

# Занятие 1

## Мобильные роботы



# Задача – моделирование складского робота в короткие сроки

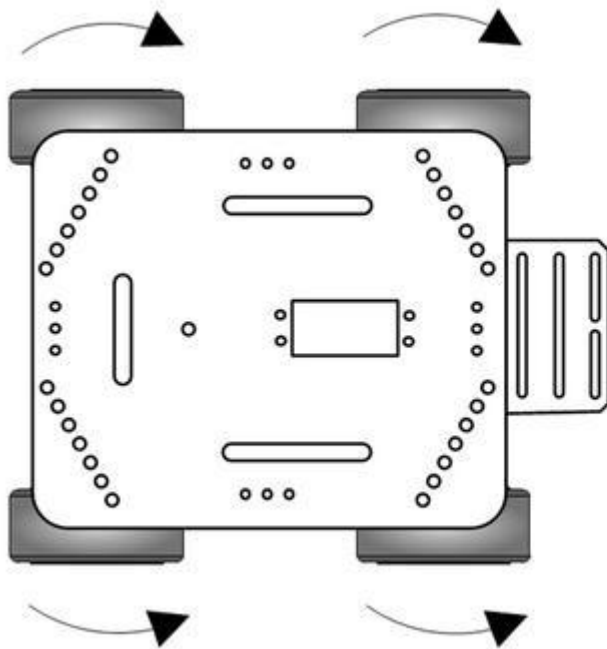




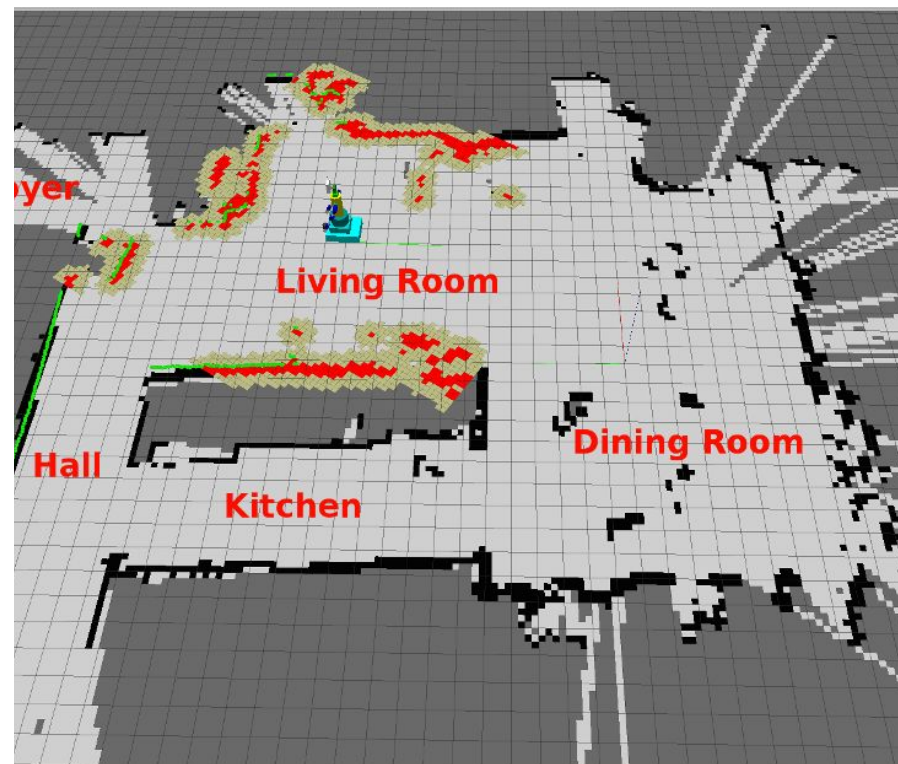
# Распределение обязанностей



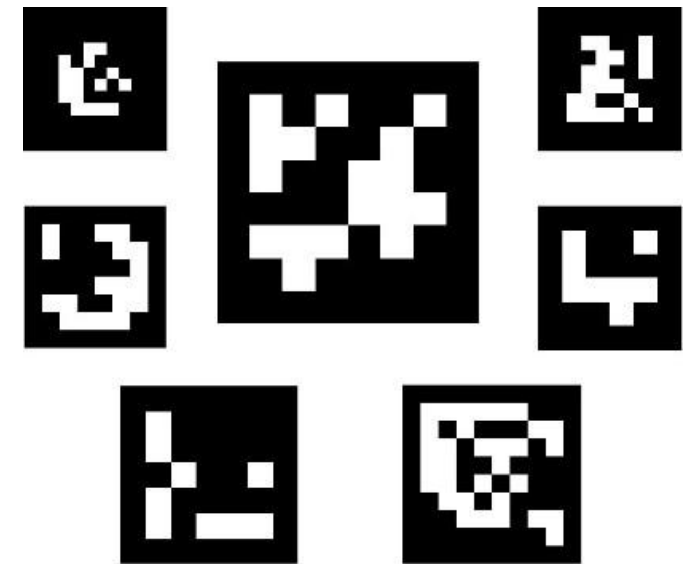
## Нижний уровень Шасси



## Навигация



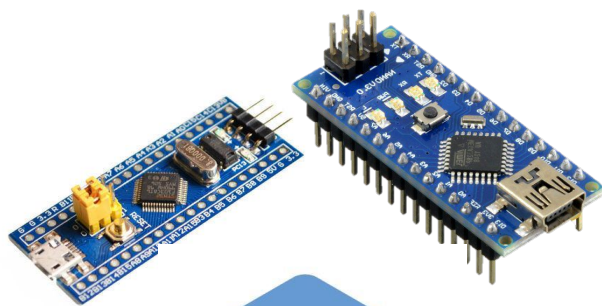
