/map/map.yaml'),
        DeclareLaunchArgument('params',
default_value='/home/pi/nav2_params.yaml'),

        # micro-ROS агент (UART)
        ExecuteProcess(
            cmd=['ros2', 'run', 'micro_ros_agent', 'micro_ros_agent',
                'serial', '--dev', '/dev/ttyAMA0', '-b', '115200'],
            output='screen'
        ),

        # Преобразование координат (статическое base_link <-> laser)
        Node(
            package='tf2_ros',
            executable='static_transform_publisher',
            arguments=['0', '0', '0.1', '0', '0', '0', 'base_link',
'laser_frame']
        ),

        # Запуск навигационного стека
        Node(
            package='nav2_bringup',
            executable='nav2_bringup',
            name='nav2_bringup',
            output='screen',
            parameters=[map_yaml, params_file],
            arguments=['--use_sim_time', use_sim_time]
        ),

        # RViz для визуализации
        Node(
            package='rviz2',
            executable='rviz2',
            name='rviz2',
            arguments=['-d', '/home/pi/nav2_config.rviz'],
            parameters=[{'use_sim_time': use_sim_time}]
        )
    ])

```

Создание файла параметров nav2_params.yaml (пример фрагмента):

```

amcl:
  ros__parameters:
    use_sim_time: False
    odom_frame_id: odom
    base_frame_id: base_link
    global_frame_id: map

```

```

bt_navigator:
  ros__parameters:
    use_sim_time: False

controller_server:
  ros__parameters:
    use_sim_time: False
    controller_plugins: ["FollowPath"]
  FollowPath:
    plugin:
      "nav2_regulated_pursuit_controller/RegulatedPurePursuitController"
    desired_linear_vel: 0.5
    max_linear_accel: 0.8
    desired_angular_vel: 0.8

planner_server:
  ros__parameters:
    use_sim_time: False
    planner_plugins: ["GridBased"]
  GridBased:
    plugin: "nav2_navfn_planner/NavfnPlanner"

recoveries_server:
  ros__parameters:
    use_sim_time: False

```

Настройка параметров робота: необходимо указать радиус робота (footprint) и максимальные скорости. Корректировка параметров AMCL (адаптивный Монте-Карло локализации) для слияния одометрии и лидара.

Запуск:

