Algorithms

Low-pass filters

IIR

```
y[i] :=  ß * x[i] + (1-ß) * y[i-1]
```

From Wiki: This discrete-time implementation of a simple RC low-pass filter is the exponentially weighted moving average (aka Exponential smoothing)

```
Exponential smoothing:
y[i] :=  y[i-1] +  α * (x[i] - y[i-1])
where  α = (1-ß) (from above)
```

```
#define ALPHA_PERCENT 60
int16_t exponential_smoothing(int16_t input, int16_t old_value) {
    int32_t val = old_value;
    int32_t diff = input - old_value;

    val += (ALPHA_PERCENT * diff) / 100;
    return val;
}
```

Moving Average (MA)

aka moving average / rolling average / running average

```
#define LPA_NUM     10
static int16_t lpa_buf[LPA_NUM];

void low_pass_average_init(int16_t init_value) {
    for (int i = 0; i < sizeof(lpa_buf)/sizeof(lpa_buf[0]); i++) {
        lpa_buf[i] = init_value;
    }
}

int16_t low_pass_average(int16_t input) {
    int32_t sum = 0;
    for (unsigned int i = (LPA_NUM)-1; i > 0; --i) {
        lpa_buf[i] = lpa_buf[i-1];
        sum += lpa_buf[i];
    }
    sum += input;
```

```
    lpa_buf[0] = input;
    return sum/LPA_NUM;
}
```

Weighted Moving Average

Time-weighted moving average LWMA - Linearly Weighted Moving Average

```
#define LPA_NUM     10
static int16_t wma_buf[LPA_NUM];
static int16_t ma_w[LPA_NUM] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };

void mag_moving_average_init(int16_t init_value) {
    for (int i = 0; i < sizeof(ma_buf)/sizeof(ma_buf[0]); i++) {
        wma_buf[i] = init_value;
    }
}

nt16_t mag_weighted_moving_average(int16_t input) {
    int32_t sum = 0;
    int32_t w_sum = 0;
    for (unsigned int i = (LPA_NUM)-1; i > 0; --i) {
        wma_buf[i] = wma_buf[i-1];
        sum += wma_buf[i] * ma_w[i];
        w_sum += ma_w[i];
    }
    wma_buf[0] = input;
    sum += wma_buf[0] * ma_w[0];
    w_sum += ma_w[0];
    return sum / w_sum;
}
```

Change detection

aka: anomaly detection

CuSum

- CUSUM - Cumulative sum control chart
- https://hal.archives-ouvertes.fr/hal-00914697/document