## Computer vision



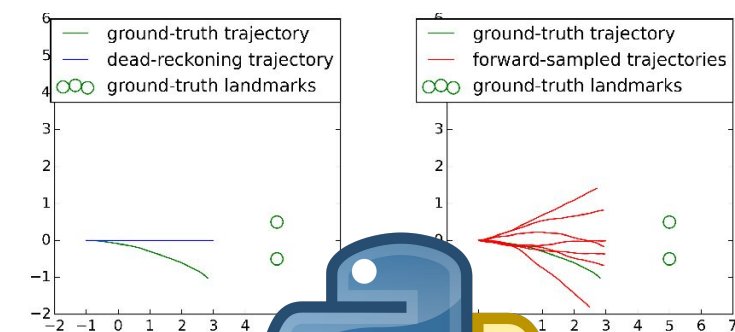
# Стек технологий



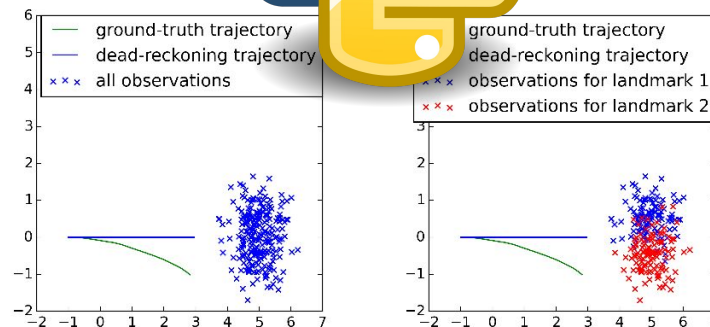
## Нижний уровень Шасси



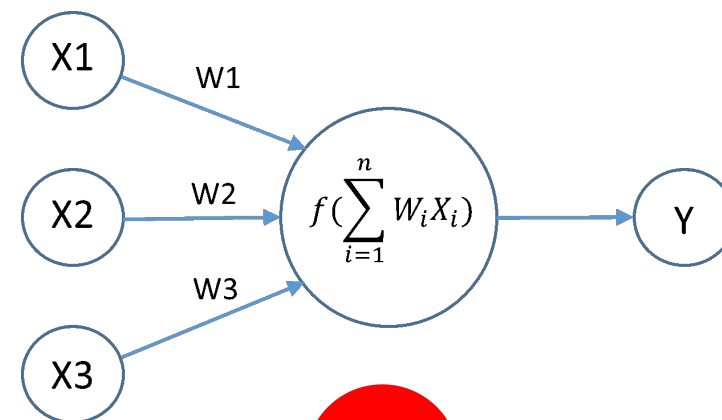
## Навигация

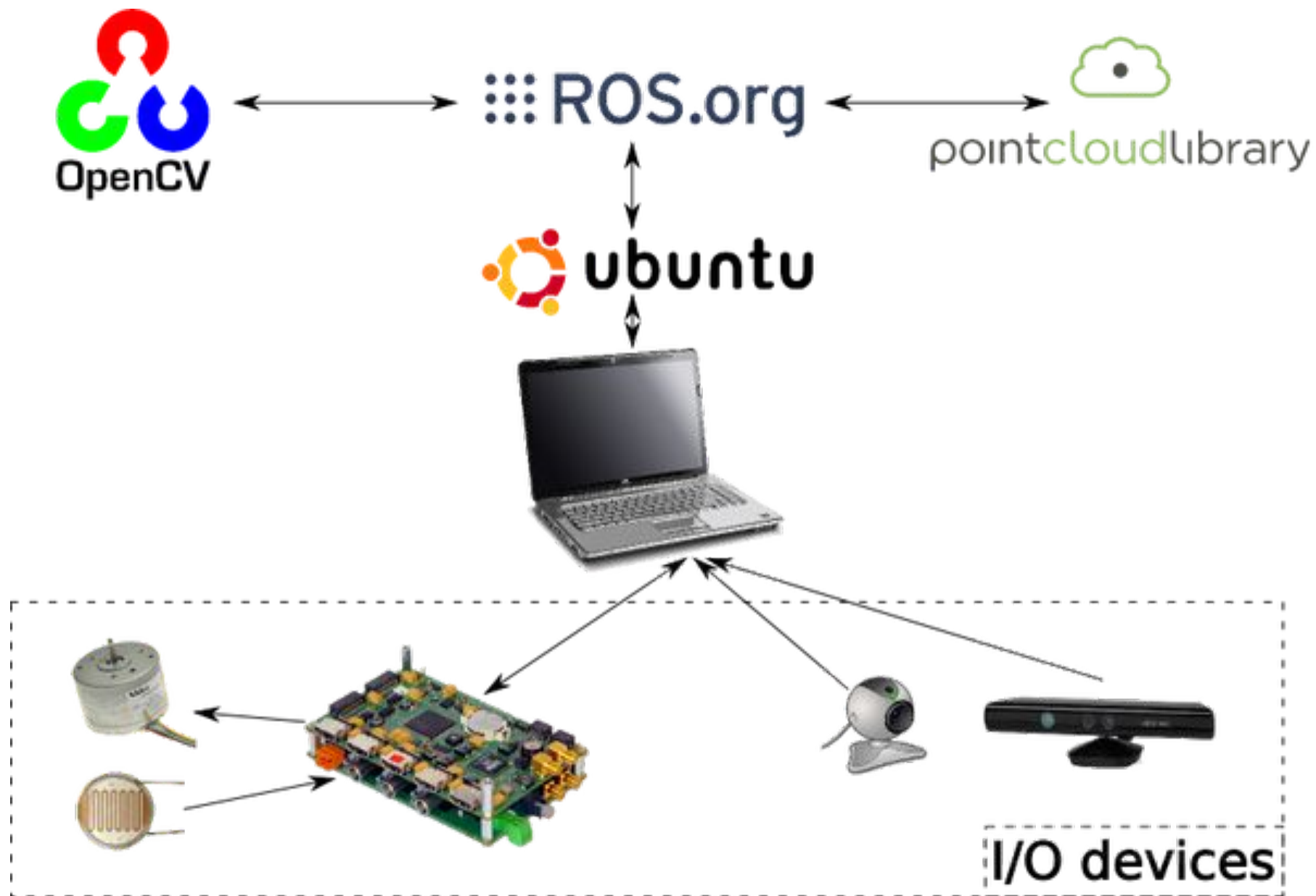


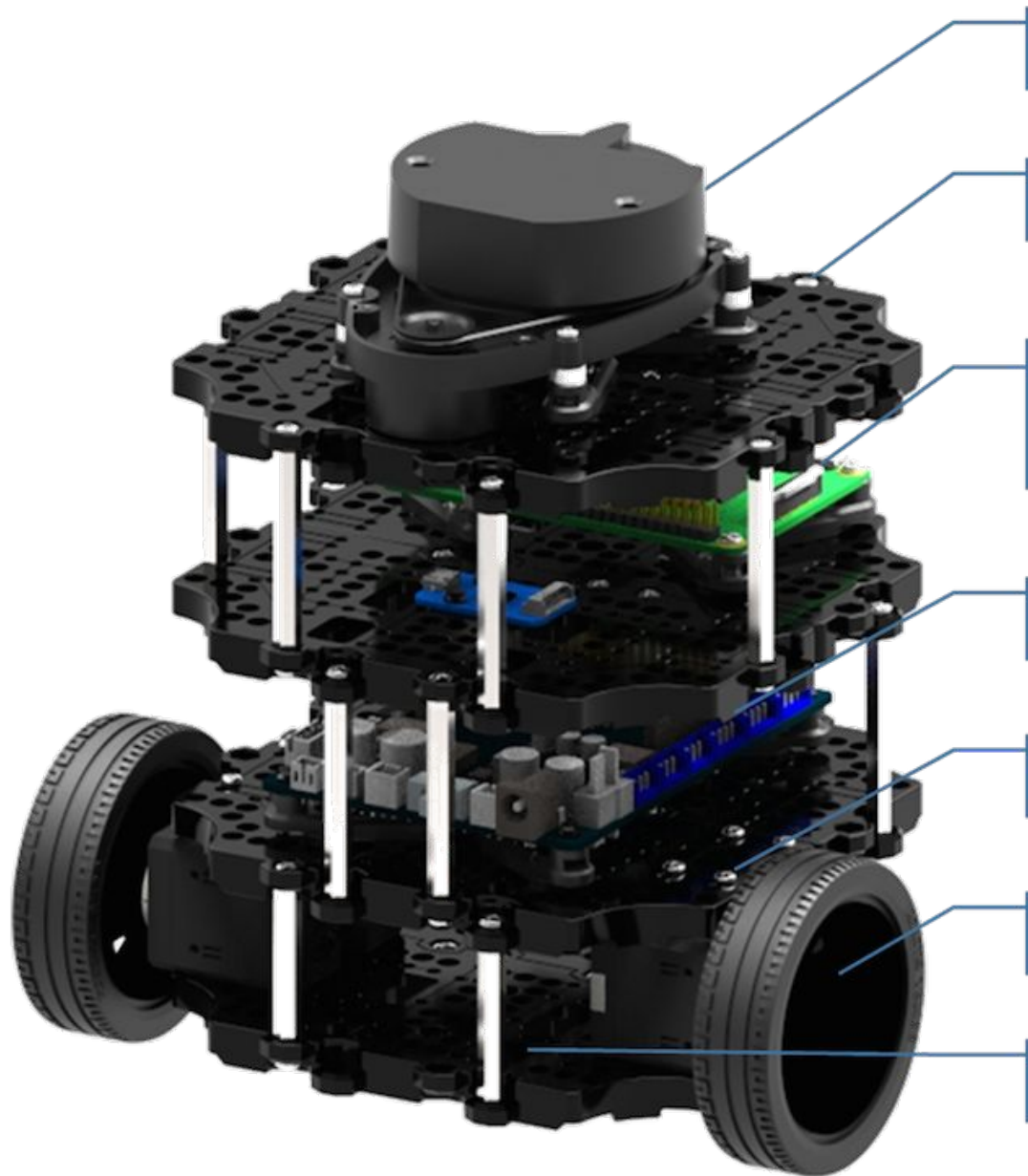
(a) The ground-truth trajectories and the trajectories sampled from the dead-reckoning trajectories.