```

cd ~/ros2_ws
colcon build --packages-select my_nav_bringup
source install/setup.bash
ros2 launch my_nav_bringup nav_bringup.launch.py map:=/home/pi/my_map.yaml

```

В RViz после открытия необходимо:

- Установить фиксированный фрейм "map".
- Добавить отображение карты (Map), глобального пути (Global Path), локального пути (Local Path), облака лазерных точек (LaserScan).
- Нажать кнопку "2D Nav Goal" и указать цель на карте.

Робот должен построить глобальный путь и начать движение, объезжая препятствия, которые видны лидаром. При отсутствии карты можно запустить одновременное построение карты (SLAM) с помощью slam_toolbox, заменив nav2_bringup на slam_toolbox.

Проверка работы: при постановке цели робот вычисляет путь, публикует /cmd_vel, STM32 отрабатывает команды. Топик /odom должен быть доступен на Raspberry Pi.

Возможные проблемы:

- Неправильная настройка tf (odom → base_link). Убедиться, что STM32 публикует tf.
- Отсутствие данных /scan – проверить лидар.
- Высокая задержка по UART – уменьшить частоту публикации одометрии до 20 Гц.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Использовать локальный планировщик DWA (dynamic window approach) вместо Regulated Pure Pursuit. Настроить параметры: максимальная линейная скорость 0.4 м/с, максимальная угловая скорость 1.0 рад/с. Продемонстрировать объезд внезапно появившегося препятствия.
2	Загрузить готовую карту кабинета (файл .pgm). Установить цель в противоположном углу комнаты. Измерить время достижения

	цели и среднюю скорость. Сравнить с движением по прямой без препятствий.
3	Вместо фиксированной карты запустить slam_toolbox в режиме онлайн-построения карты и одновременно навигации. Убедиться, что робот может исследовать новую область и достигать цели.
4	Использовать реальный лидар RPLIDAR A1 и STM32 с micro-ROS. Провести навигацию в коридоре длиной не менее 5 метров с поворотом. Записать трек (x,y) и сравнить с идеальным маршрутом.
5	Настроить поведенческое дерево (Behavior Tree) для восстановления при застревании. Добавить в файл параметров recovery: spin и backup. Убедиться, что робот при блокировке пути начинает вращаться на месте.
6	Увеличить радиус робота в параметрах (footprint) с 0.2 м до 0.4 м. Продемонстрировать, как это влияет на проходимость узких проходов. Сделать вывод.
7	Интегрировать данные с камеры глубины (Intel RealSense) как дополнительный источник препятствий (топик /depth). Настроить фильтр объединения облаков точек (pointcloud) и использовать в локальном планировщике.
8	Реализовать на STM32 отправку топика /battery_status (пользовательское сообщение) и подписку на него в узле recovery для отправки робота на зарядную станцию при низком напряжении.
9	Вместо UART использовать Wi-Fi соединение между micro-ROS агентом и STM32 (ESP8266 или ESP32). Настроить UDP транспорт. Проверить задержку и надёжность по сравнению с UART.

10	Заменить глобальный планировщик Navfn на Smac Planner (гибридный A*). Настроить параметры информирования о пространстве. Сравнить качество получаемых путей (гладкость, длина).
11	Добавить в RViz отображение маркеров цели и текущего состояния робота (текст с режимом навигации). Использовать топик /diagnostics для вывода предупреждений.
12	Протестировать навигацию с выключенным PID-регулятором скорости на STM32 (чисто разомкнутое управление). Записать фактические скорости из одометрии и сравнить с заданными. Сделать вывод о необходимости замкнутого контура.
13	Создать сервис для смены карты во время работы (например, переключение между картами этажа). Реализовать узел, который по команде загружает новый файл map.yaml и обновляет глобальный планировщик.
14	Внедрить в навигационный стек задачу «следовать за человеком» (person following). Использовать детекцию YOLO (JP13) для публикации позиции человека в системе координат камеры, преобразовать в cmd_vel и отключить глобальное планирование.
15	Провести эксперимент с различными частотами публикации одометрии (10, 20, 50 Гц) и оценить влияние на качество локализации (дрейф) при движении по квадрату 2x2 метра. Результаты оформить в виде графика.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Как настроить стек Nav2 для работы с micro-ROS?
2. Какие топики принимает move_base?
3. Что такое global_costmap и local_costmap?
4. Как задать цель через RViz (2D Nav Goal)?
5. Как загрузить карту через nav2_map_server?
6. Что такое lifecycle_manager?
7. Как настроить планировщики через YAML?
8. Как отладить навигацию через ros2 topic echo?

Лабораторная работа 17.

SLAM: КАРТОГРАФИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ЛИДАРА

Цель и содержание работы: Построение карты окружающего пространства (occupancy grid) с использованием лидара и пакета `slam_toolbox` в среде ROS 2 на одноплатном компьютере Raspberry Pi. Управление роботом вручную для обхода помещения, визуализация формируемой карты в реальном времени с помощью RViz. Сохранение построенной карты для последующего использования в задачах автономной навигации (например, LP16). Проверка загрузки сохранённой карты и возможности навигации по ней.

Теоретическое обоснование

SLAM (Simultaneous Localization and Mapping) – это алгоритмическая задача одновременного построения карты неизвестного окружения и оценки текущего положения робота внутри этой карты. В отличие от навигации по готовой карте (LP16), SLAM не требует предварительной карты и позволяет роботу исследовать пространство.

Пакет `slam_toolbox` предоставляет реализацию алгоритмов SLAM на основе графов (graph-based SLAM), оптимизированных для 2D лидаров. Основные режимы работы:

- Асинхронный (async) – для медленного движения и построения карты в фоновом режиме, хорошо подходит для ручного управления.
- Синхронный (sync) – для более точного, но более требовательного к ресурсам построения.
- Локализация (localization) – для работы с готовой картой без её обновления.

Входные данные: топик `/scan` (`sensor_msgs/LaserScan`) с данными лидара, топики `/odom` (навигационная одометрия) и преобразования `tf` (`odom`

→ `base_link`). Выходные данные: топик `/map` (`nav_msgs/OccupancyGrid`) – строящаяся карта, а также уточнённые `tf`-преобразования `map` → `odom`.

Алгоритм внутри `slam_toolbox` использует:

- Граф поз: узлы соответствуют позам робота в разные моменты времени, рёбра – относительные перемещения (из одометрии или `scan matching`).

- Оптимизацию графа (например, методом Гаусса-Ньютона) для минимизации ошибок согласования сканов.

- Вероятностную сетку занятости (`occupancy grid`), где каждая ячейка хранит логарифмическую вероятность наличия препятствия.

Для построения качественной карты необходимо:

- Наличие стабильной одометрии (хотя бы приблизительной, так как `slam_toolbox` может её корректировать).

- Медленное и плавное движение робота, чтобы сканы не были сильно искажены.

- Обход помещения так, чтобы лидар «видел» все стены и углы.

- Хорошее освещение не требуется, так как используется лидар, а не камера.

После завершения построения карта сохраняется в виде двух файлов: `.pgm` (изображение в градациях серого) и `.yaml` (метаинформация – разрешение, начало координат и т.д.). Сохранённая карта может быть загружена в навигационный стек (LP16) для автономного перемещения.

Пример

Рассмотрим процесс построения карты на Raspberry Pi 4 с подключённым лидаром RPLIDAR A1 (LP10) и роботом, управляемым с клавиатуры (`teleop_twist_keyboard`). Одометрия публикуется через `micro-ROS` (LP9) или, в случае отсутствия реального робота, может быть заменена симуляцией.

Предварительные требования:

- Установлен пакет slam_toolbox: `sudo apt install ros-humble-slam-toolbox`
- Установлен пакет teleop_twist_keyboard: `sudo apt install ros-humble-teleop-twist-keyboard`
- Лидар публикует топик `/scan`, одометрия – `/odom`, и работают преобразования `odom->base_link`.

Подготовка launch-файла для запуска slam_toolbox (асинхронный режим). Создадим пакет my_slam и файл slam_async.launch.py:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python my_slam
cd my_slam/my_slam
nano slam_async.launch.py
```

