Реализация фильтра Калмана в ультразвуковом дальномере на основе отладочной платы STM32VLDiscovery

Содержание

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.

$$x_{k+1}^{opt} = K_{stab} \cdot z_{k+1} + (1 - K_{stab}) \cdot x_k^{opt}$$

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 K_{stab}.

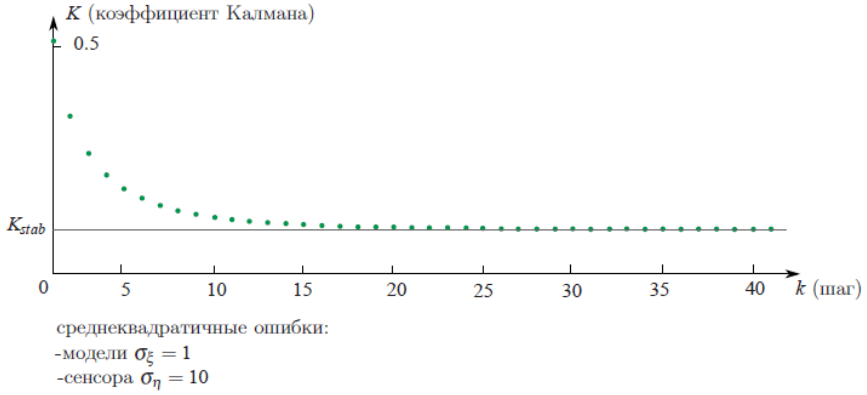


Рисунок 2. График изменения коэффициента K. Источник https://habrahabr.ru/post/166693/

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.

Предсказание

$$\hat{x}_k^- = F\hat{x}_{k-1} + Bu_{k-1}$$

$$P_k^- = FP_{k-1}F + Q$$

Корректировка

$$K_k = \frac{P_k^- H}{HP_k^- H + R}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

$$P_k = (1 - K_k H)P_k^-$$

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.