## Computer vision







360° LiDAR for SLAM & Navigation

Scalable Structure

Single Board Computer  
(Raspberry Pi)

OpenCR (ARM Cortex-M7)

DYNAMIXEL x 2 for Wheels

Sprocket Wheels for Tire and Caterpillar

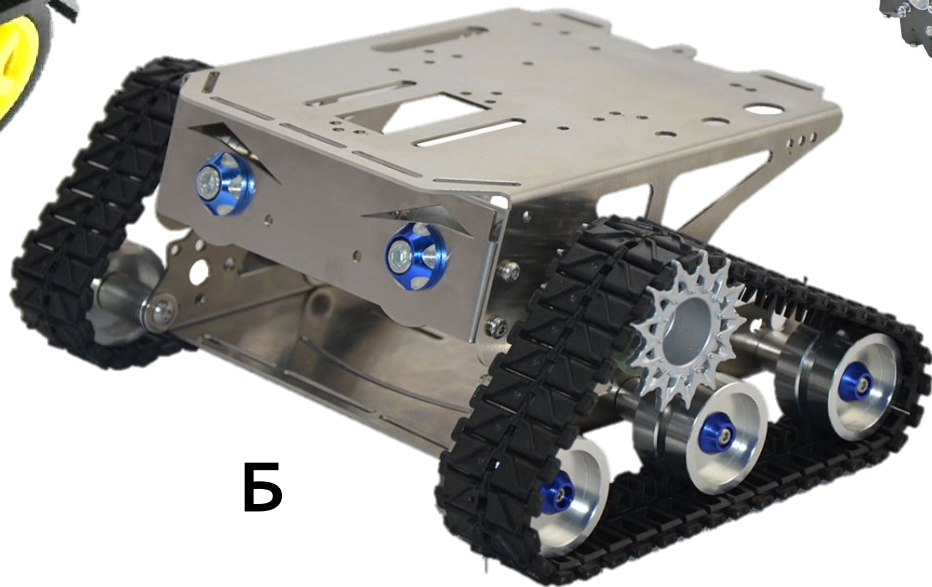
Li-Po Battery 11.1V 1,800mAh



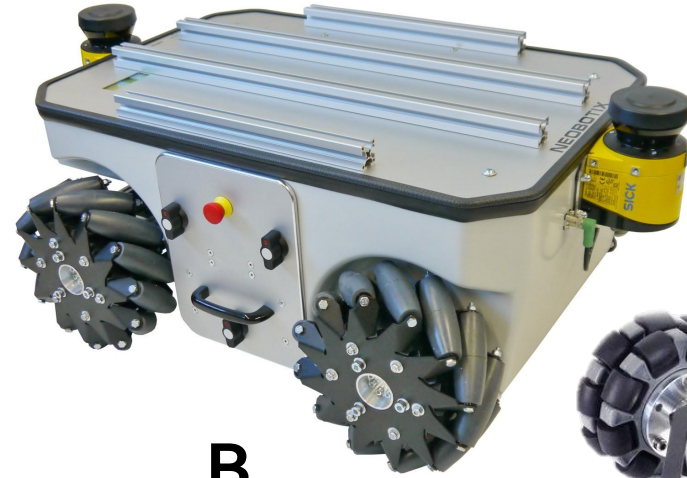
# Моделирование шасси



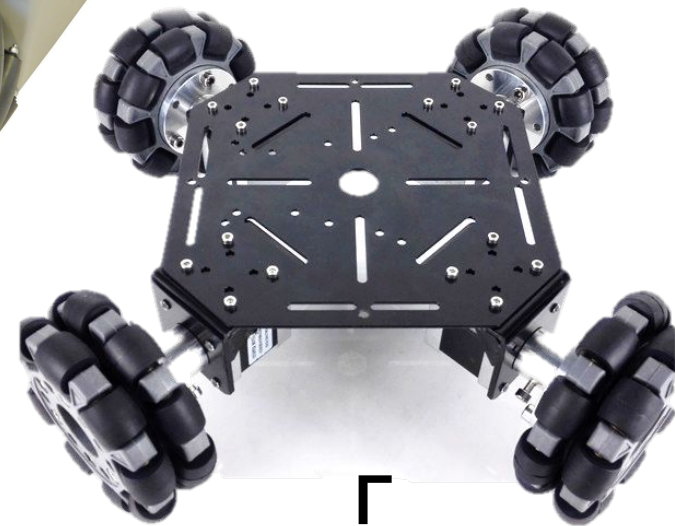
**А**



**Б**

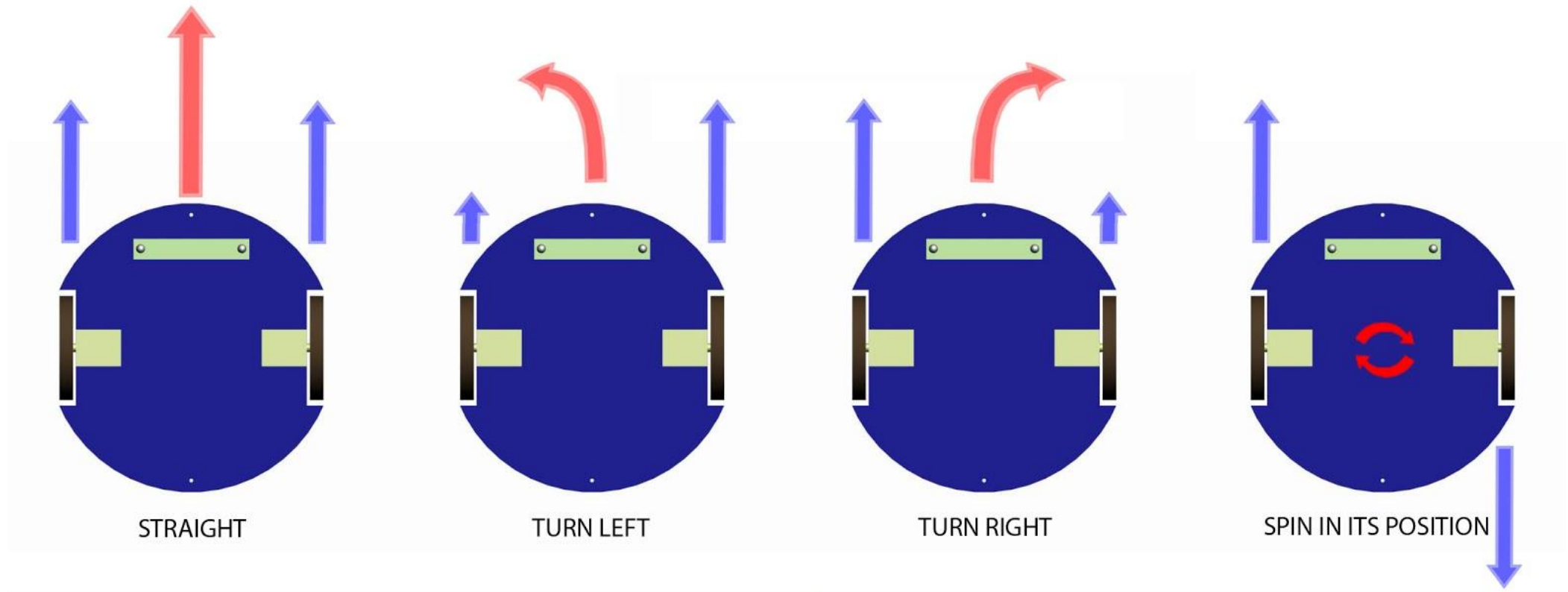


**В**



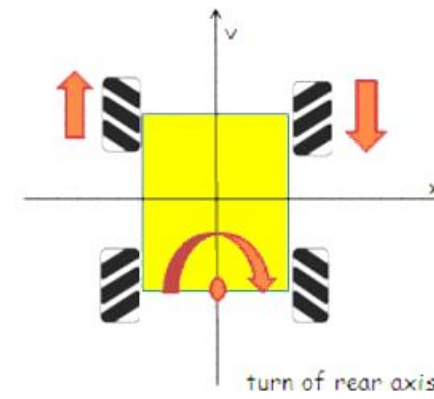
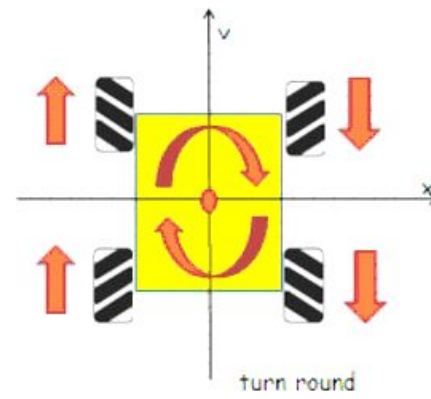
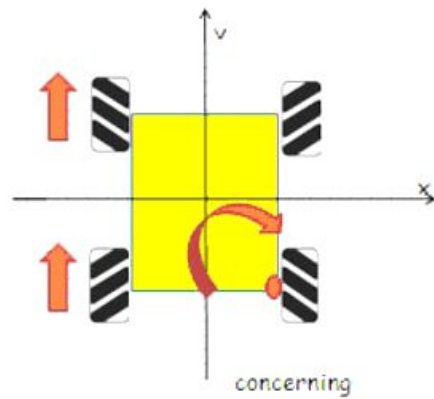
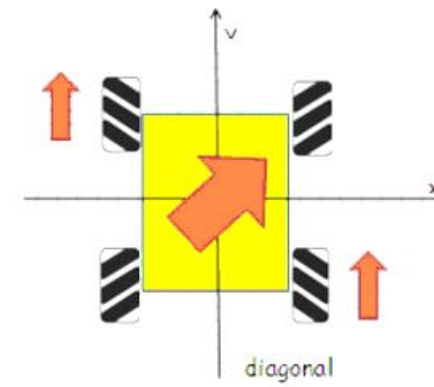
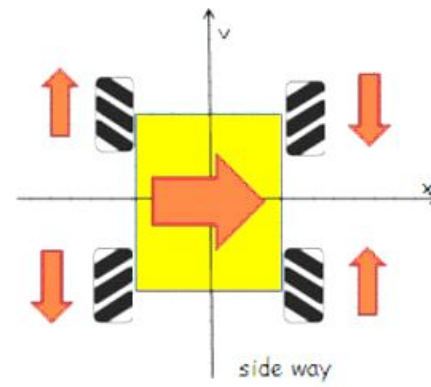
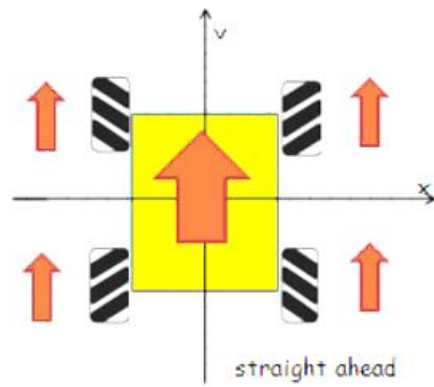
**Г**

# Дифференциальный привод

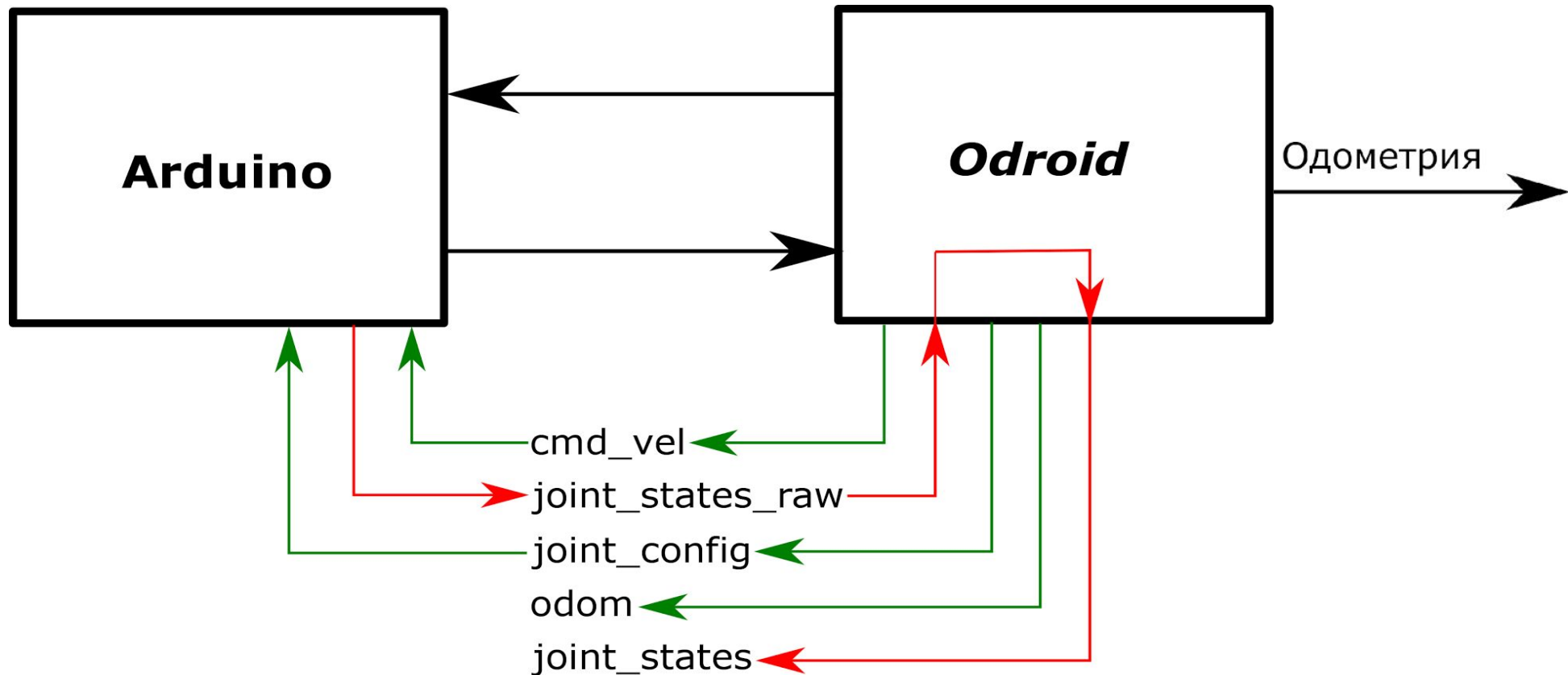




