# Реализация фильтра Калмана в ультразвуковом дальномере на основе отладочной платы STM32VLDiscovery

Содержание

[Постановка задачи 3](#_Toc483905839)

[Ход выполнения работы 3](#_Toc483905840)

[Поиск информации по алгоритму 3](#_Toc483905841)

[Определение модели процесса 4](#_Toc483905842)

[Определение сглаживающих свойств 4](#_Toc483905843)

[Эксперименты с Arduino 6](#_Toc483905844)

[Упрощенный фильтр Калмана с константным K 6](#_Toc483905845)

[Фильтр Калмана для одномерного случая 9](#_Toc483905846)

[Код взаимодействия Arduino Uno с ПК 11](#_Toc483905847)

[Реализация на STM32 по данным с дальномера 11](#_Toc483905848)

[Приложение А. Информация по ультразвуковому дальномеру HC-SR04 26](#_Toc483905849)

[Распиновка 26](#_Toc483905850)

[Характеристики 26](#_Toc483905851)

[Приложение Б. Характеристики Arduino UNO 27](#_Toc483905852)

# Постановка задачи

Реализовать фильтрацию алгоритмом Калмана данных, поступающих с дальномера на основе платы семейства STM32VL. Отфильтрованные данные и данные с датчиков должны передаваться на ПК, где должна быть реализована визуализирующая их программа.

## Ход выполнения работы

## Поиск информации по алгоритму

Я начал работу с изучения алгоритма и понимания принципов его работы.

В процессе поисков, мне удалось найти следующие ссылки:

1) <https://habrahabr.ru/post/166693/>   
Здесь имеется более-менее развернутое описание фильтра Калмана.  
Отсюда мы узнаем, что фильтр Калмана позволяет использовать при фильтрации информацию о физике процесса (по какому закону движется тело), управляющие сигналы (приказ роботу ехать быстрее, медленнее), а также данные с датчиков.  
Также в этой статье говорится, что фильтр Калмана применим для фильтрации любых видов сигналов. В статье есть также слабое упоминание о расширении фильтра Калмана на многомерный случай.  
  
Наш случай – случай одной переменной. Также у нас нет информации о законе движения тела (мы не знаем, как именно мы будем двигать наш дальномер). А также мы не располагаем информацией об управляющих сигналах, то для нашего случая можно упростить математику фильтра.

Автор статьи предлагает грубое приближение фильтр Калмана, которое использует константный коэффициент Калмана.

https://habrastorage.org/files/97c/ebf/bf0/97cebfbf02ae499984d9b245699ebc69.latex

Рисунок 1. Урощенный фильтр Калмана для 1 переменной. Источник <https://habrahabr.ru/post/166693/>

Суть коэффициента в том, что он выступает неким коэффициентом «доверия» показаниям с сенсора.

Автор обосновывает это решение отчасти тем, что коэффициент K с шагом итерации k стабилизируется к определенному значению .

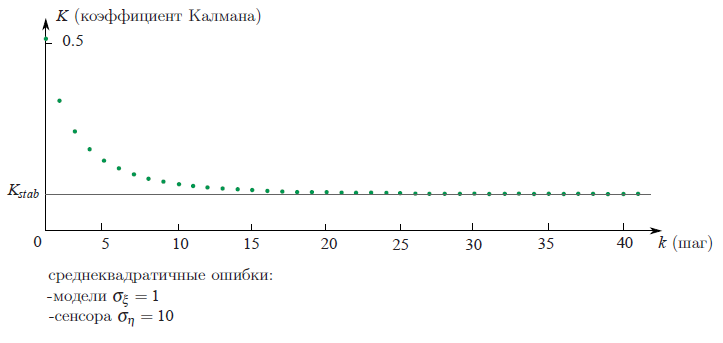


Рисунок 2. График изменения коэффициента K. Источник <https://habrahabr.ru/post/166693/>

2) <https://habrahabr.ru/post/140274/>

Данная статья пытается осветить многомерный случай для фильтра Калмана.

Здесь же представлен алгоритм для одномерного случая.

И развита такая идея: «Т.к. управляющих воздействий в нашей модели нет, то примем соответствующий коэффициент равным 0. Аналогично для динамики системы коэффициент принимается равным 1 (т.е. по нашему закону последующее состояние системы будет аналогично предыдущему).» Это позволяет использовать обобщённый алгоритм в нашей ситуации.

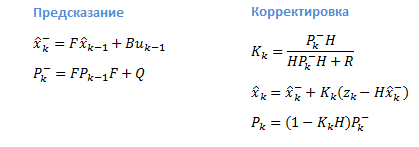


Рисунок 3. Фильтр Калмана для одномерного случая. Источник <https://habrahabr.ru/post/140274/>

### Определение модели процесса

**F**— переменная описывающая динамику системы, Для простоты примем эту переменную равную 1 (то есть мы указываем, что предсказываемое значение будет равно предыдущему состоянию).  
  
**B**— переменная определяющая применение управляющего воздействия. Так как управляющих воздействий в нашей модели нет (нет информации о них), то принимаем B = 0.  
  
**H**— матрица определяющая отношение между измерениями и состоянием системы, пока без объяснений примем эту переменную также равную 1.

### Определение сглаживающих свойств

**R**— ошибка измерения может быть определена испытанием измерительных приборов и определением погрешности их измерения.  
  
**Q**— определение шума процесса является более сложной задачей, так как требуется определить дисперсию процесса, что не всегда возможно. В любом случае, можно подобрать этот параметр для обеспечения требуемого уровня фильтрации.

3) <http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/>

По этой ссылке очень интересная англоязычная статья о фильтре Калмана для двумерного случая с большим количеством иллюстраций.