Содержимое:

```
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import ExecuteProcess

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='slam_toolbox',
            executable='async_slam_toolbox_node',
            name='slam_toolbox',
            parameters=[{'use_sim_time': False}],
            output='screen'
        ),
        Node(
            package='rviz2',
            executable='rviz2',
            name='rviz2',
            arguments=['-d', '/home/pi/slam_rviz_config.rviz']
        ),
        # teleop (опционально, можно запустить отдельно)
        ExecuteProcess(
            cmd=['ros2', 'run', 'teleop_twist_keyboard',
'teleop_twist_keyboard'],
            output='screen'
        )
    ])
```

Настройка конфигурации RViz: открыть вкладку Add -> By topic -> /map -> Map, а также добавить LaserScan, Odometry, RobotModel. Сохранить конфигурацию как slam_rviz_config.rviz.

Запуск построения карты:

```
cd ~/ros2_ws
colcon build --packages-select my_slam
```

```
source install/setup.bash
ros2 launch my_slam slam_async.launch.py
```

Управление роботом: в терминале с `teleop_twist_keyboard` используйте клавиши (`i` – вперед, `j` – поворот налево, `l` – направо, `,` – назад). Медленно объезжайте помещение, чтобы лидар сканировал стены, углы, проходы.

Визуализация: в RViz отображается серая карта, которая постепенно расширяется и уточняется. Белые области – свободное пространство, чёрные – стены, серые – неизвестная область.

Сохранение карты:

```
ros2 run nav2_map_server map_saver_cli -f ~/my_map
```

Эта команда создаст файлы `my_map.pgm` и `my_map.yaml` в домашней директории.

Проверка загрузки карты и навигации (без движения, только визуализация):

```
ros2 run nav2_map_server map_server --ros-args -p
yaml_filename:=/home/pi/my_map.yaml
```

В другом терминале: `ros2 run rviz2 rviz2`, добавить отображение карты (By topic -> /map). Карта должна отобразиться.

Для полноценной навигации с загруженной картой можно использовать `nav2_bringup` (LP16) с параметром `map:=/home/pi/my_map.yaml`.

Возможные ошибки:

- Нет данных одометрии /odom: `slam_toolbox` может работать без одометрии (только `scan matching`), но качество карты ухудшится. Для этого используется параметр `odom_frame: odom` (если есть) или установка `use_odom: false`.

- Дрейф карты: при слишком быстром повороте. Рекомендуется двигаться медленно и плавно.

- Не закрываются петли (loop closure): после обхода помещения необходимо вернуться в уже пройденное место, чтобы алгоритм оптимизировал карту.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Использовать синхронный режим slam_toolbox (sync_slam_toolbox_node) вместо асинхронного. Сравнить время обработки каждого скана и качество карты при одинаковом маршруте.
2	Построить карту лабораторного помещения площадью не менее 20 квадратных метров. При обходе сделать не менее двух пересекающихся траекторий (например, «восьмёрку»), чтобы проверить закрытие петель. Сохранить карту.
3	В процессе построения карты использовать не teleop, а простой автономный обход: робот движется вперёд до препятствия, поворачивает случайным образом. Написать скрипт на Python для такого поведения и построить карту.
4	После построения карты запустить локализацию (localization)

