11 种滤波算法程序大全（含源码）

算法爱好者 昨天

1、限幅滤波法（又称程序判断滤波法）

```
/*
A、名称：限幅滤波法（又称程序判断滤波法）
B、方法：
    根据经验判断，确定两次采样允许的最大偏差值（设为A），
    每次检测到新值时判断：
    如果本次值与上次值之差<=A，则本次值有效，
    如果本次值与上次值之差>A，则本次值无效，放弃本次值，用上次值代替本次值。
C、优点：
    能有效克服因偶然因素引起的脉冲干扰。
D、缺点：
    无法抑制那种周期性的干扰。
    平滑度差。
E、整理：shenhaiyu 2013-11-01
*/

int Filter_Value;
int Value;

void setup() {
  Serial.begin(9600);       // 初始化串口通信
  randomSeed(analogRead(0)); // 产生随机种子
  Value = 300;
}

void loop() {
  Filter_Value = Filter();       // 获得滤波器输出值
  Value = Filter_Value;          // 最近一次有效采样的值，该变量为全局变量
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
```

```
// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
}

// 限幅滤波法（又称程序判断滤波法）
#define FILTER_A 1
int Filter() {
  int NewValue;
  NewValue = Get_AD();
  if(((NewValue - Value) > FILTER_A) || ((Value - NewValue) > FILTER_A))
    return Value;
  else
    return NewValue;
}
```

2、中位值滤波法

```
/*
A、名称：中位值滤波法
B、方法：
    连续采样N次（N取奇数），把N次采样值按大小排列，
    取中间值为本次有效值。
C、优点：
    能有效克服因偶然因素引起的波动干扰；
    对温度、液位的变化缓慢的被测参数有良好的滤波效果。
D、缺点：
    对流量、速度等快速变化的参数不宜。
E、整理：shenhaiyu 2013-11-01
*/

int Filter_Value;

void setup() {
  Serial.begin(9600);       // 初始化串口通信
  randomSeed(analogRead(0)); // 产生随机种子
}

void loop() {
  Filter_Value = Filter();       // 获得滤波器输出值
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
```

```
}

// 中位值滤波法
#define FILTER_N 101
int Filter() {
  int filter_buf[FILTER_N];
  int i, j;
  int filter_temp;
  for(i = 0; i < FILTER_N; i++) {
    filter_buf[i] = Get_AD();
    delay(1);
  }
  // 采样值从小到大排列（冒泡法）
  for(j = 0; j < FILTER_N - 1; j++) {
    for(i = 0; i < FILTER_N - 1 - j; i++) {
      if(filter_buf[i] > filter_buf[i + 1]) {
        filter_temp = filter_buf[i];
        filter_buf[i] = filter_buf[i + 1];
        filter_buf[i + 1] = filter_temp;
      }
    }
  }
  return filter_buf[(FILTER_N - 1) / 2];
}
```

3、算术平均滤波法

```
/*
A、名称：算术平均滤波法
B、方法：
    连续取N个采样值进行算术平均运算：
    N值较大时：信号平滑度较高，但灵敏度较低；
    N值较小时：信号平滑度较低，但灵敏度较高；
    N值的选取：一般流量，N=12；压力：N=4。
C、优点：
    适用于对一般具有随机干扰的信号进行滤波；
    这种信号的特点是有一个平均值，信号在某一数值范围附近上下波动。
D、缺点：
    对于测量速度较慢或要求数据计算速度较快的实时控制不适用；
    比较浪费RAM。
E、整理：shenhaiyu 2013-11-01
*/

int Filter_Value;

void setup() {
  Serial.begin(9600);       // 初始化串口通信
```

```
  randomSeed(analogRead(0)); // 产生随机种子
}

void loop() {
  Filter_Value = Filter();        // 获得滤波器输出值
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
}

// 算术平均滤波法
#define FILTER_N 12
int Filter() {
  int i;
  int filter_sum = 0;
  for(i = 0; i < FILTER_N; i++) {
    filter_sum += Get_AD();
    delay(1);
  }
  return (int)(filter_sum / FILTER_N);
}
```