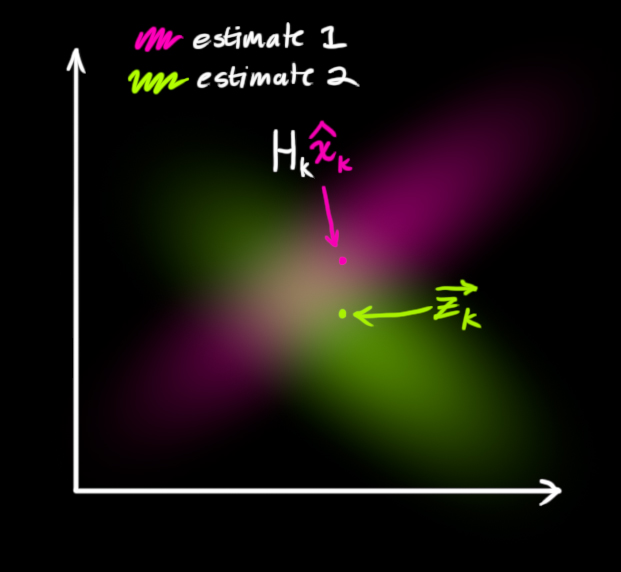


Рисунок 4. Фильтр Калмана для одномерного случая. Источник <https://habrahabr.ru/post/140274/>

Как я понял из статьи, суть фильтра в том, чтобы совместить информацию о законе изменения величины и показания сенсоров, принимая в учет то, что они будут зашумлёнными ошибками.

Причем, оба источника представляются в виде нормально распределенных случайных величин.

## Эксперименты с Arduino

На скорую руку была собрана схема, фильтрующая данные, получаемые с импровизированного датчика – фоторезистора (изменяет сопротивление в зависимости от освещения).

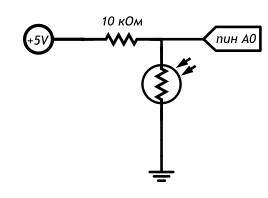


Рисунок 5. Принципиальная схема

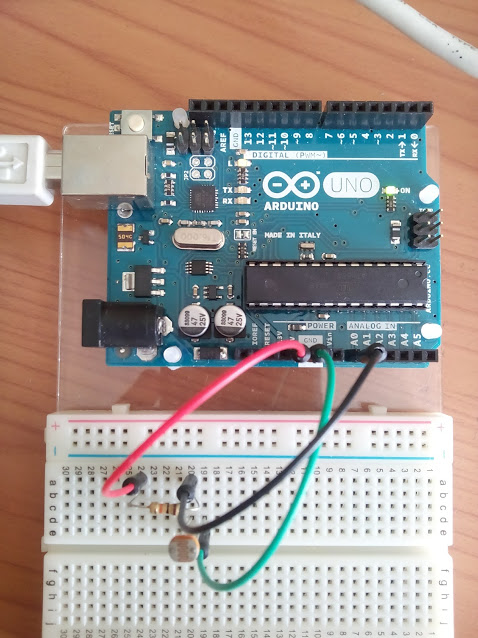


Рисунок 6. Собранная схема

### Упрощенный фильтр Калмана с константным K

#### Алгоритм

https://habrastorage.org/files/97c/ebf/bf0/97cebfbf02ae499984d9b245699ebc69.latex

#### Код

#**include** "Arduino.h"

*// Stupid kalman filter*

*// https://habrahabr.ru/post/166693/ (last paragraph)*

class StupidKalman1D {

public:

StupidKalman1D(float kalman\_coef = 0.25) {

kalman\_coefficient = kalman\_coef;

}

void SetState(float x0) {

x\_current = x0;

}

float Correct(float sensor\_data)

{

x\_current = kalman\_coefficient \* sensor\_data + (1 - kalman\_coefficient) \* x\_previous;

x\_previous = x\_current;

return x\_current;

}

float x\_current;

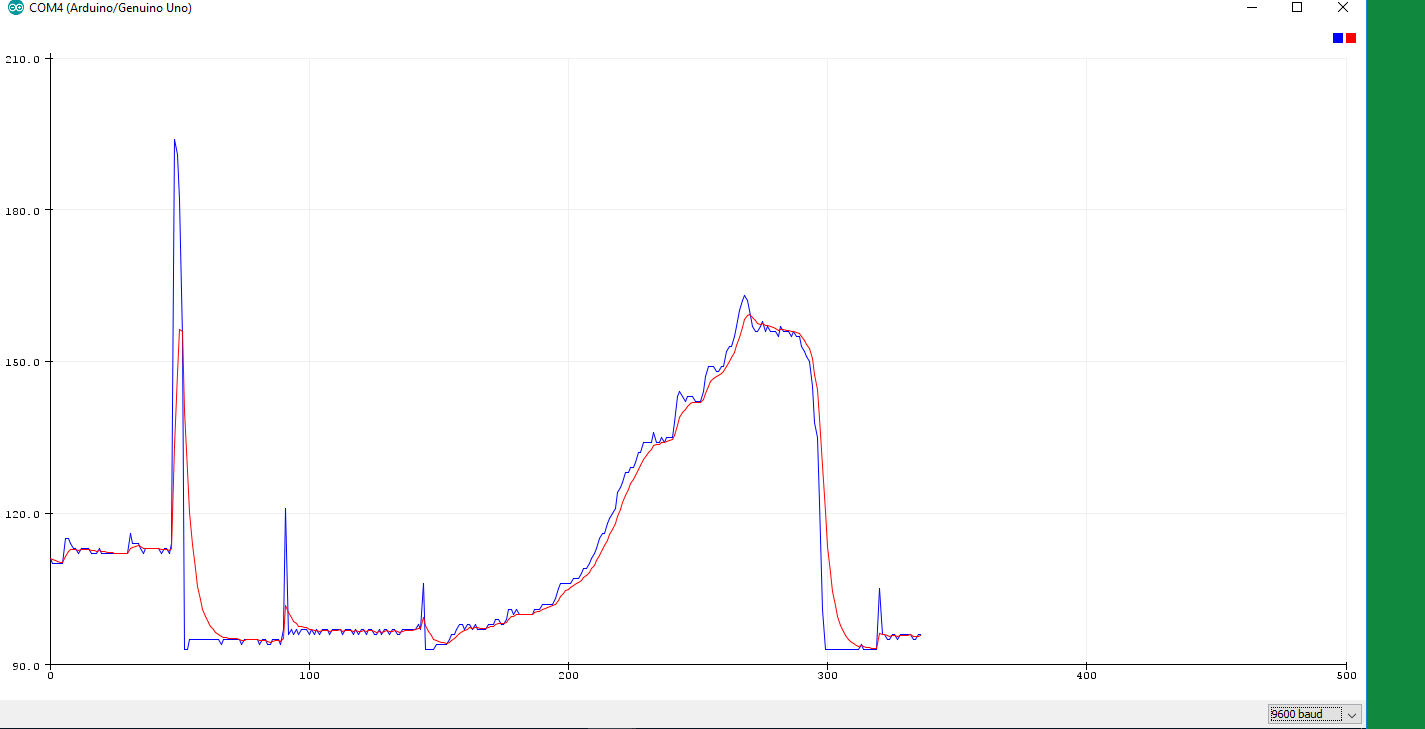
float x\_previous;