# Меканум



# Данные



# В каком виде нам приходят запросы скорости?



## geometry\_msgs/Twist Message

File: `geometry_msgs/Twist.msg`

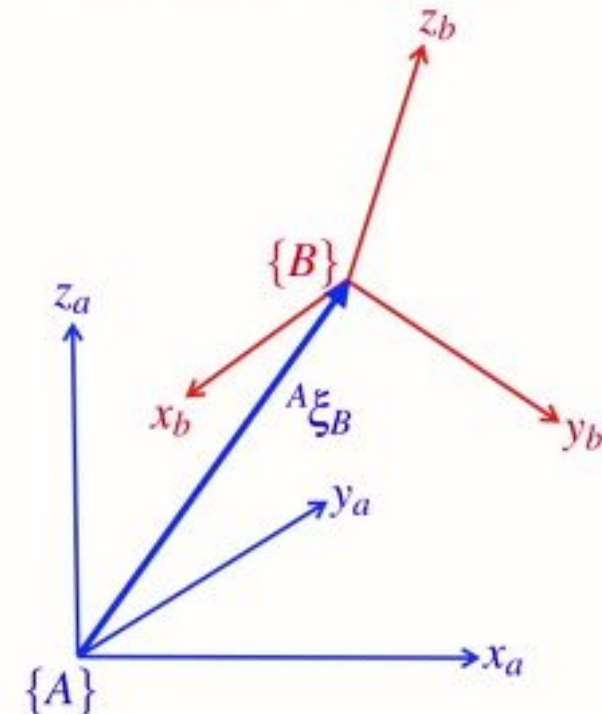
### Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.  
Vector3 linear  
Vector3 angular
```

### Compact Message Definition

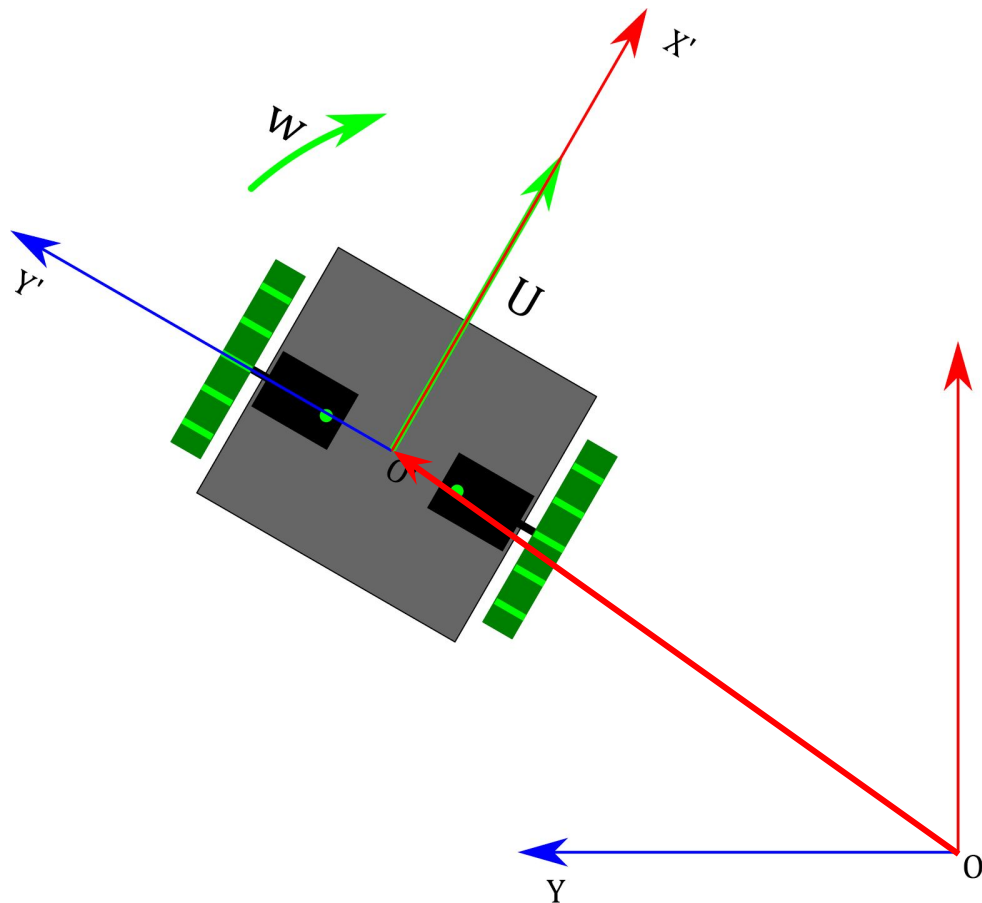
```
geometry_msgs/Vector3 linear  
geometry_msgs/Vector3 angular
```

## Homogeneous form





# Почему Twist?



This does the rotation →

$$H = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix}$$

This does the displacement →

$$= \begin{bmatrix} n_x & s_x & a_x & d_x \\ n_y & s_y & a_y & d_y \\ n_z & s_z & a_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



# Arduino IDE



*// Блок, выполняемый один раз при старте*