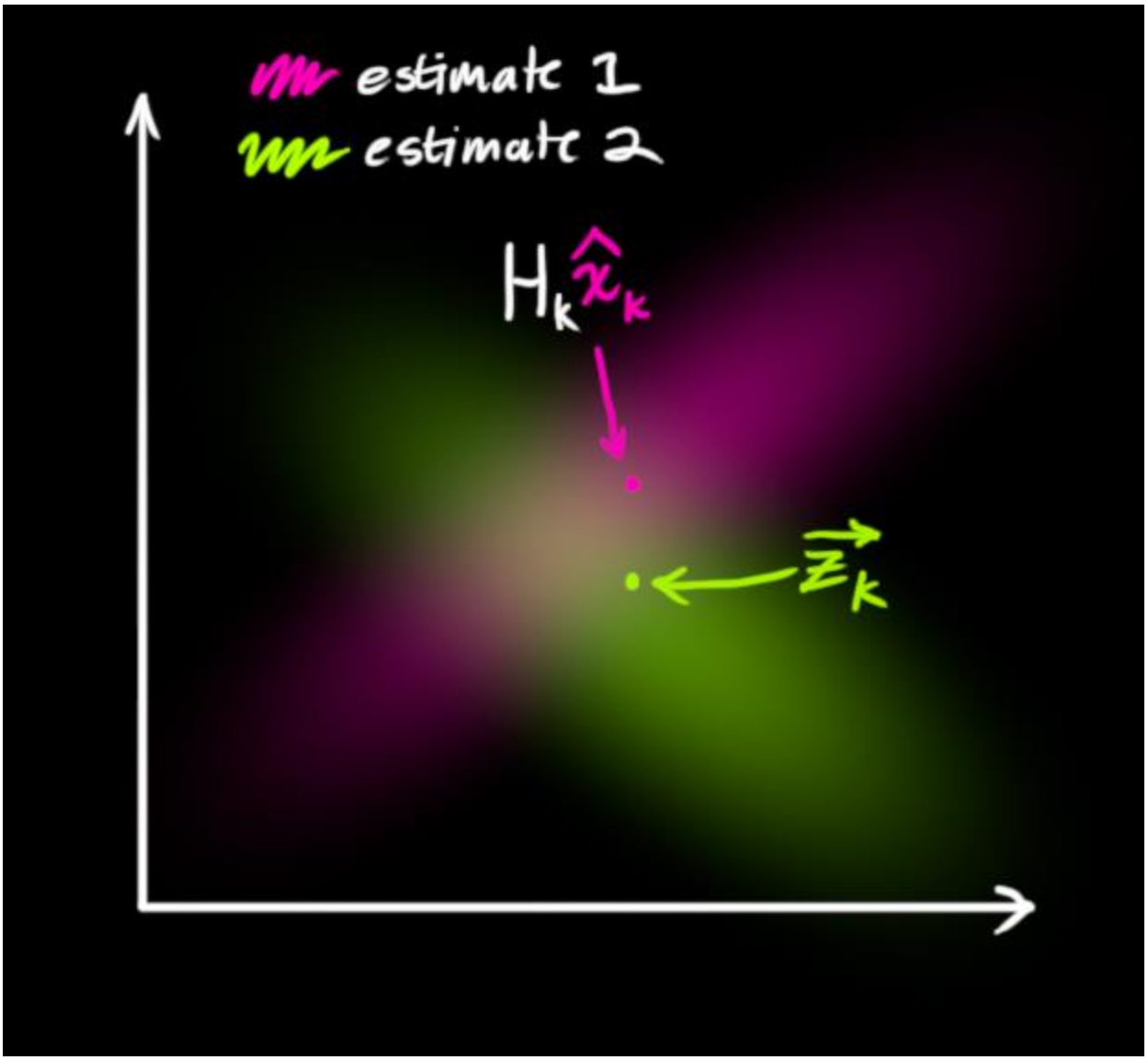


Рисунок 4. Фильтр Калмана для одномерного случая. Источник https://habrahabr.ru/post/140274/

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).