float kalman\_coefficient;

};

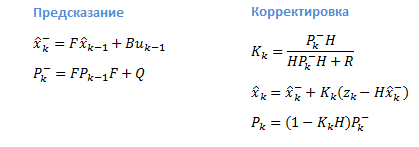
#### Результаты

Отфильтрованные значения отображаются красным цветом, сырые данные с датчика – синим.



### Фильтр Калмана для одномерного случая

#### Алгоритм



#### Код

#**include** "Arduino.h"

*// Clever kalman filter*

*// https://habrahabr.ru/post/140274/*

class CleverKalman1D {

public:

CleverKalman1D(float q, float r, float f = 1.0, float h = 1.0) {

Q = q;

R = r;

F = f;

H = h;

}

void SetState(float state, float covariance) {

State = state;

Covariance = covariance;

}

float Correct(float data)

{

//**time** update - prediction

X0 = F\*State;

P0 = F\*Covariance\*F + Q;

//measurement update - correction

float K = H\*P0/(H\*P0\*H + R);

State = X0 + K\*(data - H\*X0);

Covariance = (1 - K\*H)\*P0;

return State;

}

float X0,

P0;

float F,

Q,

H,

R;

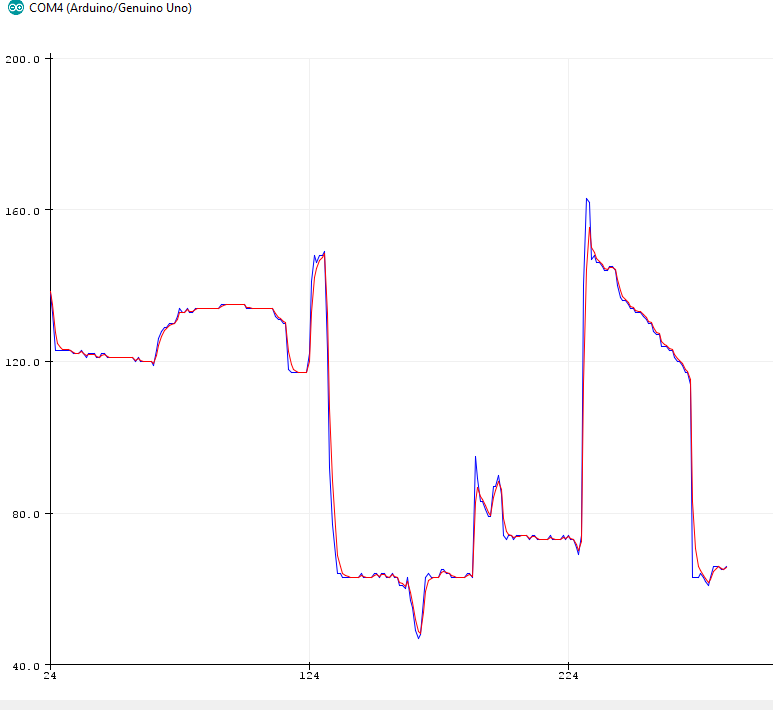
float State,

Covariance;

};

#### Результаты

Можно заметить, что этот вариант, в отличие от предыдущего, обладает более резкой реакцией на пики и спады. Здесь еще можно поиграться параметрами, т.к. на рисунке ниже представлены результаты работы алгоритма с параметрами, рекомендованными автором статьи.



### Код взаимодействия Arduino Uno с ПК

#define LDR\_PIN A0

#include "clever\_kalman.h"

#include "stupid\_kalman.h"

StupidKalman1D stupid\_kalman;

CleverKalman1D clever\_kalman(1, 1); *// play with these values*

void setup() {

Serial.begin(9600);

// Initialize both filters

float sensor\_first\_data = analogRead(LDR\_PIN);

stupid\_kalman.SetState(sensor\_first\_data);

clever\_kalman.SetState(sensor\_first\_data, 0.1);

}

String message;

float sensor\_data;

void loop() {

sensor\_data = analogRead(LDR\_PIN);

// map(val, 0, 1023, 0, 100);

float x\_stupid\_kalman = stupid\_kalman.Correct(sensor\_data);

float x\_clever\_kalman = clever\_kalman.Correct(sensor\_data);

// Write to serial port plotter

// $%d %d %d;

Serial.write('$');

Serial.print(int(sensor\_data));

Serial.write(' ');

Serial.print(int(x\_clever\_kalman));

Serial.write(';');

Serial.write('\n');

// Write for Arduino Plotter

// %f %f %f ... \n

Serial.print(sensor\_data);

Serial.print(' ');

// Serial.print(x\_stupid\_kalman);

// Serial.print(' ');

Serial.print(x\_clever\_kalman);

Serial.write('\n');

delay(500);

}

## Реализация на STM32 по данным с дальномера (работоспособность не проверена)

**#include** "stm32f10x.h"

**#include** "stm32f10x\_gpio.h"

**#include** "stm32f10x\_rcc.h"

**#include** "stm32f10x\_usart.h"

**#define** TRIG\_PIN GPIO\_Pin\_3

**#define** ECHO\_PIN GPIO\_Pin\_4

// Good GPIO example here

// http://we.easyelectronics.ru/STM32/prakticheskiy-kurs-stm32-urok-1---gpio-porty-vvoda-vyvoda.html

**void** **InitRangemeterPins**(){

GPIO\_InitTypeDef GPIO\_InitStructure; // should it be in outer scope?