```
void setup() {
```

*// Задаем режим работы пина 13 на выход*

```
pinMode(13, OUTPUT);
```

```
}
```

*// Блок, повторяющийся до выключения контроллера*

```
void loop() {
```

```
digitalWrite(13, HIGH); // Зажечь светодиод
```

```
delay(1000); // Подождать секунду
```

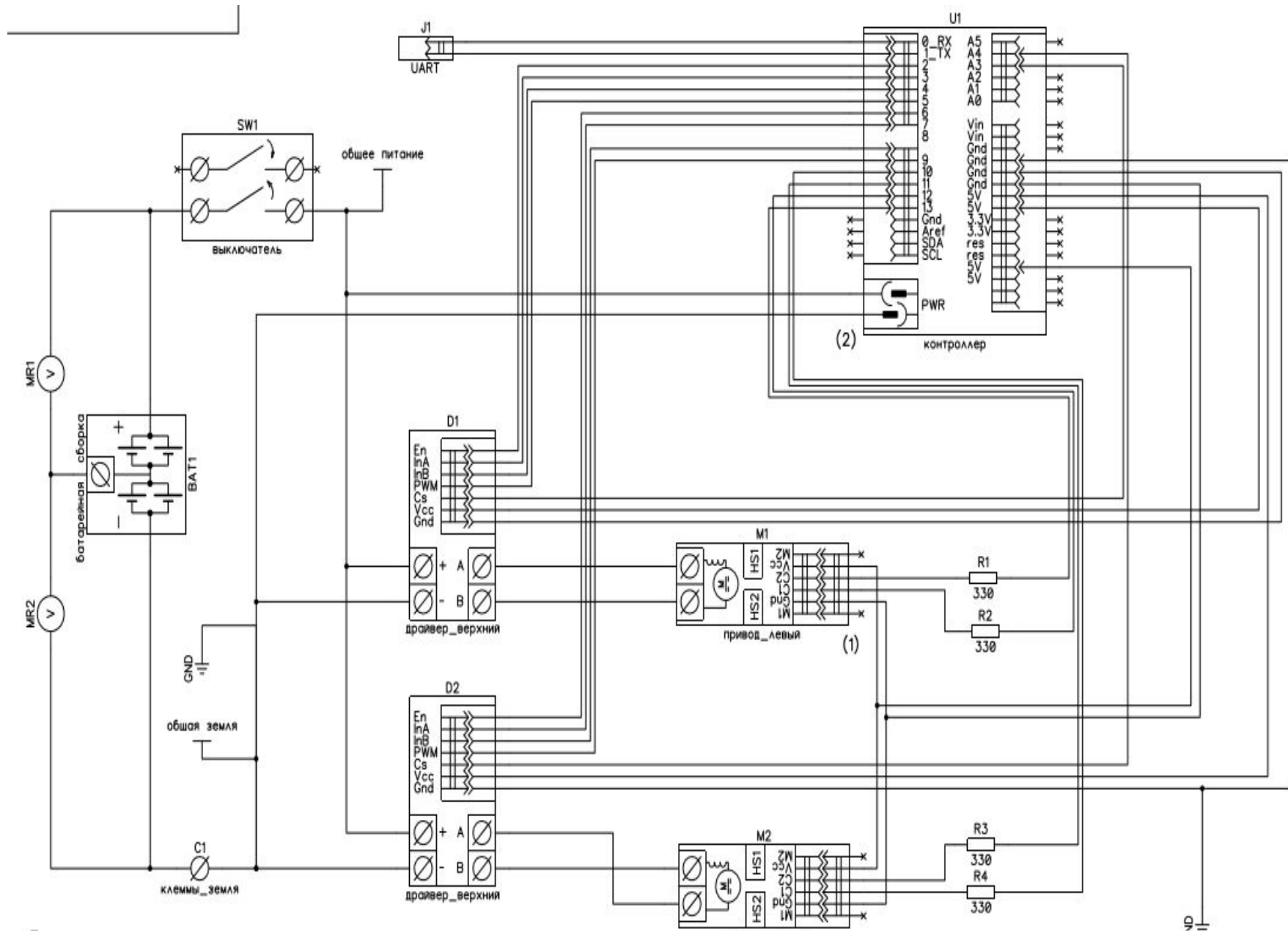
```
digitalWrite(13, LOW); // Погасить светодиод
```

```
delay(1000); // Подождать секунду
```

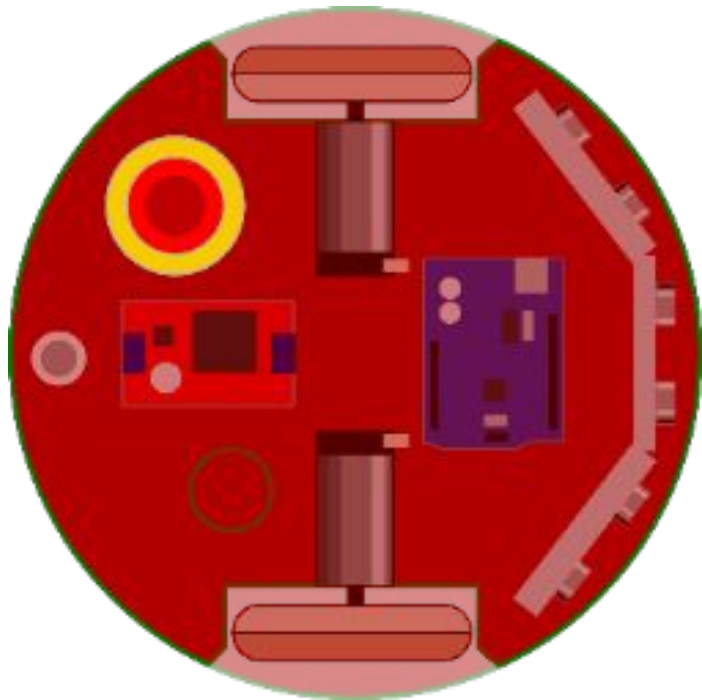
```
}
```