	mode) в slam_toolbox, не обновляя карту. Убедиться, что робот правильно локализуется на карте при движении.
5	Изменить параметры slam_toolbox: resolution (разрешение карты) с 0.05 м по умолчанию на 0.03 м. Построить карту того же помещения и сравнить размер файла .pgm и чёткость стен.
6	Подключить лидар YDLIDAR X4 и настроить slam_toolbox для работы с ним. Сравнить время построения карты и качество (наличие артефактов) по сравнению с RPLIDAR A1.
7	Запустить slam_toolbox без одометрии (только лидар). Для этого в launch-файле задать параметр use_odom: false. Построить карту комнаты и оценить искажения масштаба (например, длину стены, измеренную на карте, с реальной длиной).
8	Создать launch-файл, который одновременно запускает драйвер лидара, slam_toolbox, телеоперацию и RViz. Добавить параметры командной строки для выбора режима (async/sync) и пути сохранения карты.
9	Построить карту коридора с несколькими дверьми. Остановить робота перед каждой дверью и открыть/закрыть дверь, чтобы лидар увидел изменение. Убедиться, что карта обновляется (если slam_toolbox работает в режиме с обновлением).
10	Использовать карту, построенную в варианте 2, для навигации (LP16). Подать цель в соседнюю комнату и запустить move_base. Записать траекторию и оценить, насколько точно робот следует по коридору.
11	Вместо реального робота использовать симуляцию TurtleBot3 в Gazebo с лидаром. Построить карту виртуального мира (world) с лабиринтом. Сравнить процесс построения с реальным.
12	Изменить параметр minimum_time_interval_ms (время между

	обработкой сканов) с 200 мс на 500 мс. Построить карту и оценить влияние на сглаживание и детализацию.
13	Реализовать автоматическое сохранение карты при выходе из узла <code>slam_toolbox</code> (сигнал SIGINT). Добавить обработчик сигналов в <code>launch</code> -файле или написать обёрточный скрипт.
14	Построить карту помещения, в котором есть стеклянные поверхности (витрины, окна). Проанализировать, как лидар видит (или не видит) стекло, и как это отражается на карте. Предложить метод постобработки.
15	Сравнить два алгоритма SLAM: <code>slam_toolbox</code> и <code>cartographer</code> (Google Cartographer). Установить <code>cartographer_ros</code> для ROS 2, запустить на том же наборе данных (реальном или записанном мешке), сравнить качество карты и потребление ресурсов CPU.

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа `.docx` или `.pdf`.

Вопросы для защиты работы

1. Что такое SLAM и как он строит карту?
2. Как запустить `slam_toolbox`?
3. Как управлять роботом в режиме SLAM (`teleop`)?
4. Как сохранить карту через `map_saver_cli`?
5. Как загрузить сохранённую карту?
6. Каковы требования к данным лидара для SLAM?
7. Что такое async SLAM?

8. Как убедиться в качестве построенной карты?

Лабораторная работа 18.

ИТОГОВЫЙ ПРОЕКТ: ПОЛНОСТЬЮ АВТОНОМНЫЙ РОБОТ

Цель и содержание работы: Разработка и интеграция всех ранее изученных компонентов в законченную автономную робототехническую систему на базе микроконтроллера STM32 и одноплатного компьютера Raspberry Pi. Реализация полного навигационного стека: одометрия и управление моторами на STM32 (энкодеры, PID, micro-ROS), восприятие окружающей среды на Raspberry Pi (лидар, камера с детекцией YOLO), построение карты (SLAM) или её загрузка, планирование пути и автономное перемещение к заданной цели с объездом статических и динамических препятствий. Итоговая демонстрация работы системы в реальном или симулированном окружении. Защита проекта с предоставлением документации и демонстрационного видео.

Теоретическое обоснование

Итоговый проект обобщает навыки, полученные в предыдущих работах (ЛР1–ЛР17). Полноценный автономный робот представляет собой распределённую систему реального времени, где:

- Уровень низкоуровневого управления (STM32): выполняет циклический сбор данных с энкодеров (ЛР2), вычисление одометрии (ЛР3), фильтрацию углов (комплементарный фильтр или EKF, ЛР4, ЛР5), замкнутое управление скоростью каждого колеса (ПИД-регулятор, ЛР15) по командам /cmd_vel, полученным через micro-ROS (ЛР9), и публикацию топика /odom и преобразований tf (ЛР9). Также может обрабатывать команды аварийной остановки.

- Уровень высокоуровневого управления и восприятия (Raspberry Pi): подключает лидар (ЛР10) и публикует топик /scan, обрабатывает изображение с камеры для детекции объектов (ЛР12, ЛР13). Запускает SLAM

(JP17) для построения карты или загружает готовую карту. Использует навигационный стек Navigation2 (move_base, JP16) для глобального планирования (A^*) и локального объезда препятствий (DWA). Взаимодействует с пользователем через RViz для задания целей.

- Связь: micro-ROS агент на Raspberry Pi (JP8) соединяется с STM32 по UART, USB или Wi-Fi, обеспечивая двусторонний обмен сообщениями ($\text{cmd_vel} \rightarrow \text{STM32}$, $\text{odom} \leftarrow \text{STM32}$).