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOC, *ENABLE*);

/\* Configure the pins \*/

// PIN3 - trig

// PIN4 - echo

GPIO\_InitStructure.GPIO\_Pin = TRIG\_PIN|ECHO\_PIN;

GPIO\_InitStructure.GPIO\_Mode = *GPIO\_Mode\_Out\_PP*;

GPIO\_InitStructure.GPIO\_Speed = *GPIO\_Speed\_50MHz*;

GPIO\_Init(GPIOC, &GPIO\_InitStructure);

}

// Good tutorial here

// http://we.easyelectronics.ru/GYUR22/prostoy-start-stm32-taktirovanie-i-zaderzhka.html

// Good code here

// https://stackoverflow.com/questions/21823197/generate-delay-in-stm32-use-timer

**void** **InitTimer**(){

TIM\_TimeBaseInitTypeDef Tim5;

TIM\_DeInit(TIM5);

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM5,*ENABLE*);

Tim5.TIM\_Period=1;

Tim5.TIM\_Prescaler=80-1;

Tim5.TIM\_ClockDivision=1;

Tim5.TIM\_CounterMode=TIM\_CounterMode\_Down;

TIM\_TimeBaseInit(TIM5,&Tim5);

}

**void** **msDelay**(u16 msTime){

u16 counter=msTime;

TIM\_Cmd(TIM5,*ENABLE*);

TIM\_SetCounter(TIM5,counter);

**while**(counter>1)

{

counter=TIM\_GetCounter(TIM5);

}

TIM\_Cmd(TIM5,*DISABLE*);

}

// This example from documentation

// http://microtechnics.ru/stm32-uchebnyj-kurs-usart/

// We need use timers for 10 and 50 milliseconds

**void** **SetupClock**()

{

RCC\_DeInit (); /\* RCC system reset(for debug purpose)\*/

RCC\_HSEConfig (RCC\_HSE\_ON); /\* Enable HSE \*/

/\* Wait till HSE is ready \*/

**while** (RCC\_GetFlagStatus(RCC\_FLAG\_HSERDY) == *RESET*);

RCC\_HCLKConfig (RCC\_SYSCLK\_Div1); /\* HCLK = SYSCLK \*/

RCC\_PCLK2Config (RCC\_HCLK\_Div1); /\* PCLK2 = HCLK \*/

RCC\_PCLK1Config (RCC\_HCLK\_Div2); /\* PCLK1 = HCLK/2 \*/

RCC\_ADCCLKConfig (RCC\_PCLK2\_Div4); /\* ADCCLK = PCLK2/4 \*/

/\* PLLCLK = 8MHz \* 9 = 72 MHz \*/

RCC\_PLLConfig (0x00010000, RCC\_PLLMul\_9);

RCC\_PLLCmd (*ENABLE*); /\* Enable PLL \*/

/\* Wait till PLL is ready \*/

**while** (RCC\_GetFlagStatus(RCC\_FLAG\_PLLRDY) == *RESET*);

/\* Select PLL as system clock source \*/

RCC\_SYSCLKConfig (RCC\_SYSCLKSource\_PLLCLK);

/\* Wait till PLL is used as system clock source \*/

**while** (RCC\_GetSYSCLKSource() != 0x08);

/\* Enable USART1 and GPIOA clock \*/

RCC\_APB2PeriphClockCmd (RCC\_APB2Periph\_USART1 | RCC\_APB2Periph\_GPIOA, *ENABLE*);

}

**void** **SetupUSART**()

{

GPIO\_InitTypeDef GPIO\_InitStructure;

USART\_InitTypeDef USART\_InitStructure;

/\* Enable GPIOA clock \*/

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, *ENABLE*);

/\* Configure USART1 Rx (PA10) as input floating \*/

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_10;

GPIO\_InitStructure.GPIO\_Mode = *GPIO\_Mode\_IN\_FLOATING*;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

/\* Configure USART1 Tx (PA9) as alternate function push-pull \*/

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_9;

GPIO\_InitStructure.GPIO\_Speed = *GPIO\_Speed\_50MHz*;

GPIO\_InitStructure.GPIO\_Mode = *GPIO\_Mode\_AF\_PP*;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

/\* USART1 configured as follow:

- BaudRate = 115200 baud

- Word Length = 8 Bits

- One Stop Bit

- No parity

- Hardware flow control disabled (RTS and CTS signals)

- Receive and transmit enabled

- USART Clock disabled

- USART CPOL: Clock is active low

- USART CPHA: Data is captured on the middle

- USART LastBit: The clock pulse of the last data bit is not output to

the SCLK pin

\*/

USART\_InitStructure.USART\_BaudRate = 115200;

USART\_InitStructure.USART\_WordLength = USART\_WordLength\_8b;

USART\_InitStructure.USART\_StopBits = USART\_StopBits\_1;

USART\_InitStructure.USART\_Parity = USART\_Parity\_No ;

USART\_InitStructure.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;

USART\_InitStructure.USART\_Mode = USART\_Mode\_Rx | USART\_Mode\_Tx;

USART\_Init(USART1, &USART\_InitStructure);

USART\_Cmd(USART1, *ENABLE*);

}

**int** **GetDataFromRangemeter**(){

// Подать 1 на Trig

GPIO\_WriteBit(LED\_PORT,TRIG\_PIN,*Bit\_SET*);

// Подождать 10 мс

msDelay(10);

// Подать 0 на Trig

GPIO\_WriteBit(LED\_PORT,TRIG\_PIN,*Bit\_RESET*);

// Поймать момент, когда на Echo 0->1

**while** (!GPIO\_ReadInputDataBit(GPIOC, ECHO\_PIN));

// Начать замер времени

**int** rangemeter\_time = 0;

TIM\_Cmd(TIM5,*ENABLE*);