```
void setup() //Функция setup, вызываемая при старте работы контроллера
{
    pinMode( pin, INPUT/OUTPUT); //Инициализация пина
}
void loop() //Функция loop, выполняющаяся бесконечное количество раз после setup
{
    function(1000); //вызов функции function с аргументом 1000
}
void function( int argument) //Функция function, с аргументом argument тип целого
{
    digitalWrite( pin, HIGH/LOW(1/0)); //Подать высокое/низкое значение напряжение на пин
    delay(argument); //Ожидать определенное количество миллисекунд
    //analogWrite( pin, 0..255); Подать напряжение между высоким и низким, где 0- 0v, а 255 ~ 5v
    delay(argument);
}
```



# Управление моторами



Двигатель

Драйвер

Контроллер





**Драйвер 1**

**$E_n \rightarrow 2$**

**$InA \rightarrow 4$**

**$InB \rightarrow 3$**

**$PWM \rightarrow 5$**

**$InA/InB$**  ↓

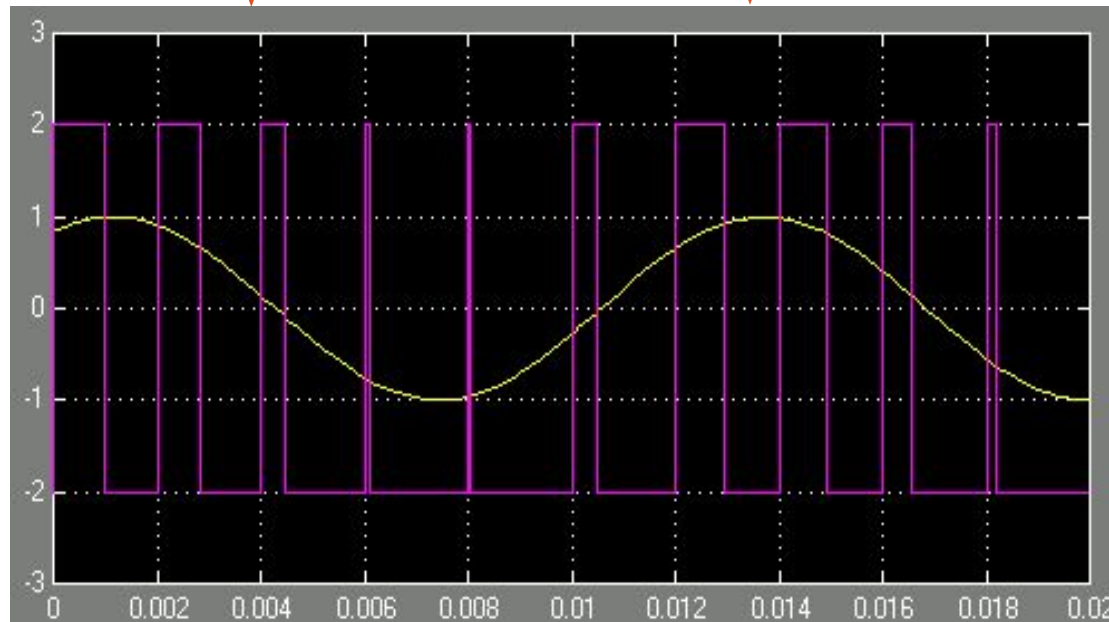
**Драйвер 2 (Инвертирован)**

**$E_n \rightarrow 6$  (есть не всегда)**

**$InA \rightarrow 8$**

**$InB \rightarrow 7$**

**$PWM \rightarrow 9$  (ШИМ иногда завязан на**



***#define LPWM 5 //Left PWM - управление скоростью  
левого колеса***

***#define LCCW 4 //Left Counter Clockwise - вращение  
против часовой***

***#define LCW 3 //Left Clockwise - вращение по часовой***

***#define LEN 2 //Left Enable***

***#define RPWM 9 //Right PWM - управление скоростью  
правого колеса***

***#define RCCW 8 //Right Counter Clockwise - управление  
скоростью правого колеса***

***#define RCW 7 //Right Clockwise - вращение по часовой***

***#define REN 6 //Right Enable***