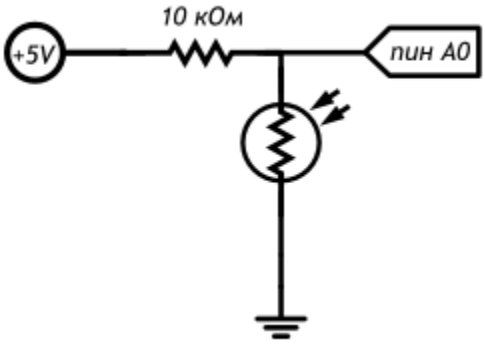


Рисунок 5. Принципиальная схема

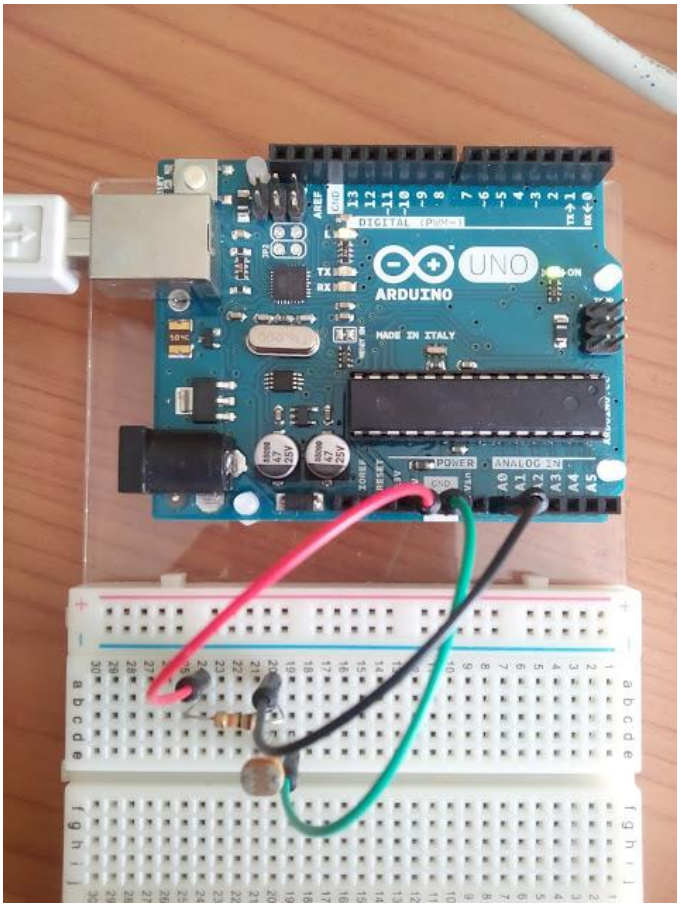


Рисунок 6. Собранная схема

Упрощенный фильтр Калмана с константным K

Алгоритм

$$x_{k+1}^{opt} = K_{stab} \cdot z_{k+1} + (1 - K_{stab}) \cdot x_k^{opt}$$

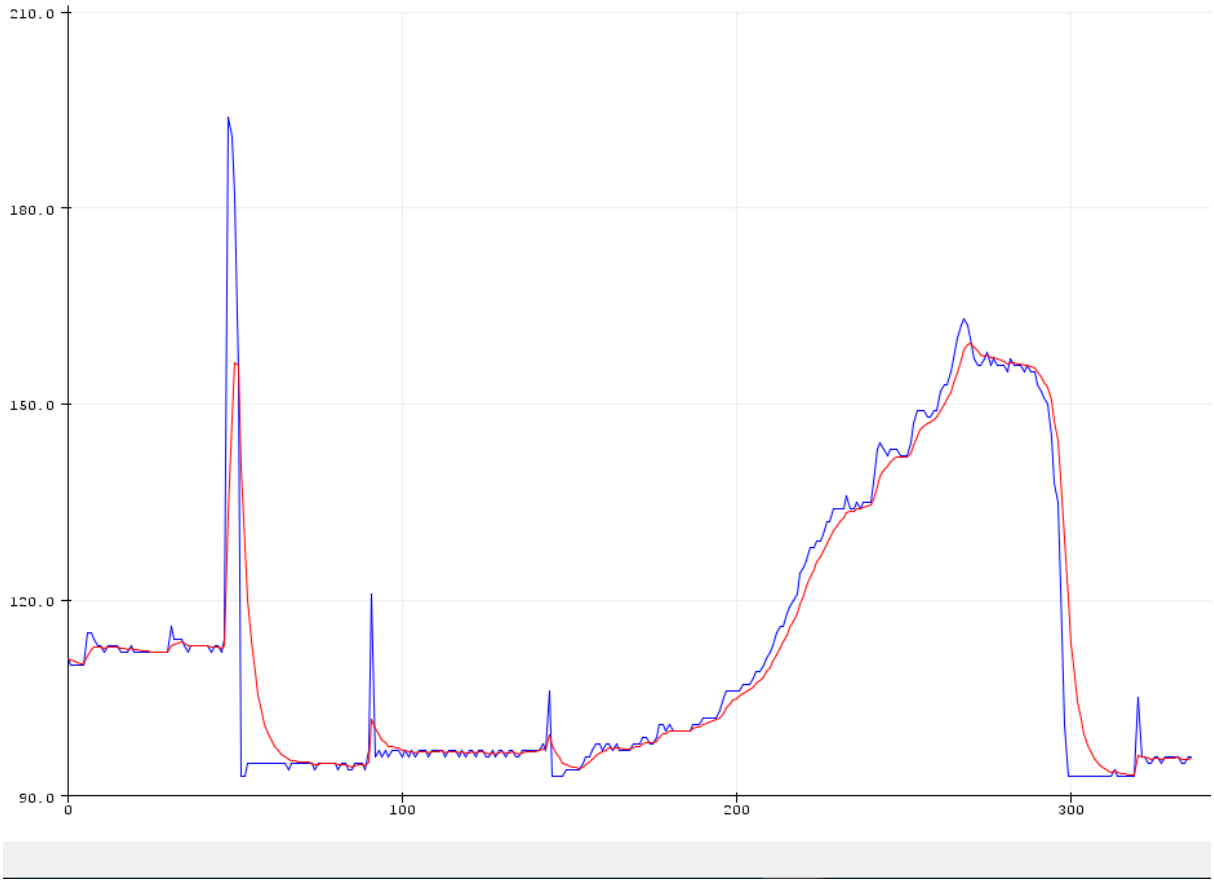
Код

```
#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.

COM4 (Arduino/Genuino Uno)



Фильтр Калмана для одномерного случая

Алгоритм

Предсказание

$$\hat{x}_k^- = F\hat{x}_{k-1} + Bu_{k-1}$$

