

ArGVIZ - телеметрия для роботов это просто

Необходимость телеметрии

Разработка системы управления - сложный и творческий процесс. Его продуктом является программа, объединяющая в себе работу с датчиками и исполнительными механизмами, внутренней логикой и большим количеством вычислений и различных функций.

В случае простой программы поведение робота достаточно легко предсказуемо и отлаживаемо даже по косвенным признакам его поведения. Но, к сожалению, чем сложнее программа, тем больше внутри логики и тем сложнее ее отлаживать вслепую.

В этом случае встает вопрос подробного взгляда внутрь поведения робота. Как это сделать наиболее просто и эффективно и не повлиять при этом на само поведение робота фактом наблюдения за ним?

Самым распространенным способом отлаживания роботов на базе Arduino является использование последовательного порта для вывода отладочной информации.

Отладка с использованием последовательного порта

Рассмотрим несложный пример:

```
#define SHARP_PIN A2
#define LED_PIN 13

void setup()
{
    pinMode(SHARP_PIN, INPUT);
    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
    if(analogRead(SHARP_PIN) < 200)
    {
        digitalWrite(LED_PIN, HIGH);
    }
    else
    {
        digitalWrite(LED_PIN, LOW);
    }
}
```

Эта программа считывает данные ИК-дальномера и по результату измерения зажигает или гасит светодиод.

Представим, что вам дали этот код и сказали сделать так, чтобы светодиод загорелся при расстоянии до объекта менее 20см.

Сначала рассмотрим способ отладки этой программы вслепую.

Слепая отладка

Первое - нам нужно понять в какую сторону изменяются значения датчика. Для этого загрузим ту программу которая у нас есть на микроконтроллер и проведем эксперимент - оценим на каком расстоянии изменяется значение индикатора и как.

Предположим, что на расстоянии 50см индикатор не светился, а при увеличении расстояния до 60см индикатор засветился.

Это нам говорит несколько важных вещей:

1. чем дальше объект, тем меньше значение на датчике;
2. значение 200 на датчике соответствует расстоянию 55 ± 5 см.

Значит нам нужно сделать несколько изменений в коде:

1. изменить знак сравнения с < на >;
2. подобрать новый порог расстояния для 20см.

Подбор точного значения операция долгая и неточная. Посмотрим как мы можем улучшить нашу жизнь с использованием последовательного порта.

"Зрячая" отладка

Для такой отладки немного изменим программу:

```
#define SHARP_PIN A2
#define LED_PIN 13

void setup()
{
    Serial.begin(9600);

    pinMode(SHARP_PIN, INPUT);
    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
    if(analogRead(SHARP_PIN) < 200)
    {
        digitalWrite(LED_PIN, HIGH);
    }
    else
    {
        digitalWrite(LED_PIN, LOW);
    }
}
```

```
}  
  
Serial.println(analogRead(SHARP_PIN));  
}
```

Теперь параллельно с основным действием программы в последовательный порт посылается информация об уровне сигнала на датчике.

И вместо слепого подбора мы можем поставить на расстоянии 20см объект и посмотреть в последовательный порт:

```
705  
707  
707  
704  
705  
705  
703
```

Из чего мы сможем выбрать подходящее значение порога и изменить условие:

```
if(analogRead(SHARP_PIN) > 705)
```

Подводные камни

Казалось бы, в чем могут быть проблемы такого подхода? В большинстве случаев подход работает хорошо и позволяет быстро и надежно смотреть за состоянием системы.

Проблемы же начинаются при разработке высокопроизводительных и сложных роботов, которые требуют большого количества времени процессора и высокой частоты обновления главного цикла, как например роботы для движения по линии, для соревнований Micromouse или при использовании большого количества регуляторов и математики для управления шагающим роботом.