```
void setup(){
```

```
...
```

```
//Выставляем Enable в "1"
```

```
digitalWrite(LEN,HIGH);
```

```
digitalWrite(REN,HIGH);
```

```
}
```

```
void Lmove( int speed)
```

```
{
```

```
...
```

```
}
```

```
void Rmove( int speed)
```

```
{
```

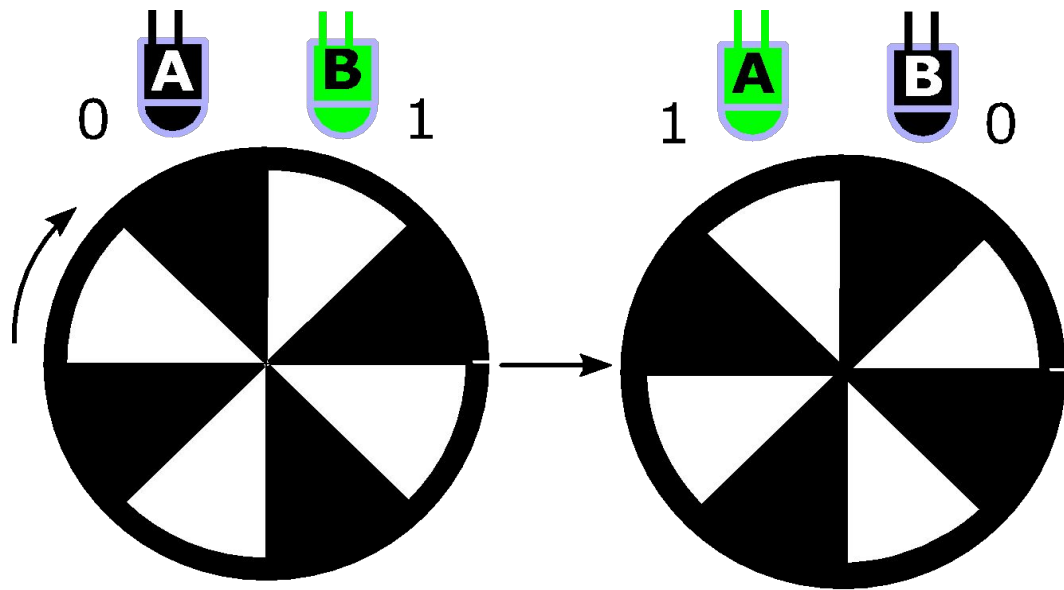
```
...
```

```
}
```

```
void Lmove( int speed)
{
    if (speed >= 0)
    {
        digitalWrite(LCW, HIGH); //Выставляем пин "По часовой стрелке" в 1
        digitalWrite(LCCW, LOW); //А против часовой- в 0.
    }
    else //если меньше 0, то против часовой
    {
        speed=-speed; //Помним, что analogWrite воспринимает только положительные значе
        digitalWrite(LCCW, HIGH); //аналогично
        digitalWrite(LCW, LOW);
    }
    analogWrite(LPWM, speed); //И теперь уже выставляем саму скорость
}
```



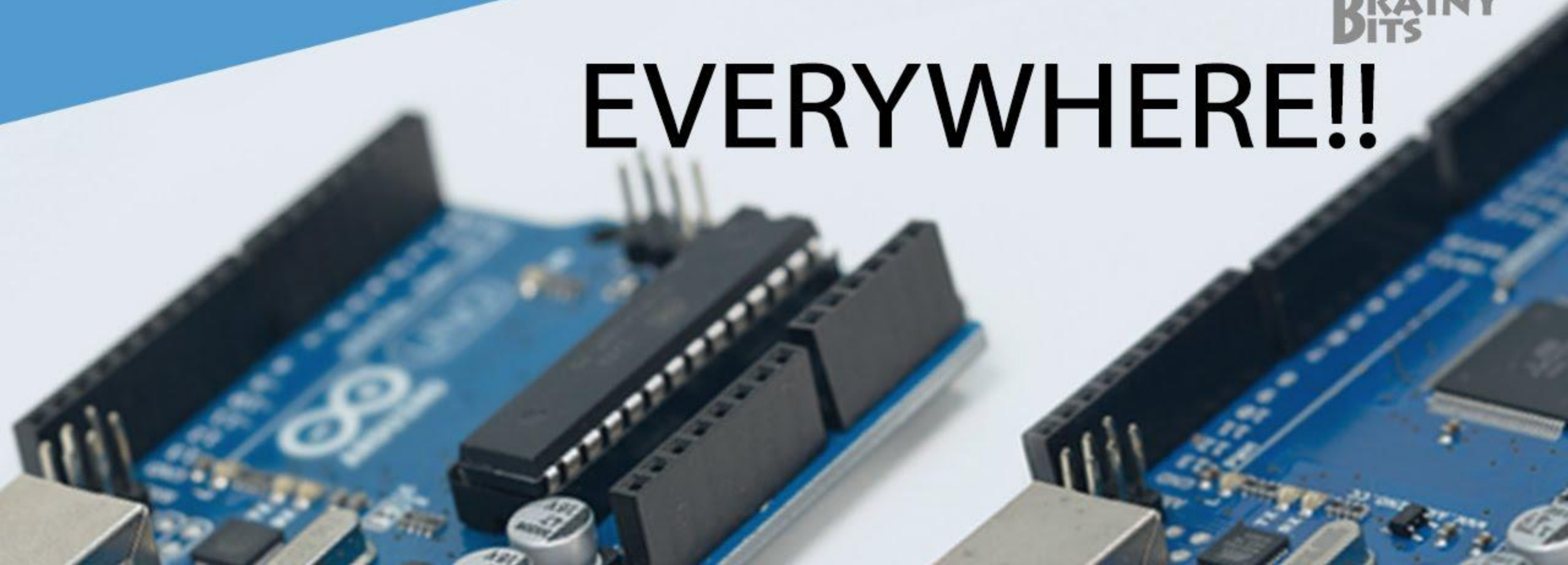
# Движение прямо: Энкодеры и ПИД регулятор



**INTERRUPTS,**

**INTERRUPTS**

**EVERYWHERE!!**



Другими словами, нам достаточно привязать прерывание на RISING одного из пинов, а внутри самой функции прерывания считывать с помощью функции digitalRead(pin2) в "1" или в "0" находится другой пин.

После этого, в той же функции мы либо прибавляем, либо отнимаем единицу от значения суммы тиков.

*attachInterrupt(interruptPin, function, CHANGE);*

*interruptPin* – наш пин прерывания, который реагирует на сигнал

*function* – функция, которая будет вызываться при получении сигнала

*CHANGE* – фильтрация рабочего сигнала ( *CHANGE, FALLING, RISING, LOW* )

*digitalRead(pin)* – Читает в каком именно состоянии находится пин (0/1)

```

#include <PinChangeInt.h> //Подключаем заголовочный файл библиотеки
#define RENCA 11 //Энкодер на правом двигателе. Сигнал A
#define RENCB 10 //Энкодер на правом двигателе. Сигнал B
#define LENCA 12 //Энкодер на левом двигателе. Сигнал A
#define LENCB 13 //Энкодер на левом двигателе. Сигнал B
char buf[128]; //Буфер сообщения
int ltickes, rtickes; //Тику на левом и правом колесах

void setup(){
  Serial.begin(9600); //Настраиваем скорость общения контроллера и компьютера
  //Мы будем считывать состояние с пинов, так что их надо поставить в INPUT
  pinMode(LENCA, INPUT);
  pinMode(LENCB, INPUT);
  pinMode(RENCA, INPUT);
  pinMode(RENCB, INPUT);

  //Прерывание Left Encoder A и Right Encoder A привязываем к переднему фронту сигнала A. В соответствующей функции будем анализировать состояние сигнала B
  attachPinChangeInterrupt(LENCA, LinterruptFunc, RISING);
  attachPinChangeInterrupt(RENCA, RinterruptFunc, RISING);

  //Для безопасности вначале программы обнуляем тику
  ltickes=0;
  rtickes=0;
}

void loop(){
  sprintf(buf, "Left wheel: %d; Right: %d", ltickes, rtickes); //sprintf позволяет формировать буфер сообщения
  Serial.println(buf); //который мы будем выводить в терминал
  delay(200); //каждые 200 миллисекунд
}

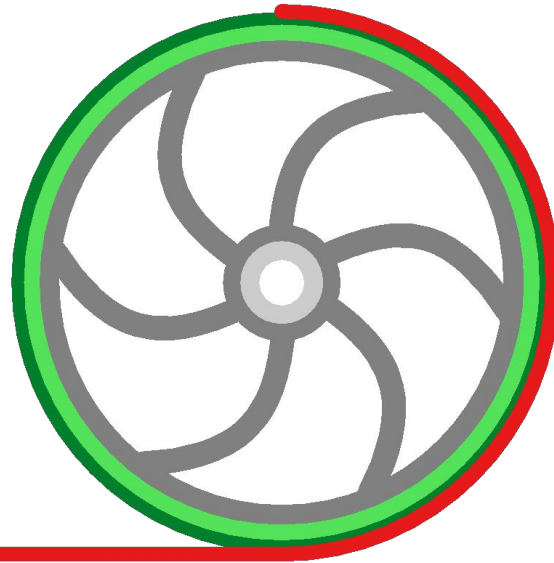
```





## Получение скоростей по энкодерам

$L$



Диаметр  $D$  колеса 82 мм

Длина дуги равен  $P_i * D = 82 * P_i$

Количество тиков на оборот равно 390

За  $x$  тиков делается  $x/390$  оборотов

Тогда пройденное расстояние будет равно произведению количества оборотов на длину дуги всего колеса. То есть, пройденный колесом путь равен:

$S = (X/390) * (82 * \pi)$ , где  $X$ - количество тиков на колесе.

Скорость вращения колес равна

$V = S/t = (X / 390) * (82 * \pi) / t$ , где  $t \rightarrow 0$

```
...  
int Lvel, Rvel; //скорости правого и левого колес  
long timer; //таймер, тип long (больше, чем int, но  
тоже число будет кончено)  
int delta; //дельта времени замера
```