$$P_k^- = FP_{k-1}F + Q$$

Корректировка

$$K_k = \frac{P_k^- H}{HP_k^- H + R}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

$$P_k = (1 - K_k H)P_k^-$$

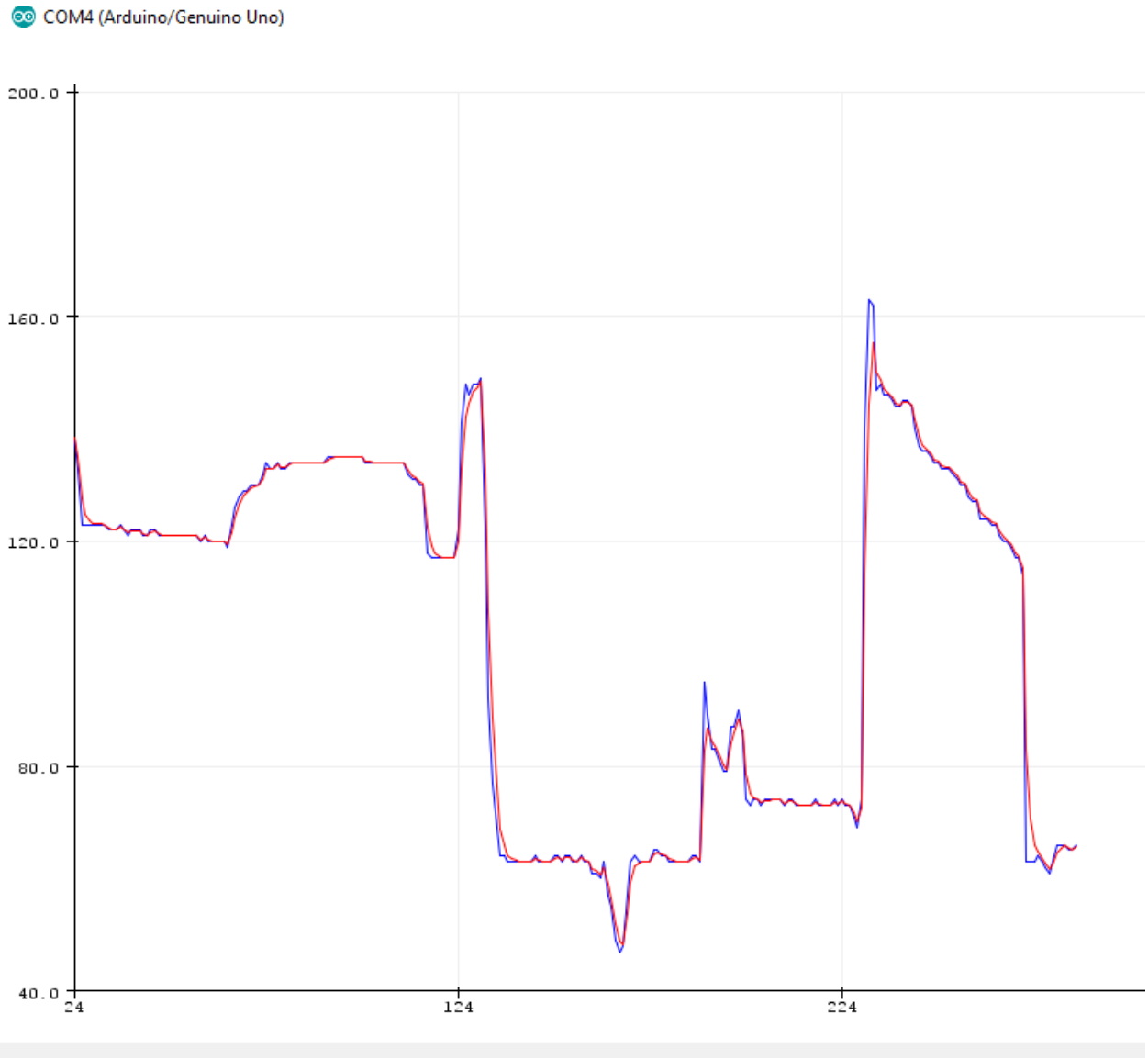
Код

```
#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;
}
```

```c
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 USARTa через USB, как на платах MiscoElectronika, пришлось воспользоваться пинами GPIO, на которых в альтернативном режиме можно было задействовать USART1.