4、递推平均滤波法（又称滑动平均滤波法）

```
/*
A、名称：递推平均滤波法（又称滑动平均滤波法）
B、方法：
    把连续取得的N个采样值看成一个队列，队列的长度固定为N，
    每次采样到一个新数据放入队尾，并扔掉原来队首的一次数据（先进先出原则），
    把队列中的N个数据进行算术平均运算，获得新的滤波结果。
    N值的选取：流量，N=12；压力，N=4；液面，N=4-12；温度，N=1-4。
C、优点：
    对周期性干扰有良好的抑制作用，平滑度高；
    适用于高频振荡的系统。
D、缺点：
    灵敏度低，对偶然出现的脉冲性干扰的抑制作用较差；
    不易消除由于脉冲干扰所引起的采样值偏差；
    不适用于脉冲干扰比较严重的场合；
    比较浪费RAM。
E、整理：shenhaiyu 2013-11-01
*/

int Filter_Value;
```

```
void setup() {
  Serial.begin(9600);       // 初始化串口通信
  randomSeed(analogRead(0)); // 产生随机种子
}

void loop() {
  Filter_Value = Filter();       // 获得滤波器输出值
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
}

// 递推平均滤波法（又称滑动平均滤波法）
#define FILTER_N 12
int filter_buf[FILTER_N + 1];
int Filter() {
  int i;
  int filter_sum = 0;
  filter_buf[FILTER_N] = Get_AD();
  for(i = 0; i < FILTER_N; i++) {
    filter_buf[i] = filter_buf[i + 1]; // 所有数据左移，低位仍掉
    filter_sum += filter_buf[i];
  }
  return (int)(filter_sum / FILTER_N);
}
```

5、中位值平均滤波法（又称防脉冲干扰平均滤波法）

```
/*
A、名称：中位值平均滤波法（又称防脉冲干扰平均滤波法）
B、方法：
    采一组队列去掉最大值和最小值后取平均值，
    相当于“中位值滤波法”+“算术平均滤波法”。
    连续采样N个数据，去掉一个最大值和一个最小值，
    然后计算N-2个数据的算术平均值。
    N值的选取：3-14。
C、优点：
    融合了“中位值滤波法”+“算术平均滤波法”两种滤波法的优点。
    对于偶然出现的脉冲性干扰，可消除由其所引起的采样值偏差。
    对周期干扰有良好的抑制作用。
    平滑度高，适于高频振荡的系统。
D、缺点：
    计算速度较慢，和算术平均滤波法一样。
```

```
    比较浪费RAM。
E、整理：shenhaiyu 2013-11-01
*/

int Filter_Value;

void setup() {
  Serial.begin(9600);       // 初始化串口通信
  randomSeed(analogRead(0)); // 产生随机种子
}

void loop() {
  Filter_Value = Filter();       // 获得滤波器输出值
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
}

// 中位值平均滤波法（又称防脉冲干扰平均滤波法）（算法1）
#define FILTER_N 100
int Filter() {
  int i, j;
  int filter_temp, filter_sum = 0;
  int filter_buf[FILTER_N];
  for(i = 0; i < FILTER_N; i++) {
    filter_buf[i] = Get_AD();
    delay(1);
  }
  // 采样值从小到大排列（冒泡法）
  for(j = 0; j < FILTER_N - 1; j++) {
    for(i = 0; i < FILTER_N - 1 - j; i++) {
      if(filter_buf[i] > filter_buf[i + 1]) {
        filter_temp = filter_buf[i];
        filter_buf[i] = filter_buf[i + 1];
        filter_buf[i + 1] = filter_temp;
      }
    }
  }
  // 去除最大最小极值后求平均
  for(i = 1; i < FILTER_N - 1; i++) filter_sum += filter_buf[i];
  return filter_sum / (FILTER_N - 2);
}


//  中位值平均滤波法（又称防脉冲干扰平均滤波法）（算法2）
/*
```

```
/*
#define FILTER_N 100
int Filter() {
  int i;
  int filter_sum = 0;
  int filter_max, filter_min;
  int filter_buf[FILTER_N];
  for(i = 0; i < FILTER_N; i++) {
    filter_buf[i] = Get_AD();
    delay(1);
  }
  filter_max = filter_buf[0];
  filter_min = filter_buf[0];
  filter_sum = filter_buf[0];
  for(i = FILTER_N - 1; i > 0; i--) {
    if(filter_buf[i] > filter_max)
      filter_max=filter_buf[i];
    else if(filter_buf[i] < filter_min)
      filter_min=filter_buf[i];
    filter_sum = filter_sum + filter_buf[i];
    filter_buf[i] = filter_buf[i - 1];
  }
  i = FILTER_N - 2;
  filter_sum = filter_sum - filter_max - filter_min + i / 2; // +i/2 的目的是为了四舍五入
  filter_sum = filter_sum / i;
  return filter_sum;
}*/
```