- Сенсорная интеграция: для более надёжного избегания препятствий могут объединяться данные лидара и камеры (JP14). Например, лидар обеспечивает точные дальности, а камера позволяет классифицировать объекты (человек, конус) и адаптировать поведение (замедление перед человеком).

- Функциональная безопасность: должен быть реализован таймаут команд (если нет $\text{cmd_vel} > 500$ мс – остановка), аппаратное прерывание от энкодеров для контроля застревания, а также реакция на экстренную остановку (например, по кнопке).

- Тестирование: система должна демонстрировать способность достичь заданной цели (расстояние 5–10 метров) в помещении с известными статическими препятствиями (столы, коробки) и динамическим препятствием (человек, внезапно переходящий дорогу). Робот должен остановиться или объехать динамическое препятствие в зависимости от сценария.

- Документация: требуется предоставить файл README с описанием аппаратной архитектуры, схемой подключения, инструкцией по установке ПО (зависимости, сборка), краткими пояснениями по ключевым узлам (PID, одометрия, навигация), а также ссылку на видеодемонстрацию (YouTube или локальный файл).

Пример

Рассмотрим типовую реализацию итогового проекта для робота на колёсной базе с дифференциальным приводом. Аппаратная платформа: STM32F103 (Blue Pill), два энкодерных мотора, драйвер L298N, MPU6050. Raspberry Pi 4B, лидар RPLIDAR A1, USB-камера. Связь: UART между STM32 и Raspberry Pi (GPIO 14/15). Питание: две литиевые батареи 18650 через стабилизатор на 5В и 3.3В.

Общая архитектура программного обеспечения (ROS 2 Humble, Ubuntu 22.04 на Raspberry Pi).

1. Схема узлов ROS 2:

```
/scan (sensor_msgs/LaserScan) - лидар  
/camera/raw (sensor_msgs/Image) - камера  
/detection (detection_msgs/Detection) - детектор YOLO  
/odom (nav_msgs/Odometry) - от STM32 (micro-ROS)  
/tf (tf2_msgs/TFMessage) - от STM32 и static publisher  
/cmd_vel (geometry_msgs/Twist) - от move_base (или от teleop)  
/battery_status (пользовательский) - опционально
```

2. Прошивка STM32 (используем FreeRTOS, micro-ROS, HAL):

- Задачи: чтение энкодеров (TIM2, TIM3 в режиме Encoder) с периодом 10 мс.
- Вычисление одометрии ($\Delta_{\text{left}}, \Delta_{\text{right}} \rightarrow dS, d\theta \rightarrow x, y, \theta$).
- Комплементарный фильтр с MPU6050 для улучшения оценки угла (LIP4).
- ПИД-регулятор скорости (LIP15) с частотой 100 Гц.
- micro-ROS publisher для /odom (20 Гц), подписчик на /cmd_vel.
- При отсутствии команды > 0.5 секунды – остановка двигателей.

3. На Raspberry Pi:

- Установлены драйверы лидара (rplidar_ros), камеры (usb_cam или picamera2), slam_toolbox, nav2, ultralytics YOLO.
- micro-ROS агент запускается как systemd-сервис на /dev/ttyAMA0.

- Запускается узел детекции YOLOv8n (LP13) с публикацией /detection.
- Запускается узел слияния сенсоров (LP14) для приоритетов (лидар -> остановка, камера -> замедление).
- Запускается nav2 с загруженной картой (или slam_toolbox в режиме локализации).
- RViz2 для задания цели.

Пример интеграционного launch-файла (final_robot.launch.py):

```
from launch import LaunchDescription
from launch.actions import ExecuteProcess, DeclareLaunchArgument,
IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration, PathJoinSubstitution
from launch_ros.actions import Node
import os

def generate_launch_description():
    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    map_yaml = LaunchConfiguration('map', default='/home/pi/map.yaml')
    params_file = LaunchConfiguration('params',
    default='/home/pi/nav2_params.yaml')

    return LaunchDescription([
        # micro-ROS агент (UART)
        ExecuteProcess(
            cmd=['ros2', 'run', 'micro_ros_agent', 'micro_ros_agent',
                'serial', '--dev', '/dev/ttyAMA0', '-b', '115200'],
            output='screen'
        ),
        # Драйвер лидара
        Node(
            package='rplidar_ros',
            executable='rplidar_composition',
            name='rplidar',
            parameters=[{'serial_port': '/dev/ttyUSB0', 'serial_baudrate':
115200}],
            output='screen'
        ),
        # Камера
        Node(
            package='usb_cam',
            executable='usb_cam_node_exe',
            name='usb_cam',
            output='screen'
        ),
        # YOLO детектор (LP13)
        Node(
            package='yolo_detector',
            executable='detector',
            name='yolo_detector',
            parameters=[{'confidence_threshold': 0.5, 'processing_width': 320}]
        ),
        # Слияние сенсоров (LP14)
        Node(
```

```

        package='sensor_fusion',
        executable='fusion_avoider',
        name='sensor_fusion',
        parameters=[{'lidar_stop_dist': 0.4, 'camera_slow_dist': 1.2}]
    ),
    # Преобразование координат (статическое)
    Node(
        package='tf2_ros',
        executable='static_transform_publisher',
        arguments=['0', '0', '0.1', '0', '0', '0', 'base_link',
'lasер_frame']
    ),
    # Navigation2 (загрузка карты + планировщик)
    Node(
        package='nav2_bringup',
        executable='nav2_bringup',
        name='nav2_bringup',
        output='screen',
        parameters=[map_yaml, params_file, {'use_sim_time': use_sim_time}]
    ),
    # RViz
    Node(
        package='rviz2',
        executable='rviz2',
        name='rviz2',
        arguments=['-d', '/home/pi/final_nav.rviz']
    ),
    # Teleop (опционально, для ручного режима)
    # Можно закомментировать
    ExecuteProcess(
        cmd=['ros2', 'run', 'teleop_twist_keyboard',
'teleop_twist_keyboard'],
        output='screen'
    )
]