7	42	PA9	I/O	Port A9	USART1_TX / TIM1_CH2	TIM15_BKIN
8	43	PA10	I/O	Port A10	USART1_RX / TIM1_CH3	TIM17_BKIN

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» USARTa 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

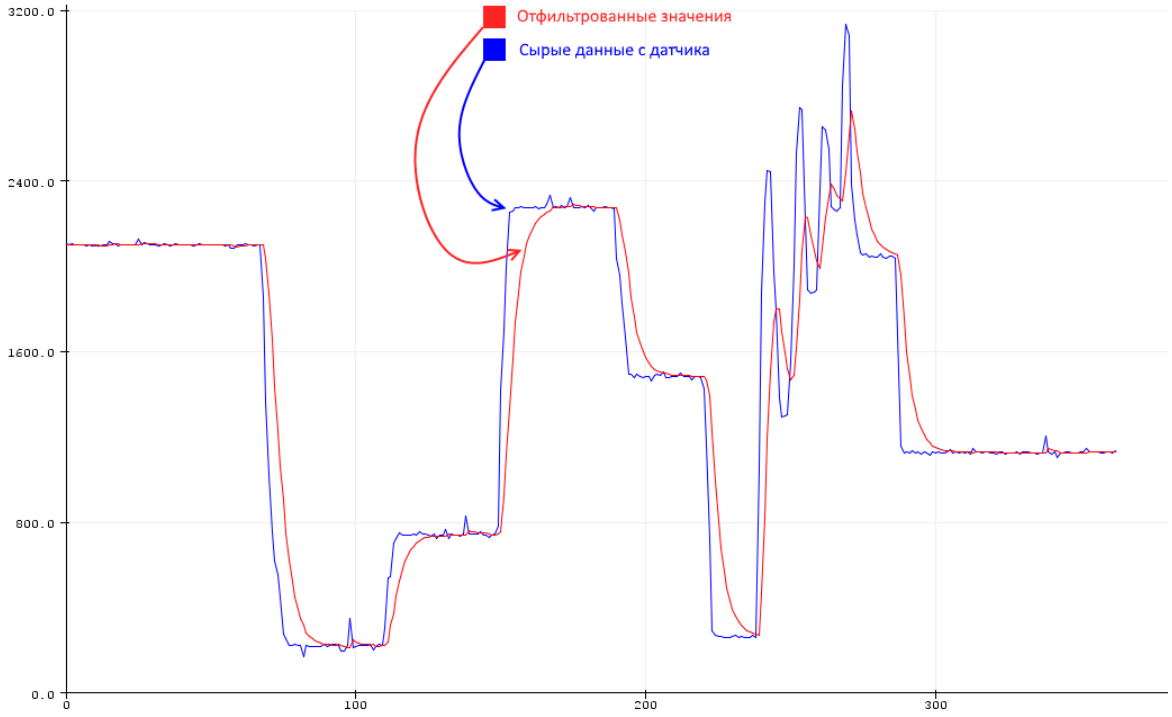
```
#ifndef UsartKalman
#define UsartKalman
#include "stm32f10x_usart.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stdlib.h"
#include "misc.h"


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);
}
```

```c
/*******************************************************************************
* 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
```