6、限幅平均滤波法

```
/*
A、名称：限幅平均滤波法
B、方法：
    相当于“限幅滤波法”+“递推平均滤波法”；
    每次采样到的新数据先进行限幅处理，
    再送入队列进行递推平均滤波处理。
C、优点：
    融合了两种滤波法的优点；
    对于偶然出现的脉冲性干扰，可消除由于脉冲干扰所引起的采样值偏差。
D、缺点：
    比较浪费RAM。
E、整理：shenhaiyu 2013-11-01
*/

#define FILTER_N 12
int Filter_Value;
int filter_buf[FILTER_N];
```

```
void setup() {
  Serial.begin(9600);       // 初始化串口通信
  randomSeed(analogRead(0)); // 产生随机种子
  filter_buf[FILTER_N - 2] = 300;
}

void loop() {
  Filter_Value = Filter();       // 获得滤波器输出值
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
}

// 限幅平均滤波法
#define FILTER_A 1
int Filter() {
  int i;
  int filter_sum = 0;
  filter_buf[FILTER_N - 1] = Get_AD();
  if(((filter_buf[FILTER_N - 1] - filter_buf[FILTER_N - 2]) > FILTER_A) || ((filter_buf[FILTER_N -
    filter_buf[FILTER_N - 1] = filter_buf[FILTER_N - 2];
  for(i = 0; i < FILTER_N - 1; i++) {
    filter_buf[i] = filter_buf[i + 1];
    filter_sum += filter_buf[i];
  }
  return (int)filter_sum / (FILTER_N - 1);
}
```

7、一阶滞后滤波法

```
/*
A、名称：一阶滞后滤波法
B、方法：
    取a=0-1，本次滤波结果=(1-a)*本次采样值+a*上次滤波结果。
C、优点：
    对周期性干扰具有良好的抑制作用；
    适用于波动频率较高的场合。
D、缺点：
    相位滞后，灵敏度低；
    滞后程度取决于a值大小；
    不能消除滤波频率高于采样频率1/2的干扰信号。
E、整理：shenhaiyu 2013-11-01
```

```
*/

int Filter_Value;
int Value;

void setup() {
  Serial.begin(9600);       // 初始化串口通信
  randomSeed(analogRead(0)); // 产生随机种子
  Value = 300;
}

void loop() {
  Filter_Value = Filter();       // 获得滤波器输出值
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
}

// 一阶滞后滤波法
#define FILTER_A 0.01
int Filter() {
  int NewValue;
  NewValue = Get_AD();
  Value = (int)((float)NewValue * FILTER_A + (1.0 - FILTER_A) * (float)Value);
  return Value;
}
```

8、加权递推平均滤波法

```
/*
A、名称：加权递推平均滤波法
B、方法：
    是对递推平均滤波法的改进，即不同时刻的数据加以不同的权；
    通常是，越接近现时刻的数据，权取得越大。
    给予新采样值的权系数越大，则灵敏度越高，但信号平滑度越低。
C、优点：
    适用于有较大纯滞后时间常数的对象，和采样周期较短的系统。
D、缺点：
    对于纯滞后时间常数较小、采样周期较长、变化缓慢的信号；
    不能迅速反应系统当前所受干扰的严重程度，滤波效果差。
E、整理：shenhaiyu 2013-11-01
*/

int Filter_Value;
```

```
void setup() {
  Serial.begin(9600);       // 初始化串口通信
  randomSeed(analogRead(0)); // 产生随机种子
}

void loop() {
  Filter_Value = Filter();       // 获得滤波器输出值
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
}

// 加权递推平均滤波法
#define FILTER_N 12
int coe[FILTER_N] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};    // 加权系数表
int sum_coe = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12; // 加权系数和
int filter_buf[FILTER_N + 1];
int Filter() {
  int i;
  int filter_sum = 0;
  filter_buf[FILTER_N] = Get_AD();
  for(i = 0; i < FILTER_N; i++) {
    filter_buf[i] = filter_buf[i + 1]; // 所有数据左移，低位仍掉
    filter_sum += filter_buf[i] * coe[i];
  }
  filter_sum /= sum_coe;
  return filter_sum;
}
```