Дело в том, что отправка большого количества данных по последовательному порту может вносить серьезную задержку. Так, например, на скорости 115200бод/с отправка строки:

```
dt: 13, left: 20, right: 24, target_left: 58, target_right: 59, err: 38,  
u: 34.6
```

в которой 80 символов в первый раз займет около 1250мкс, а каждый последующий (если не подождать отправки предыдущих байтов) будет уже занимать около 7000мкс (Рис. 1) [1].

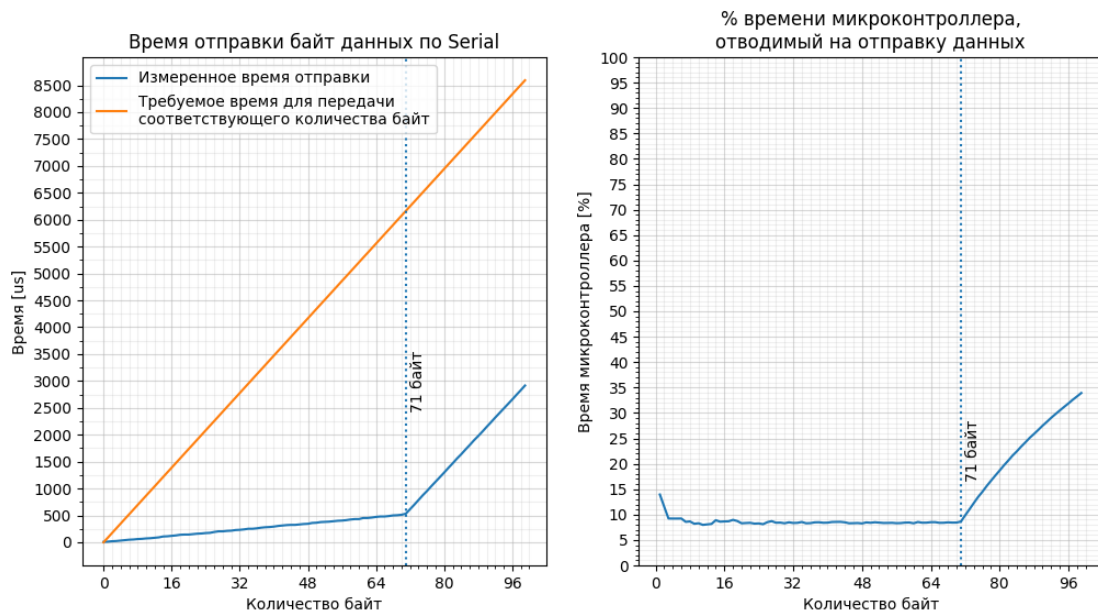


Рис. 1. Зависимость необходимого МК времени на отправку данных по последовательному порту на скорости 115200бод/с

Это происходит из-за того, что отправка данных по последовательному порту это долгий процесс и тем это заметнее, чем чаще мы пытаемся отправлять эти данные в главном цикле.

Эти ограничения всегда вынуждали нас на кружке Практической теории автоматического управления максимально минимизировать количество данных, которые посылаются. Все ради того, чтобы выдерживать период главного цикла 2мс (2000мкс), либо увеличивать его период, что негативно сказывалось на работе регуляторов и быстродействии микромыши.

Как нетрудно догадаться, это сильно усложняло отладку кода картографирования и проезда по лабиринту.

С целью решения этой проблемы была разработана библиотека ArGVIZ [2], [3].

Библиотека ArGVIZ

Библиотека ArGVIZ (Arduino Guidance Vizualisation library) предоставляет возможность посылать большой объем телеметрии в реальном времени по последовательному порту без блокирования главного цикла программы `void loop()` более чем на 10% от времени его выполнения.

Достигается это за счет использования того факта, что библиотека `Serial` отправляет данные в фоне с использованием буфера на 64 байта. Более подробно об этом можно прочитать в [1].

Помимо этого, библиотека предоставляет интерфейс для взаимодействия с программой с помощью клавиш клавиатуры, что позволяет быстро запускать функции и процедуры без перезапуска МК. Это очень важно при отладке сложных систем.