```

Для тестирования:

- Сначала построить карту помещения с помощью slam_toolbox (ЛР17) и сохранить как map.yaml.
- Запустить итоговый launch-файл.
- В RViz задать цель (2D Nav Goal). Робот должен начать движение к цели.
- Проверить реакцию на статическое препятствие (поставить коробку на пути) – робот объедет.
- Проверить реакцию на динамическое препятствие (человек пересекает путь) – робот должен замедлиться или остановиться в зависимости от конфигурации.

Ожидаемый результат: робот достигает цели с точностью ± 0.2 м и $\pm 5^\circ$.

Порядок выполнения индивидуального задания

- 1) изучить теоретический материал по теме лабораторной работы;
- 2) разобрать представленный пример выполнения задания;
- 3) выполнить индивидуальное задание по полученному варианту;
- 4) оформить отчёт: ход выполнения, листинги кода, скриншоты, выводы;
- 5) подготовиться к защите лабораторной работы.

Варианты индивидуальных заданий

№ варианта	Содержание индивидуального задания
1	Реализовать итогового робота на реальном аппаратном обеспечении (STM32 + Raspberry Pi + лидар + камера). Провести не менее трёх тестов: движение по прямой 5 м, движение с поворотом на 90°, движение к цели с объездом двух препятствий. Предоставить видео.
2	Полностью симулированный проект в Gazebo с моделью робота, имеющего энкодеры, камеру и лидар. Настроить micro-ROS для симуляции (используя pseudo-STM32). Реализовать навигацию по карте офиса. Оценить точность достижения цели в симуляции.
3	Вместо использования готовой карты, робот должен строить карту в реальном времени (SLAM) и одновременно выполнять навигацию (SLAM + navigation). Использовать slam_toolbox в режиме async с включённым планированием. Показать, что после обхода помещения можно задать цель и робот успешно к ней придёт.
4	Интегрировать детекцию и избегание людей (на основе YOLO) с приоритетом: при обнаружении человека на расстоянии менее 1.5

	м – скорость снижается до 0.1 м/с, при менее 0.8 м – робот останавливается и издаёт звуковой сигнал (пьезоизлучатель). После того как человек уходит, робот возобновляет движение.
5	Заменить лидар на стереокамеру (Intel RealSense D435) для получения глубины. Использовать глубину для построения карты препятствий (depthimage_to_laserscan) и публикации /scan. Настроить навигацию на основе этого искусственного лидара. Сравнить качество избегания препятствий.
6	Реализовать на STM32 не только одометрию, но и интеграцию с IMU (MPU6050) через комплементарный фильтр для улучшения оценки угла. При движении по дуге сравнить дрейф угла с и без IMU. Внести соответствующие коррективы в топик /odom.
7	Вместо ПИД-регулятора на STM32 реализовать на Raspberry Pi узел, который подписывается на /odom и вычисляет PID для каждого колеса, отправляя команды непосредственно в /cmd_vel_raw (пользовательский топик) на STM32. Это переносит управление в ROS. Сравнить задержки и качество.
8	Создать графический интерфейс на PyQt5 или веб-интерфейс на Flask для задания целей роботу без RViz. Интерфейс должен отображать карту, текущее положение робота, а также позволять кликнуть мышью для установки цели. ROS-узел транслирует клик в цель для move_base.
9	Добавить в систему функцию возврата на зарядную станцию при низком напряжении батареи (<11.5В). Использовать топик /battery_status (STM32 измеряет напряжение АКБ через АЦП). При срабатывании условия move_base отменяет текущую цель и публикует цель на зарядную станцию (координаты известны).
10	Реализовать режим следования за человеком (person following). С