9、消抖滤波法

```
/*
A、名称：消抖滤波法
B、方法：
    设置一个滤波计数器，将每次采样值与当前有效值比较：
    如果采样值=当前有效值，则计数器清零；
    如果采样值<>当前有效值，则计数器+1，并判断计数器是否>=上限N（溢出）；
    如果计数器溢出，则将本次值替换当前有效值，并清计数器。
C、优点：
    对于变化缓慢的被测参数有较好的滤波效果；
    可避免在临界值附近控制器的反复开/关跳动或显示器上数值抖动。
```

```
D、缺点：
    对于快速变化的参数不宜；
    如果在计数器溢出的那一次采样到的值恰好是干扰值,则会将干扰值当作有效值导入系统。
E、整理：shenhaiyu 2013-11-01
*/

int Filter_Value;
int Value;

void setup() {
  Serial.begin(9600);       // 初始化串口通信
  randomSeed(analogRead(0)); // 产生随机种子
  Value = 300;
}

void loop() {
  Filter_Value = Filter();       // 获得滤波器输出值
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
}

// 消抖滤波法
#define FILTER_N 12
int i = 0;
int Filter() {
  int new_value;
  new_value = Get_AD();
  if(Value != new_value) {
    i++;
    if(i > FILTER_N) {
      i = 0;
      Value = new_value;
    }
  }
  else
    i = 0;
  return Value;
}
```

10、限幅消抖滤波法

```
/*
```

```
/*
A、名称：限幅消抖滤波法
B、方法：
    相当于“限幅滤波法”+“消抖滤波法”；
    先限幅，后消抖。
C、优点：
    继承了“限幅”和“消抖”的优点；
    改进了“消抖滤波法”中的某些缺陷，避免将干扰值导入系统。
D、缺点：
    对于快速变化的参数不宜。
E、整理：shenhaiyu 2013-11-01
*/

int Filter_Value;
int Value;

void setup() {
  Serial.begin(9600);       // 初始化串口通信
  randomSeed(analogRead(0)); // 产生随机种子
  Value = 300;
}

void loop() {
  Filter_Value = Filter();       // 获得滤波器输出值
  Serial.println(Filter_Value); // 串口输出
  delay(50);
}

// 用于随机产生一个300左右的当前值
int Get_AD() {
  return random(295, 305);
}

// 限幅消抖滤波法
#define FILTER_A 1
#define FILTER_N 5
int i = 0;
int Filter() {
  int NewValue;
  int new_value;
  NewValue = Get_AD();
  if(((NewValue - Value) > FILTER_A) || ((Value - NewValue) > FILTER_A))
    new_value = Value;
  else
    new_value = NewValue;
  if(Value != new_value) {
    i++;
    if(i > FILTER_N) {
      i = 0;
      Value = new_value;
    }
```

```
  }
  else
    i = 0;
  return Value;
}
```