TIM\_SetCounter(TIM5,counter);

// Поймать момент, когда на Echo 1->0

**while**(GPIO\_ReadInputDataBit(GPIOC, ECHO\_PIN))

rangemeter\_time=TIM\_GetCounter(TIM5);

// Завершить замер времени

TIM\_Cmd(TIM5,*DISABLE*);

// Вычислить расстояние

**int** sensor\_data = rangemeter\_time / 58;

**return** sensor\_data;

}

// Clever Kalman

**float** Q = 1; // Шум измерений

**float** R = 1; // Шум окружения

**float** F = 1; // Коэфициент между предыдущим и текущим значением

**float** H = 1; // Коэффициент между измеренным и реальным значениями

**float** State;

**float** Covariance;

**float** X0, P0;

**float** **CleverCalman\_Correct**(**float** data){

//time update - prediction

X0 = F\*State;

P0 = F\*Covariance\*F + Q;

//measurement update - correction

**float** K = H\*P0/(H\*P0\*H + R);

State = X0 + K\*(data - H\*X0);

Covariance = (1 - K\*H)\*P0;

**return** State;

}

// Stupid Kalman

**float** XCur, XPrev, KalmanCoef = 0.25;

**float** **StupidKalman\_Correct**(**float** data){

XCur = kalman\_coefficient \* data + (1 - KalmanCoef) \* XPrev;

XPrev = XCur;

**return** XCur;

}

**int** **main**(**void**)

{

SetupClock();

InitTimer();

InitRangemeterPins();

SetupUSART();

// Инициализация фильтров

sensor\_data = GetDataFromRangemeter();

// Stupid Kalman

XCur = sensor\_data;

// Clever Kalman

State = sensor\_data;

Covariance = sensor\_data;

**while**(1)

{

// Получить данные с дальномера

**int** sensor\_data = GetDataFromRangemeter();

// Фильтр Калмана -> фильтрованое значение

**float** filtered\_data = StupidKalman\_Correct(sensor\_data);

// Отослать оба значения по USART

// Формат передачи важен

USART\_SendData(USART1, sensor\_data);

USART\_SendData(USART1, filtered\_data);

// Подождать 50 мс. (для дальномера)

msDelay(50);

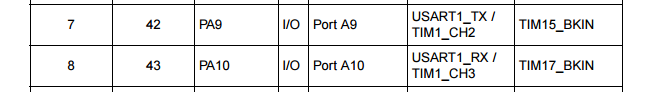
}

}

## Реализация на STM32 Discovery (F100RB)

### USART

Т.к. не STM32 Discovery нет эмуляции USARTа через USB, как на платах MiscoElectronika, пришлось воспользоваться пинами GPIO, на которых в альтернативном режиме можно было задействовать USART1.



О плате и распиновке вкратце можно почитать по ссылке <https://goo.gl/nD7zEK>

Так как у меня нет ничего, через что можно было бы подключить Tx /Rx к моему ноутбуку, я сделал «трансмиттер» USARTа на ПК с помощью Arduino UNO.

На первом программно эмулируемом USART соединении Arduino принимает данные от STM32. А по второму соединению отправляет их на ноутбук.

#### Код посредника USART для Arduino

#**include** <SoftwareSerial.h>

SoftwareSerial **mySerial**(10, 11); *// RX, TX*

**void** **setup**() {

Serial.**begin**(9600);

mySerial.**begin**(9600);

}

**void** **loop**() {

if (mySerial.**available**()) {

Serial.**write**(mySerial.**read**());

}

}

### Код STM32

### Главная функция

**#include** "Kalman.h"

**#include** "Usart.h"

**#include** "ADC.h"

KalmanFilter filter;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name : main

\* Description : Read data from ADC, filter it and send via USART1 to PC

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **main**(**void**)

{

/\* Enable USART1 and GPIOA clock \*/

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1 | RCC\_APB2Periph\_GPIOA, *ENABLE*);

/\* Configure the GPIOs \*/

GPIO\_Configuration();

/\* Configure the USART1 \*/

USART\_Configuration();

adc\_init();

KalmanInit(&filter, get\_adc\_value(), 0.1, 1, 1, 1, 1);

**while**(1)

{

uint16\_t sensor\_data = get\_adc\_value();

uint16\_t filtered\_data = KalmanCorrect(&filter, sensor\_data);

USART\_SendNumber(sensor\_data);

USART\_SendChar(' ');

USART\_SendNumber(filtered\_data);

USART\_SendChar(' ');

USART\_SendChar('\n');

**for** (**int** i = 0; i < 100000; i++); // Delay

}

}

### ADC.h

**#ifndef** Adc

**#define** Adc

**#include** "stm32f10x\_adc.h"

**#include** "stm32f10x\_rcc.h"

// Инициализация ADC1 на 1 ножке PA1

**void** **adc\_init**()

{

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, *ENABLE*);

// настройки ADC

ADC\_InitTypeDef ADC\_InitStructure;

ADC\_StructInit(&ADC\_InitStructure);

ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent; // режим работы - одиночный, независимый

ADC\_InitStructure.ADC\_ScanConvMode = *DISABLE*; // не сканировать каналы, просто измерить один канал

ADC\_InitStructure.ADC\_ContinuousConvMode = *DISABLE*; // однократное измерение

ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None; // без внешнего триггера

ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right; //выравнивание битов результат - прижать вправо

ADC\_InitStructure.ADC\_NbrOfChannel = 1; //количество каналов - одна штука

ADC\_Init(ADC1, &ADC\_InitStructure);

ADC\_Cmd(ADC1, *ENABLE*);

// настройка канала

ADC\_RegularChannelConfig(ADC1, ADC\_Channel\_1, 1, ADC\_SampleTime\_55Cycles5);

// калибровка АЦП

ADC\_ResetCalibration(ADC1);

**while** (ADC\_GetResetCalibrationStatus(ADC1));

ADC\_StartCalibration(ADC1);