```
void setup() {  
...  
timer=millis(); //начальный момент времени  
}
```

```
void loop() {  
//замеряем дельту времени каждую  
итерацию цикла. Если она будет >= 20  
миллисекунд, то  
if ((delta = millis() - timer) >= 20)  
{  
//рассчитываем скорости  
Lvelcalc();  
Rvelcalc();  
//переключаем таймер  
timer=millis();  
}  
printf(buf, "Left wheel: %d; Right: %d", (int)Lvel,  
(int)Rvel);  
Serial.println(buf);  
}
```

**//Функции расчета скоростей**

**void Lvelcalc() {**

**/\*Скорость равна количеству оборотов за единицу времени \* длину дуги колеса.**

**\*Длина дуги колеса =  $PI * D$ , где  $D = 82$  мм.**

**\*Количество оборотов за единицу времени = разность тиков, деленная на количество тиков за оборот**

**\*и умноженная на дельту времени (так как дельта времени в миллисекундах, то ее делим на 1000).**

**\*Количество тиков за ~ равно 390.1 (определяется документацией на энкодер).**

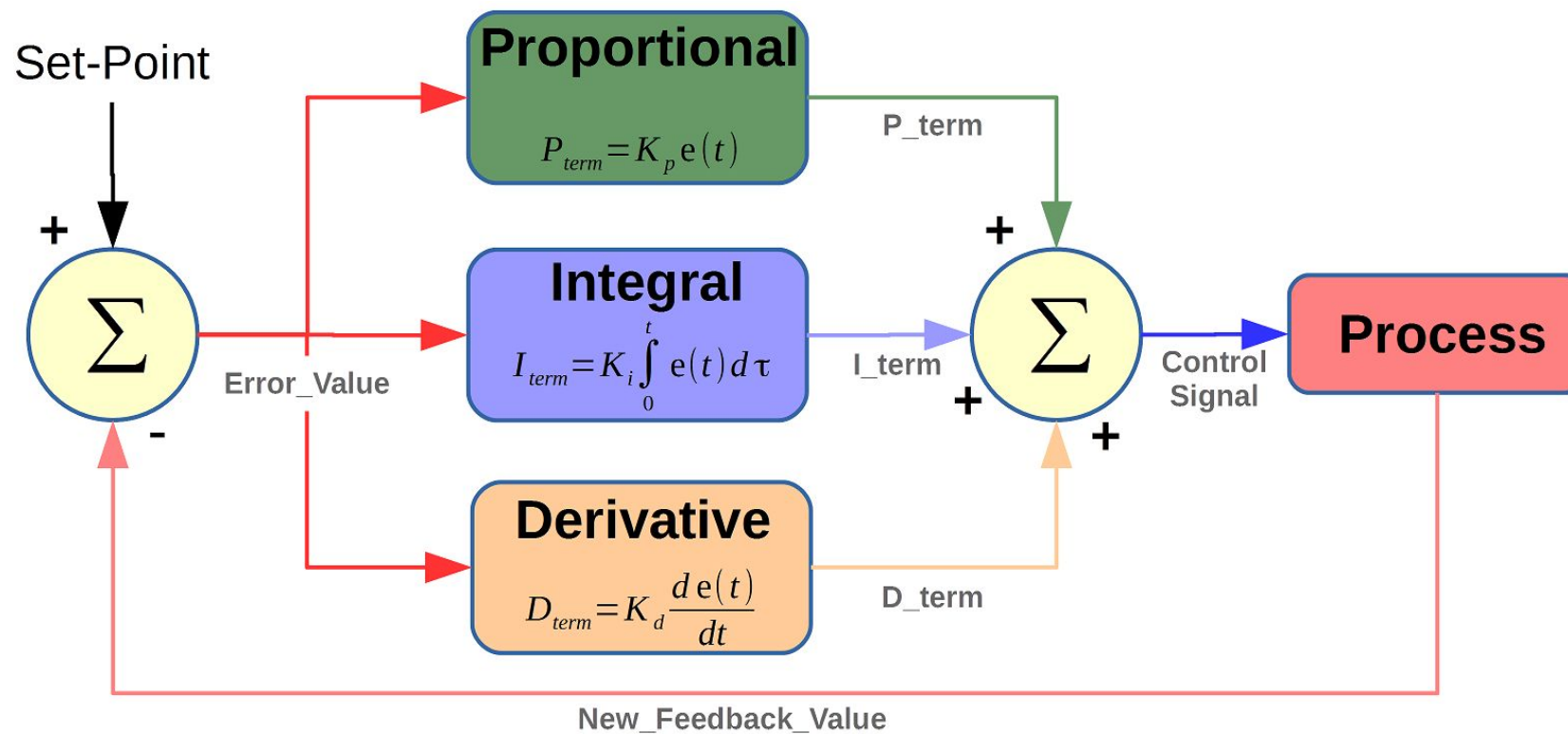
**\*/**

**Lvel= ((Itickes/390.1)\*1000/delta)\*PI\* 82;**

**Itickes = 0; //обнуляем тики, чтобы не допустить переполнение переменной тиков**

**}**

# ПИД регуляторы



```
void loop()
{
  if ((delta = millis() - timer) >= 20)
  {
    //расчитываем скорость левого колеса
    Lvelcalc();
    //ошибка левого колеса
    error = reqvel - Lvel;
    //расчет компенсации
    analogchange = error * p;
    //сама компенсация
    Lmove(analogchange);
    //аналогично для правого
    Rvelcalc();
    error = reqvel - Rvel;
    analogchange = error * p;
    Rmove(analogchange);
    timer = millis();
  }
  ...
}
```

# П регулятор



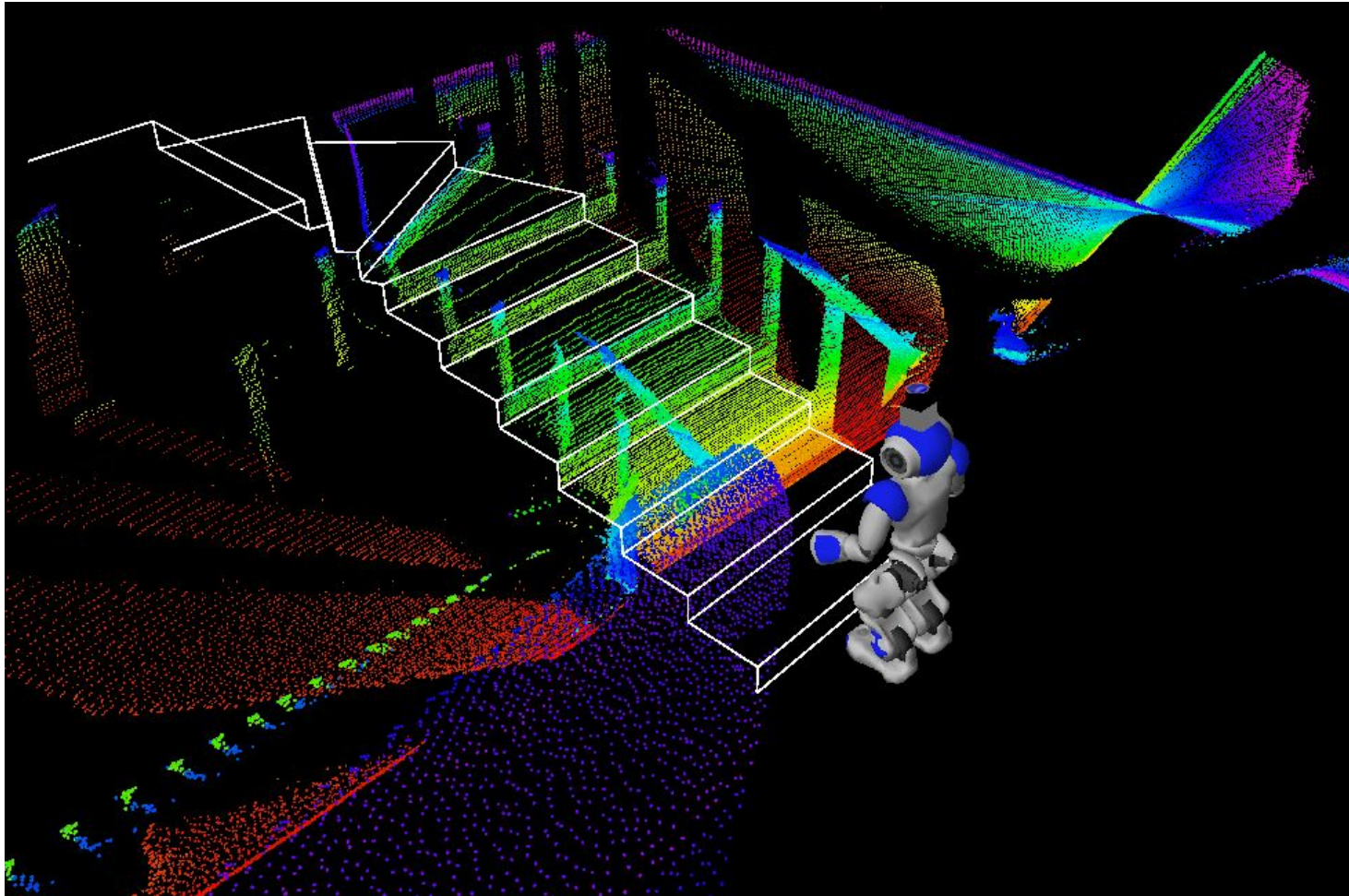
```
void setup() {...}  
void loop()  
{  
  if ((delta = millis() - timer) >= 20)  
  {  
    //рассчитываем скорость левого колеса  
    Lvelcalc();  
    //ошибка левого колеса  
    error = reqvel - Lvel;  
    //расчет интеграла  
    lerrint += error * delta / 1000;  
    //расчет компенсации  
    analogchange = error * p + lerrint * i;  
    //сама компенсация  
    Lmove(analogchange);  
    //аналогично для правого  
  }  
  ...  
}
```

# ПИ регулятор

```
void setup() {...}
void loop()
{
if ((delta = millis() - timer) >= 20)
{
//расчитываем скорость левого колеса
Lvelcalc();
//ошибка левого колеса
error = (-reqvel) - Lvel;
//расчет интеграла
lerrint += error * delta / 1000;
//расчет дифференциальной составляющей
deriv = (error- lerrorold) / (delta / 1000);
lerrorold = error;
//расчет компенсации
analogchange = error * p + lerrint * i - deriv * d;
//сама компенсация
Lmove(analogchange);
//аналогично для правого
...
}
```

# ПИД регулятор

# Навигация роботов



- Датчики глубины
- Датчики линий
- Одометрия
- IMU
- Сонары
- Камеры