Как ей пользоваться?

Главный концепт библиотеки: 10 кастомизируемых Экранов, между которыми можно переключаться. В один момент времени показывается один из 10 Экранов. Пример внешнего вида

интерфейса:

```
+-----+
|0123456789|23fs| 332| 508|
|-----|
|counter [us]: 151957700 |
|counter [ms]: 151960    |
|click me (h|l|space):  0|
|HINT:                  |
|control with hjkl+space |
|                       |
+-----+
```

В нижней области экрана находится данные, которые выводятся на соответствующем Экране.

В верхней области показан фиксированный набор информации:

```
+-----+
|0123456789|23fs| 332| 508|
|-----|
|\_____/ \_/ \_/ \_/ |
| |           | |   `период вызова void loop() в микросекундах
| |           | |   `время выполнения void loop() в микросекундах
| |           | |   `частота обновления интерфейса (кадры в секунду, fps)
| |           | |   `список всех экранов; выделением показан текущий экран
| |           | |
+-----+
```

С интерфейсом можно взаимодействовать с помощью клавиш **h****j****k****l****+****space**, по аналогии с перемещением в **vim**:

```

      ↑
      k      Hint: The `h`{normal} key is at the left and moves
left.
    ← h      l →      The `l`{normal} key is at the right and moves
right.
      j      The `j`{normal} key looks like a down arrow.
      ↓
```

Пробел (**space**) используется для нажатия на элементы интерфейса.

Задаются Экраны с помощью специального макроса **SCREEN**, внутри которого прописывается до 6 Строк, отображаемых на соответствующем Экране. Строки могут просто отображать произвольную информацию (**ROW**), либо также иметь три обработчика нажатий (**CLICK_ROW**) влево-вправо-вниз (**hl+space**).

Простейший пример программы, использующей все возможности библиотеки представлен ниже:

```
#include <Arduino.h>
#include <argviz.h>

uint8_t counter = 0;

SCREEN(screen0,
{
    ROW("counter [us]: %lu", micros());
    ROW("counter [ms]: %lu", millis());
    CLICK_ROW([])(CLICK_STATE state)
    {
        switch(state)
        {
            case CLICK_LEFT:
                counter--;
                break;
            case CLICK_RIGHT:
                counter++;
                break;
            case CLICK_DOWN:
                counter = 0;
                break;
            default:
                break;
        } }, "click me (h|l|space): %3u", counter);

    ROW("HINT:")
    ROW("control with hjkl+space")
})

void setup()
{
    Serial.begin(115200);

    argviz_init(Serial);
    argviz_registerScreen(0, screen0);
    argviz_start();
}

void loop()
{
    static uint32_t timer = 0;
    while (micros() - timer < 500)
        ;
    timer = micros();
}
```

Более расширенную информацию об использовании библиотеки можно найти в документации к библиотеке, расположенной в git-репозитории [2] и на ее странице в регистре PlatformIO [3].

Заключение

В заключение хочется сказать - всегда уделяйте особое внимание телеметрии и индикации в своих роботах! Качественное наблюдение за внутренним состоянием логики робота позволяет значительно ускорить разработку и предупредить большое количество скрытых багов, которые могут проявлять себя очень неочевидно.

Удачной разработки!

Источники

[1] «Эксперименты со скоростью отправки данных по Serial». Просмотрено: 12 сентябрь 2025 г. [Онлайн]. Доступно на: <https://arsenier.github.io/serialBenchmark/serialBenchmark.html>

[2] «Библиотека ArGVIZ: GIT репозиторий». Просмотрено: 12 сентябрь 2025 г. [Онлайн]. Доступно на: <https://github.com/arsenier/argviz>

[3] «Библиотека ArGVIZ: Регистр PlatformIO». Просмотрено: 12 сентябрь 2025 г. [Онлайн]. Доступно на: <https://registry.platformio.org/libraries/arsenier/argviz->