**while** (ADC\_GetCalibrationStatus(ADC1));

}

// получение значения с ADC1

uint16\_t **get\_adc\_value**()

{

ADC\_SoftwareStartConvCmd(ADC1, *ENABLE*);

**while**(ADC\_GetFlagStatus(ADC1, ADC\_FLAG\_EOC) == *RESET*);

**return** ADC\_GetConversionValue(ADC1);

}

**#endif**

### Usart.h

**#ifndef** UsartKalman

**#define** UsartKalman

**#include** "stm32f10x\_usart.h"

**#include** "stm32f10x\_rcc.h"

**#include** "stm32f10x\_gpio.h"

**#include** "stdlib.h"

**#include** "misc.h"

**void** **USART\_SendNumber**(uint16\_t number){

**char** buffer[10];

**while**(USART\_GetFlagStatus(USART1, USART\_FLAG\_TXE) == *RESET*); // Wait for Empty

**itoa**(number, buffer, 10);

UARTSend(buffer, **sizeof**(buffer));

}

**void** **USART\_SendChar**(**char** ch){

**char** buffer[10];

**while**(USART\_GetFlagStatus(USART1, USART\_FLAG\_TXE) == *RESET*); // Wait for Empty

USART\_SendData(USART1, ch);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name : GPIO\_Configuration

\* Description : Configures the different GPIO ports.

\* Input : None

\* Output : None

\* Return : None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// PA9 - USART1 TX

// PA10 - USART1 RX

**void** **GPIO\_Configuration**(**void**)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

/\* Configure USART1 Tx (PA.09) as alternate function push-pull \*/

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_9;

GPIO\_InitStructure.GPIO\_Mode = *GPIO\_Mode\_AF\_PP*;

GPIO\_InitStructure.GPIO\_Speed = *GPIO\_Speed\_50MHz*;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

/\* Configure USART1 Rx (PA.10) as input floating \*/

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_10;

GPIO\_InitStructure.GPIO\_Mode = *GPIO\_Mode\_IN\_FLOATING*;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name : USART\_Configuration

\* Description : Configures the USART1.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **USART\_Configuration**(**void**)

{

USART\_InitTypeDef USART\_InitStructure;

/\* USART1 configuration --------------------------------------------------\*/

/\* USART1 configured as follow:

- BaudRate = 9600 baud

- Word Length = 8 Bits

- One Stop Bit

- No parity

- Hardware flow control disabled (RTS and CTS signals)

- Receive and transmit enabled

- USART Clock disabled

- USART CPOL: Clock is active low

- USART CPHA: Data is captured on the middle

- USART LastBit: The clock pulse of the last data bit is not output to

the SCLK pin

\*/

USART\_InitStructure.USART\_BaudRate = 9600;

USART\_InitStructure.USART\_WordLength = USART\_WordLength\_8b;

USART\_InitStructure.USART\_StopBits = USART\_StopBits\_1;

USART\_InitStructure.USART\_Parity = USART\_Parity\_No;

USART\_InitStructure.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;

USART\_InitStructure.USART\_Mode = USART\_Mode\_Rx | USART\_Mode\_Tx;

USART\_Init(USART1, &USART\_InitStructure);

/\* Enable USART1 \*/

USART\_Cmd(USART1, *ENABLE*);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name : UARTSend

\* Description : Send a string to the UART.

**void** **UARTSend**(**const** **unsigned** **char** \*pucBuffer, **unsigned** **long** ulCount)

{

// Loop while there are more characters to send.

**while**(ulCount--)

{

**if** (\*pucBuffer == '\0') **return**; // send until '\0'

USART\_SendData(USART1, \*pucBuffer++);

/\* Loop until the end of transmission \*/

**while**(USART\_GetFlagStatus(USART1, USART\_FLAG\_TC) == *RESET*)

{

}

}

}

**#endif**

### Kalman.h

**#ifndef** Kalman

**#define** Kalman

// Kalman Filter

// ============================================================================================================================

**typedef** **struct** {

**float** X0, P0;

**float** F, // переменная описывающая динамику системы (например скорость). Т.к. неизвестно, принимаем равнымм 1 (x\_next = x\_curr)

B, // переменная определяющая применение управляющего воздействия. Т.к. упр. воздействий нет, принимает 0

H, // матрица определяющая отношение между измерениями и состоянием системы

Q, // определение шума процесса является более сложной задачей, так как требуется определить дисперсию процесса, что не всегда возможно.

// В любом случае, можно подобрать этот параметр для обеспечения требуемого уровня фильтрации.

R; // ошибка измерения может быть определена испытанием измерительных приборов и определением погрешности их измерения.

**float** Xk; // Последнее скорректированное фильтром значение

**float** Pk; // Ошибка ковариации

**float** K;

}KalmanFilter;

**void** **KalmanInit**(KalmanFilter \* k\_f, **float** x0, **float** p0, **float** f, **float** h, **float** q, **float** r){

k\_f->X0 = x0;

k\_f->P0 = p0;

k\_f->F = f;

k\_f->H = h;

k\_f->Q = q;

k\_f->R = r;

}

**float** **KalmanCorrect**(KalmanFilter \* k\_f, **float** sensor\_data){

**float** F = k\_f->F, Xk = k\_f->Xk, Pk = k\_f->Pk, X0 = k\_f->X0, P0 = k\_f->P0, K = k\_f->K, H = k\_f->H, Q=k\_f->Q, R=k\_f->R;

// Clever Kalman

//time update - prediction

// k\_f->X0 = F\*Xk;

// k\_f->P0 = F\*Pk\*F + Q;

//

// //measurement update - correction

// k\_f->K = H\*P0/(H\*P0\*H + R);

// k\_f->Xk = X0 + K\*(sensor\_data - H\*X0);

// k\_f->Pk = (1 - K\*H)\*P0;

// Stupid Kalman

k\_f->K = 0.25;