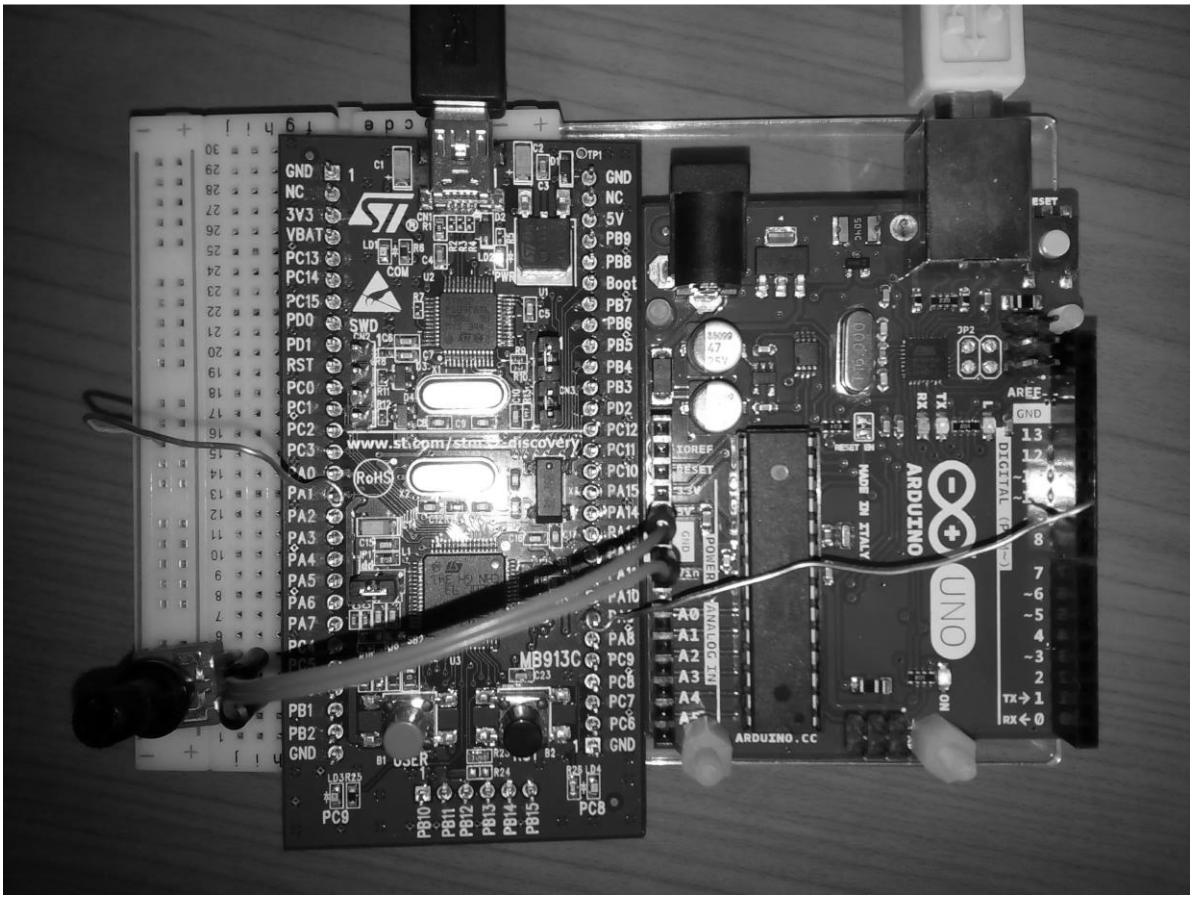
Kalman.h

```
#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

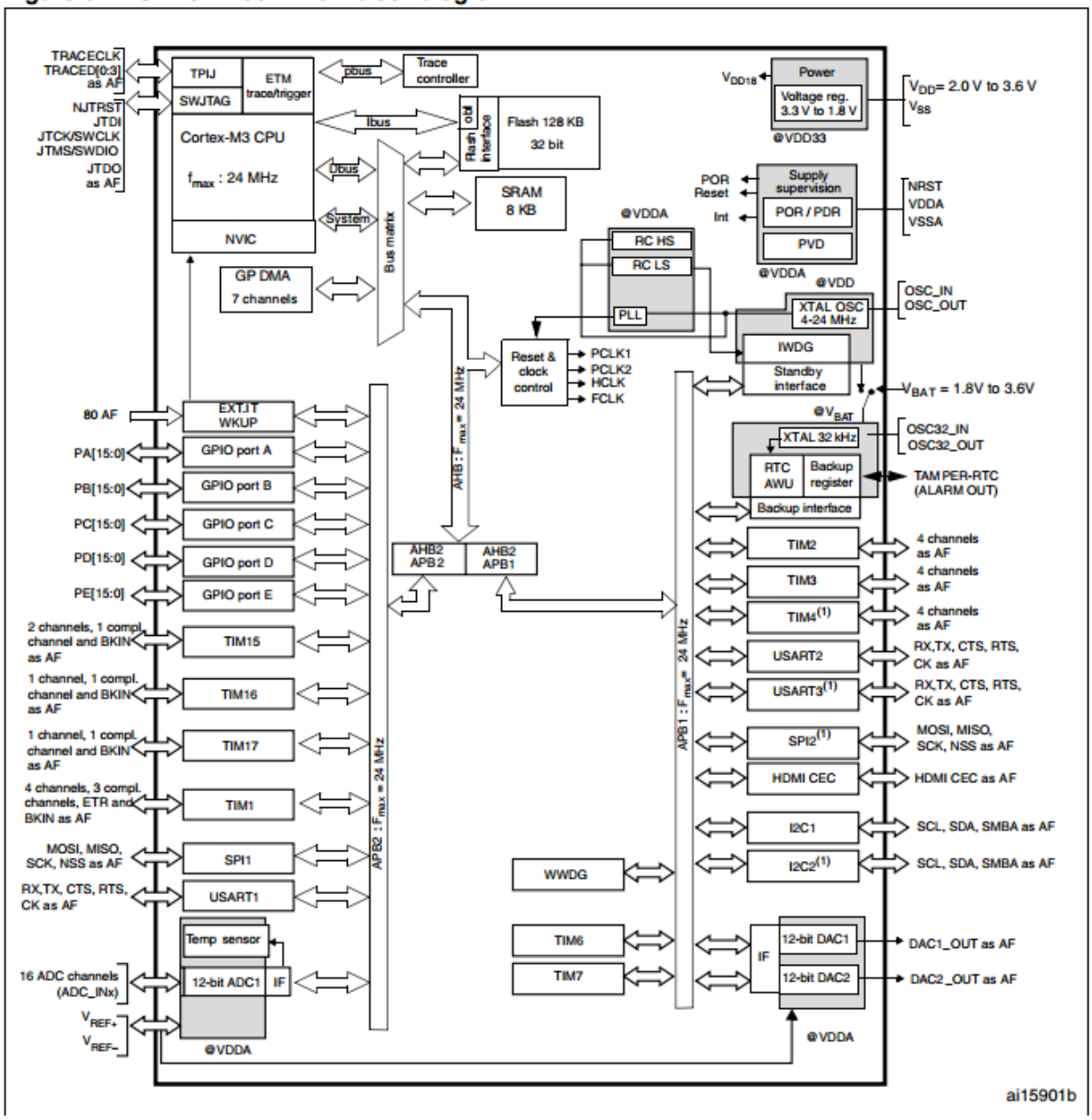


Внешний вид схемы

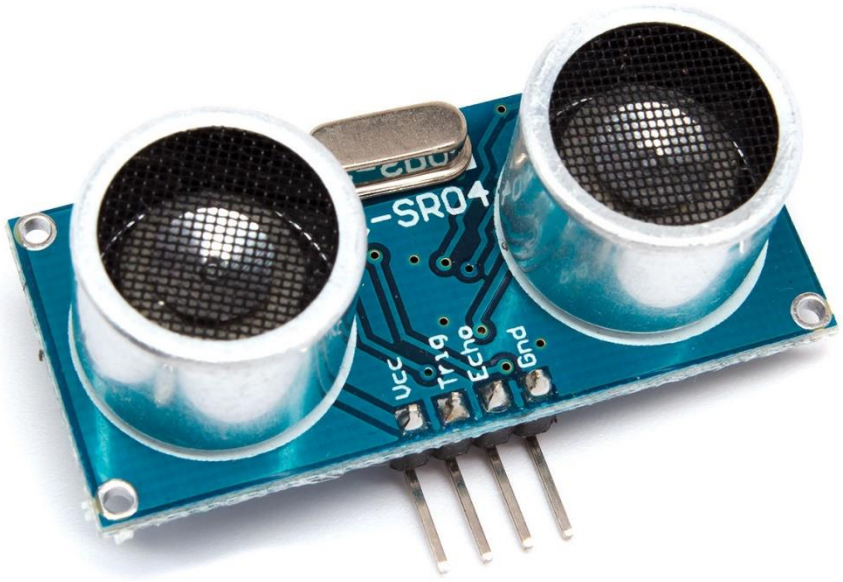


Приложение А. Схема STM32F100RB

Figure 6. STM32F100RBT6B block diagram



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)
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