11、卡尔曼滤波（非扩展卡尔曼）

```
#include <Wire.h> // I2C library, gyroscope

// Accelerometer ADXL345
#define ACC (0x53)    //ADXL345 ACC address
#define A_TO_READ (6)        //num of bytes we are going to read each time (two bytes for each axis


// Gyroscope ITG3200
#define GYRO 0x68 // gyro address, binary = 11101000 when AD0 is connected to Vcc (see schematics c
#define G_SMPLRT_DIV 0x15
#define G_DLPF_FS 0x16
#define G_INT_CFG 0x17
#define G_PWR_MGM 0x3E

#define G_TO_READ 8 // 2 bytes for each axis x, y, z


// offsets are chip specific.
int a_offx = 0;
int a_offy = 0;
int a_offz = 0;

int g_offx = 0;
int g_offy = 0;
int g_offz = 0;
//////////////////////////

//////////////////////////
char str[512];

void initAcc() {
  //Turning on the ADXL345
  writeTo(ACC, 0x2D, 0);
  writeTo(ACC, 0x2D, 16);
  writeTo(ACC, 0x2D, 8);
  //by default the device is in +-2g range reading
}
```

```
void getAccelerometerData(int* result) {
  int regAddress = 0x32;    //first axis-acceleration-data register on the ADXL345
  byte buff[A_TO_READ];

  readFrom(ACC, regAddress, A_TO_READ, buff); //read the acceleration data from the ADXL345

  //each axis reading comes in 10 bit resolution, ie 2 bytes.  Least Significat Byte first!!
  //thus we are converting both bytes in to one int
  result[0] = (((int)buff[1]) << 8) | buff[0] + a_offx;
  result[1] = (((int)buff[3]) << 8) | buff[2] + a_offy;
  result[2] = (((int)buff[5]) << 8) | buff[4] + a_offz;
}

//initializes the gyroscope
void initGyro()
{
  /**************************************
  * ITG 3200
  * power management set to:
  * clock select = internal oscillator
  *     no reset, no sleep mode
  *   no standby mode
  * sample rate to = 125Hz
  * parameter to +/- 2000 degrees/sec
  * low pass filter = 5Hz
  * no interrupt
  **************************************/
  writeTo(GYRO, G_PWR_MGM, 0x00);
  writeTo(GYRO, G_SMPLRT_DIV, 0x07); // EB, 50, 80, 7F, DE, 23, 20, FF
  writeTo(GYRO, G_DLPF_FS, 0x1E); // +/- 2000 dgrs/sec, 1KHz, 1E, 19
  writeTo(GYRO, G_INT_CFG, 0x00);
}


void getGyroscopeData(int * result)
{
  /**************************************
  Gyro ITG-3200 I2C
  registers:
  temp MSB = 1B, temp LSB = 1C
  x axis MSB = 1D, x axis LSB = 1E
  y axis MSB = 1F, y axis LSB = 20
  z axis MSB = 21, z axis LSB = 22
  **************************************/

  int regAddress = 0x1B;
  int temp, x, y, z;
  byte buff[G_TO_READ];

  readFrom(GYRO, regAddress, G_TO_READ, buff); //read the gyro data from the ITG3200
```

```
  result[0] = ((buff[2] << 8) | buff[3]) + g_offx;
  result[1] = ((buff[4] << 8) | buff[5]) + g_offy;
  result[2] = ((buff[6] << 8) | buff[7]) + g_offz;
  result[3] = (buff[0] << 8) | buff[1]; // temperature

}



float xz=0,yx=0,yz=0;
float p_xz=1,p_yx=1,p_yz=1;
float q_xz=0.0025,q_yx=0.0025,q_yz=0.0025;
float k_xz=0,k_yx=0,k_yz=0;
float r_xz=0.25,r_yx=0.25,r_yz=0.25;
  //int acc_temp[3];
  //float acc[3];
  int acc[3];
  int gyro[4];
  float Axz;
  float Ayx;
  float Ayz;
  float t=0.025;
void setup()
{
  Serial.begin(9600);
  Wire.begin();
  initAcc();