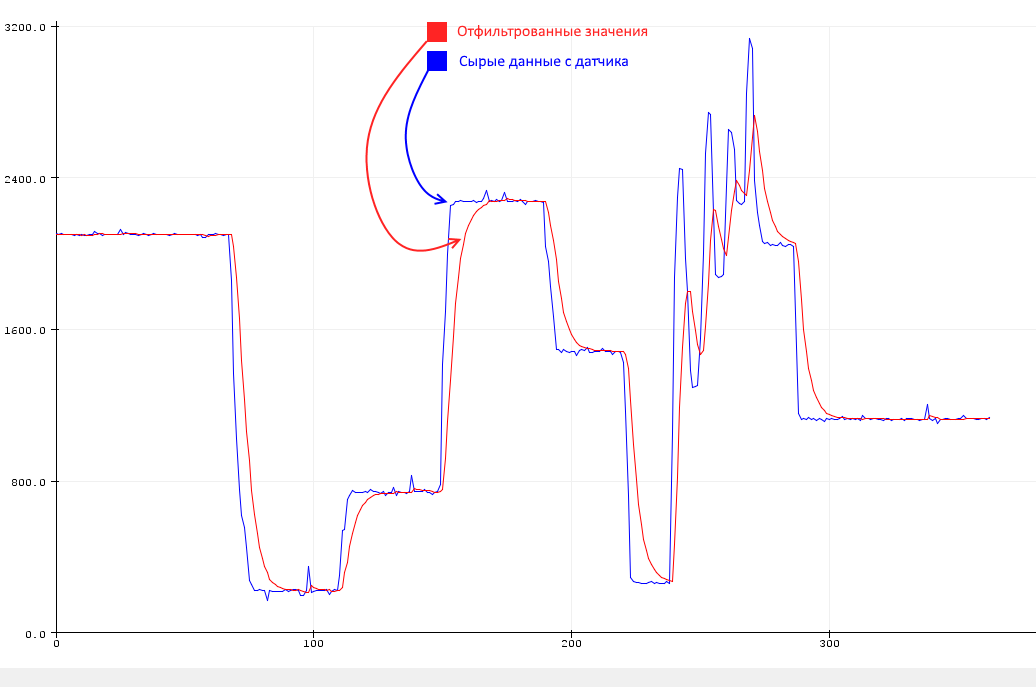
k\_f->Xk = K\*sensor\_data+(1-K)\*Xk;