	помощью YOLO определить человека и его смещение в кадре. Вычислить <code>cmd_vel</code>: линейная скорость пропорциональна обратному расстоянию, угловая – пропорциональна смещению центра bounding box. Отключить глобальное планирование. Продемонстрировать следование за движущимся человеком.
11	Использовать не RPLIDAR, а 2D-лидар на 360°, подключенный по Ethernet (например, LD06). Настроить драйвер, опубликовать <code>/scan</code> и выполнить навигацию на большой площади (коридор 20 метров). Измерить максимальную дальность надёжного обнаружения.
12	Ввести в систему элемент машинного обучения для классификации препятствий: по данным лидара (профиль) и камеры (изображение) решать, является ли препятствие проходимым (например, коробка – объезжать) или не проходимым (стена). Использовать нейросеть на Raspberry Pi.
13	Создать систему с двумя роботами (или один робот и статический маяк). Один робот публикует свою одометрию и цель, второй – избегает столкновений с первым на основе данных лидара и обмена по ROS (топик <code>/other_robot_pose</code>). Реализовать кооперативную навигацию.
14	Произвести нагрузочное тестирование: робот должен двигаться к цели в течение 30 минут (непрерывно) по круговому маршруту, выполняя 10–15 циклов. Записать ошибку накопления одометрии (конечное положение относительно старта). Построить график дрейфа.
15	Разработать полную документацию в формате PDF (не менее 10 страниц), включая: схемы подключения, блок-схему алгоритмов, листинги кода (основные функции), инструкцию по сборке и

	запуску, результаты экспериментов. Приложить видеодемонстрацию (ссылка). Защитить проект перед комиссией.
--	---

Содержание отчёта и его форма

Отчёт должен содержать название лабораторной работы, её цель, краткое описание аппаратного окружения (микроконтроллер, плата, датчики, ОПК), ход выполнения (листинги кода C/Python, результаты измерений, скриншоты), выводы. Отчёт оформляется в виде документа .docx или .pdf.

Вопросы для защиты работы

1. Опишите полный стек автономного робота.
2. Как организовать запуск всех компонентов через launch-файл?
3. Как обеспечить объезд динамических препятствий?
4. Что делает micro-ROS агент в финальной системе?
5. Как обработать потерю сигнала от сенсоров?
6. Как логировать работу системы (rosbag2)?
7. Что такое видео-демонстрация итогового проекта?
8. Какие показатели оценивают эффективность навигации?

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по организации и проведению самостоятельной работы по дисциплине
«Программная инженерия встроенных систем для автономных платформ»
для студентов направления подготовки
09.03.04 Программная инженерия
Направленность (профиль)
«Разработка и сопровождение программного обеспечения»

Ставрополь 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ	3
МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА	4
МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ	5
ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ....	6
КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ	7

ВВЕДЕНИЕ

Настоящие методические указания определяют порядок организации и проведения самостоятельной работы студентов по дисциплине «Программная инженерия встроенных систем для автономных платформ». Дисциплина реализуется в рамках направления подготовки 09.03.04 «Программная инженерия» (направленность «Разработка и сопровождение программного обеспечения»).

Цель самостоятельной работы — сформировать у студентов устойчивые навыки разработки бортового программного обеспечения для автономных платформ и готовность к практическому применению полученных знаний при решении инженерных задач.

Указания содержат общую характеристику видов самостоятельной работы, рекомендации по изучению теоретического материала, порядок подготовки к лабораторным работам, вопросы для собеседования и критерии оценивания компетенций.

ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Самостоятельная работа по дисциплине включает проработку теоретического материала, подготовку к лабораторным работам, выполнение индивидуального задания и подготовку к защите. Форма контроля — защита отчёта и собеседование.

Таблица 1 — Виды самостоятельной работы и формы контроля

Вид самостоятельной работы	Содержание деятельности	Форма контроля
Проработка теоретического	Изучение лекций, официальной документации STM32 HAL, micro-	Собеседование, тестирование

Вид самостоятельной работы	Содержание деятельности	Форма контроля
материала	ROS, ROS 2, OpenCV, YOLOv8, Nav2, slam_toolbox	
Подготовка к лабораторной работе	Изучение теоретического обоснования и примера, подготовка оборудования (STM32, Raspberry Pi, датчики), уточнение варианта задания	Допуск к выполнению работы
Выполнение индивидуального задания	Самостоятельное выполнение задания по варианту, оформление отчёта	Проверка отчёта
Подготовка к защите	Подготовка ответов на контрольные вопросы, обоснование принятых решений	Защита отчёта

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ИЗУЧЕНИЮ ТЕОРЕТИЧЕСКОГО МАТЕРИАЛА

Изучение теоретического материала рекомендуется проводить последовательно. По каждой теме следует изучить официальную документацию (STM32 HAL, micro-ROS, ROS 2 Humble, OpenCV, Ultralytics YOLOv8, Nav2), материалы лекций и рекомендованную литературу.