  initGyro();

}

//unsigned long timer = 0;
//float o;
void loop()
{

  getAccelerometerData(acc);
  getGyroscopeData(gyro);
  //timer = millis();
  sprintf(str, "%d,%d,%d,%d,%d,%d", acc[0],acc[1],acc[2],gyro[0],gyro[1],gyro[2]);

  //acc[0]=acc[0];
  //acc[2]=acc[2];
  //acc[1]=acc[1];
  //r=sqrt(acc[0]*acc[0]+acc[1]*acc[1]+acc[2]*acc[2]);
  gyro[0]=gyro[0]/ 14.375;
  gyro[1]=gyro[1]/ (-14.375);
  gyro[2]=gyro[2]/ 14.375;
```

```
  Axz=(atan2(acc[0],acc[2]))*180/PI;
  Ayx=(atan2(acc[0],acc[1]))*180/PI;
  /*if((acc[0]!=0)&&(acc[1]!=0))
    {
      Ayx=(atan2(acc[0],acc[1]))*180/PI;
    }
    else
    {
      Ayx=t*gyro[2];
    }*/
  Ayz=(atan2(acc[1],acc[2]))*180/PI;


 //kalman filter
  calculate_xz();
  calculate_yx();
  calculate_yz();

  //sprintf(str, "%d,%d,%d", xz_1, xy_1, x_1);
  //Serial.print(xz);Serial.print(",");
  //Serial.print(yx);Serial.print(",");
  //Serial.print(yz);Serial.print(",");
  //sprintf(str, "%d,%d,%d,%d,%d,%d", acc[0],acc[1],acc[2],gyro[0],gyro[1],gyro[2]);
  //sprintf(str, "%d,%d,%d",gyro[0],gyro[1],gyro[2]);
    Serial.print(Axz);Serial.print(",");
    //Serial.print(Ayx);Serial.print(",");
    //Serial.print(Ayz);Serial.print(",");
  //Serial.print(str);
  //o=gyro[2];//w=acc[2];
  //Serial.print(o);Serial.print(",");
  //Serial.print(w);Serial.print(",");
  Serial.print("\n");


  //delay(50);
}
void calculate_xz()
{

 xz=xz+t*gyro[1];
 p_xz=p_xz+q_xz;
 k_xz=p_xz/(p_xz+r_xz);
 xz=xz+k_xz*(Axz-xz);
 p_xz=(1-k_xz)*p_xz;
}
void calculate_yx()
{
```

```
  yx=yx+t*gyro[2];
  p_yx=p_yx+q_yx;
  k_yx=p_yx/(p_yx+r_yx);
  yx=yx+k_yx*(Ayx-yx);
  p_yx=(1-k_yx)*p_yx;

}
void calculate_yz()
{
  yz=yz+t*gyro[0];
  p_yz=p_yz+q_yz;
  k_yz=p_yz/(p_yz+r_yz);
  yz=yz+k_yz*(Ayz-yz);
  p_yz=(1-k_yz)*p_yz;

}


//---------------- Functions
//Writes val to address register on ACC
void writeTo(int DEVICE, byte address, byte val) {
   Wire.beginTransmission(DEVICE); //start transmission to ACC
   Wire.write(address);        // send register address
   Wire.write(val);        // send value to write
   Wire.endTransmission(); //end transmission
}


//reads num bytes starting from address register on ACC in to buff array
void readFrom(int DEVICE, byte address, int num, byte buff[]) {
  Wire.beginTransmission(DEVICE); //start transmission to ACC
  Wire.write(address);        //sends address to read from
  Wire.endTransmission(); //end transmission

  Wire.beginTransmission(DEVICE); //start transmission to ACC
  Wire.requestFrom(DEVICE, num);    // request 6 bytes from ACC

  int i = 0;
  while(Wire.available())    //ACC may send less than requested (abnormal)
  {
    buff[i] = Wire.read(); // receive a byte
    i++;
  }
  Wire.endTransmission(); //end transmission
}
```

> 来源：机器人网

- EOF -

推荐阅读 点击标题可跳转

1、为什么腾讯/阿里不去开发被卡脖子的工业软件?

2、从此明白了卷积神经网络（CNN）

3、QUIC 是如何解决TCP 性能瓶颈的?

觉得本文有帮助？请分享给更多人

推荐关注「算法爱好者」，修炼编程内功

算法爱好者

算法是程序员的内功！「算法爱好者」专注分享算法相关文章、工具资源和算法题，帮...

27篇原创内容

公众号

点赞和在看就是最大的支持♡

喜欢此内容的人还喜欢

使用 Tini 清理 Docker 容器僵死进程

k8s技术圈

逐步搭建出团队的 vue3 前端架构

程序员哆啦A梦

Linux I/O 那些事儿

腾讯技术工程