**return** Xk;

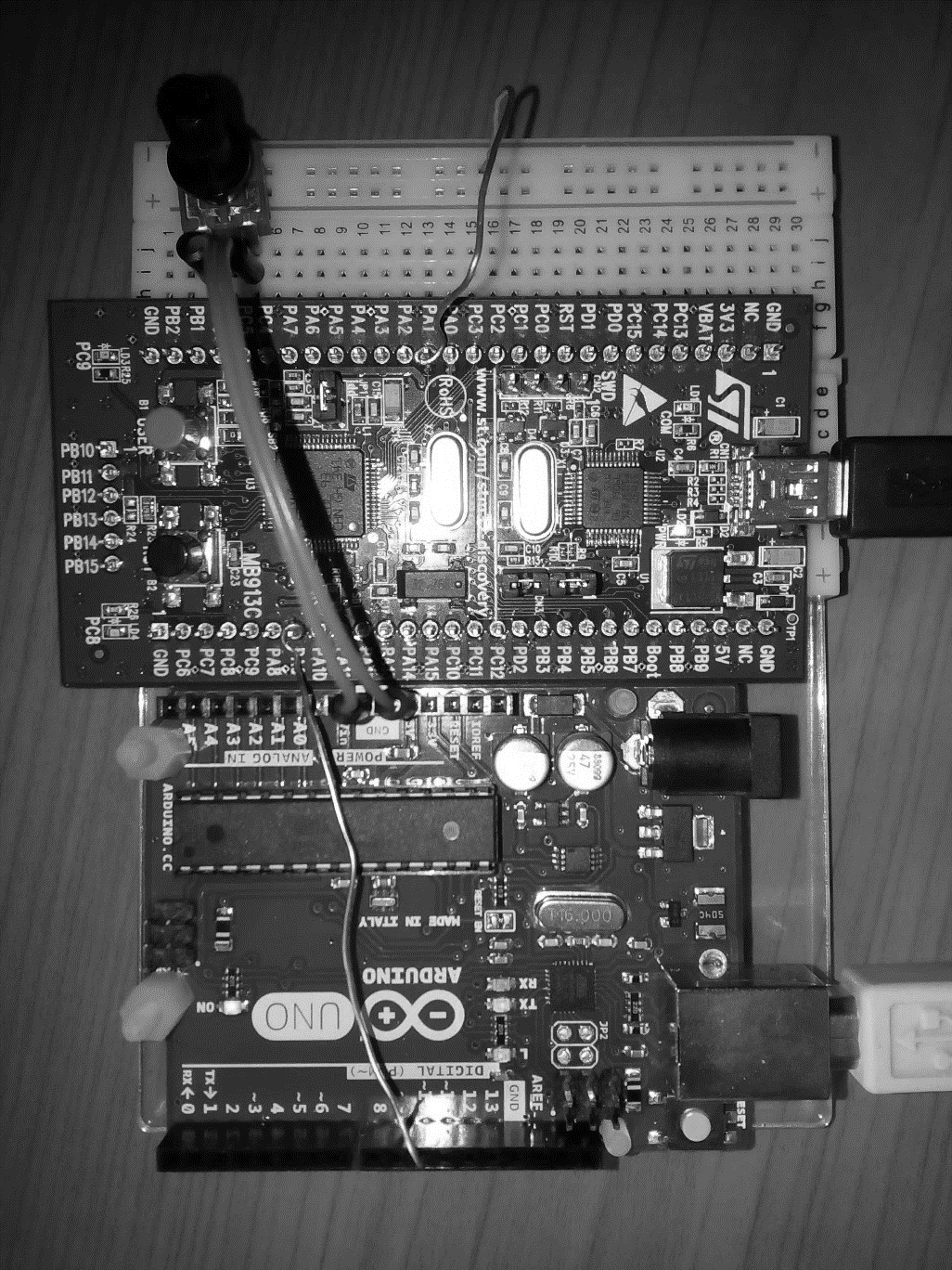
}

**#endif**

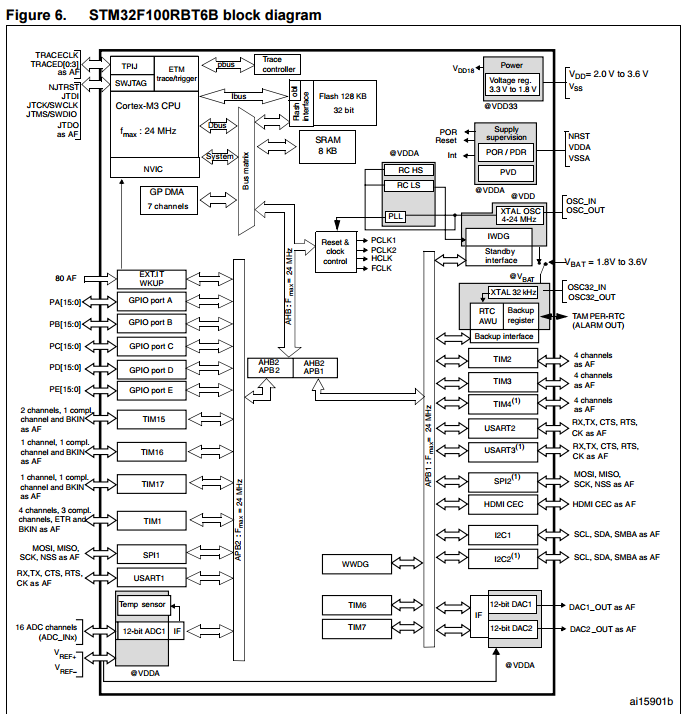
### Результаты фильтрации на STM32 Discovery



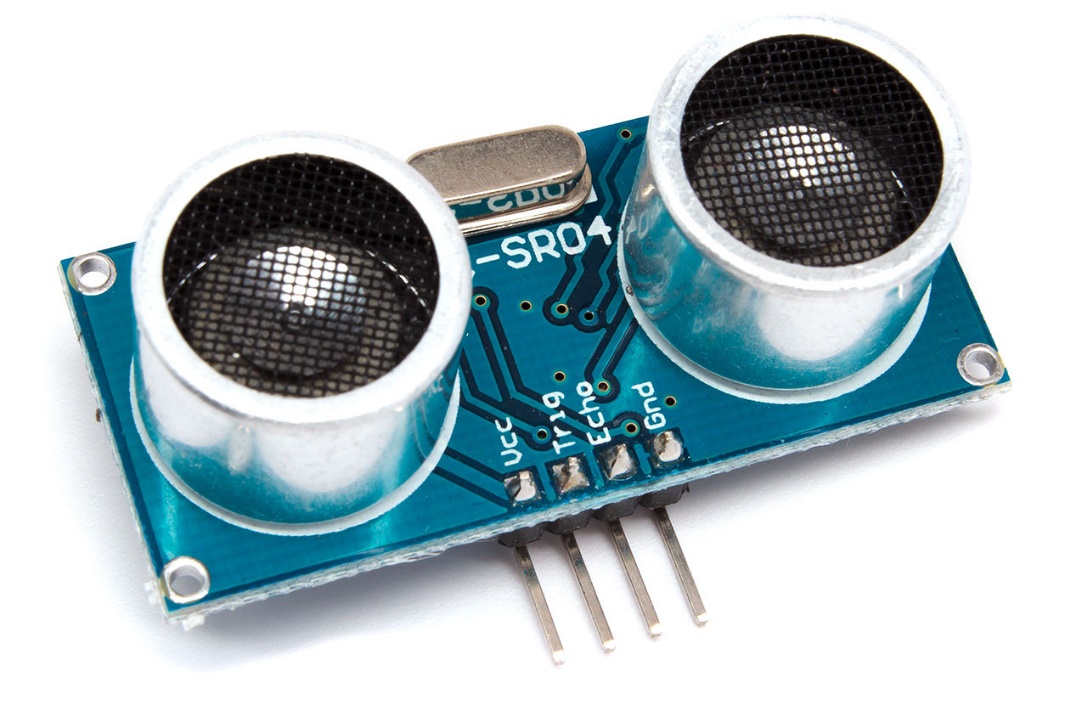
### Внешний вид схемы



# Приложение А. Схема STM32F100RB



## Приложение Б. Информация по ультразвуковому дальномеру HC-SR04



### Распиновка

Vcc — положительный контакт питания.

Trig — цифровой вход. Для запуска измерения необходимо подать на этот вход логическую единицу на 10 мкс. Следующее измерение рекомендуется выполнять не ранее чем через 50 мс.

Echo — цифровой выход. После завершения измерения, на этот выход будет подана логическая единица на время, пропорциональное расстоянию до объекта.

GND — отрицательный контакт питания.

### Характеристики

Напряжение питание: 5 В

Потребление в режиме тишины: 2 мА

Потребление при работе: 15 мА

Диапазон расстояний: 2–400 см

Эффективный угол наблюдения: 15°

Рабочий угол наблюдения: 30°

Размеры и диаграмма направленности

## Приложение В. Характеристики Arduino UNO

|  |  |
| --- | --- |
| Микроконтроллер | ATmega328 |
| Рабочее напряжение | 5 В |
| Входное напряжение (рекомендуемое) | 7-12 В |
| Входное напряжение (предельное) | 6-20 В |
| Цифровые Входы/Выходы | 14 (6 из которых могут использоваться как выходы [ШИМ](http://arduino.ru/Tutorial/PWM)) |
| Аналоговые входы | 6 |
| Постоянный ток через вход/выход | 40 мА |
| Постоянный ток для вывода 3.3 В | 50 мА |
| Флеш-память | 32 Кб (ATmega328) из которых 0.5 Кб используются для загрузчика |
| ОЗУ | 2 Кб (ATmega328) |
| EEPROM | 1 Кб (ATmega328) |
| Тактовая частота | 16 МГц |