Таблица 2 — Темы для самостоятельного изучения теоретического материала

№	Тема для самостоятельного изучения	Вопросы для проработки
1–3	Продвинутая периферия STM32	DMA, ADC, энкодеры, таймеры в capture mode, одометрия

№	Тема для самостоятельного изучения	Вопросы для проработки
4–5	Слияние сенсоров	Комплементарный фильтр, EKF, MPU6050 (I2C)
6–7	Связь МК — ОПК	UART, бинарные фреймы, CRC, DMA, cmd_vel
8–9	micro-ROS и ROS 2	rcl, publisher, nav_msgs/Odometry, micro-ROS агент
10–11	Лидар	LaserScan, rplidar_ros, обнаружение препятствий
12–14	Компьютерное зрение	OpenCV, Canny, YOLOv8, cv_bridge, sensor fusion
15	ПИД управление	PID-регулятор, энкодеры как обратная связь, anti-windup
16–17	Навигация и SLAM	Nav2, slam_toolbox, costmap, move_base, map_server
18	Комплексный проект	Интеграция всех компонентов, launch-файл, демонстрация

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

Выполнение каждой лабораторной работы осуществляется самостоятельно в соответствии с полученным вариантом.

Таблица 3 — Этапы выполнения лабораторной работы

Этап	Содержание деятельности
Подготовительный	Изучение теоретического обоснования и примера, подготовка оборудования (STM32/Raspberry Pi), уточнение варианта задания

Этап	Содержание деятельности
Основной	Выполнение индивидуального задания: написание кода C (STM32) или Python (RPi), запуск, измерение, фиксация результатов
Оформление отчёта	Подготовка отчёта: цель, ход выполнения, листинги кода, измерения, скриншоты, выводы
Защита	Ответы на контрольные вопросы, обоснование принятых технических решений

ВОПРОСЫ ДЛЯ СОБЕСЕДОВАНИЯ ПО САМОСТОЯТЕЛЬНОМУ ИЗУЧЕНИЮ ЛИТЕРАТУРЫ

Для подготовки к собеседованию рекомендуется проработать следующие вопросы:

1. Что такое DMA? Как АЦП работает совместно с DMA в режиме circular? Каковы преимущества?
2. Как работает энкодер в режиме Quadrature? Как STM32 аппаратно определяет направление вращения?
3. Опишите кинематическую модель дифференциального привода. Как из двух энкодеров получить (x, y, theta)?
4. Сравните комплементарный фильтр и EKF. В каких условиях каждый предпочтителен?
5. Что такое CRC и зачем нужна контрольная сумма при UART-передаче? Как организовать бинарный протокол?
6. Что такое micro-ROS? Какова роль агента? Как STM32 публикует топики в ROS 2-экосистему?
7. Опишите структуру nav_msgs/Odometry. Как перевести угол theta в кватернион для 2D случая?
8. Что такое RPLIDAR? Какие поля содержит LaserScan? Как построить детектор препятствий на основе /scan?

9. Как работает алгоритм Canny? Какие параметры влияют на результат?
10. Что такое YOLOv8? Как запустить инференс на Raspberry Pi? Как ограничить нагрузку на CPU?
11. Как объединить данные лидара и камеры для принятия решений? Как организовать синхронизацию топиков?
12. Что такое ПИД-регулятор? Как каждая составляющая влияет на систему управления скоростью?
13. Как устроен навигационный стек Nav2? Что такое costmap и lifecycle_manager?
14. Что такое SLAM? Как slam_toolbox строит карту по данным лидара и одометрии?
15. Опишите полный стек автономного робота: от сенсоров до планирования пути.

КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценивание результатов осуществляется по итогам выполнения и защиты лабораторных работ и собеседования. Критерии приведены в таблице 4.

Таблица 4 — Критерии оценивания

Оценка	Критерии оценивания
«Отлично»	Задание выполнено без ошибок; отчёт соответствует требованиям; студент уверенно отвечает на все вопросы и обосновывает технические решения.
«Хорошо»	Задание выполнено с незначительными недочётами; отчёт с небольшими замечаниями; студент владеет материалом, допускает неточности.

Оценка	Критерии оценивания
«Удовлетворительно»	Задание выполнено частично; отчёт содержит ошибки; студент испытывает затруднения при ответах.
«Неудовлетворительно»	Задание не выполнено или выполнено с грубыми ошибками; отчёт отсутствует; студент не владеет материалом.