Алгоритм регистрации шага

Описанный алгоритм предполагается использовать для детектирования шага пользователя при помощи его мобильного устройства. Детектор использует амплитуду и производную (по времени) нормы ускорения:

$$|\vec{a}(i)| = \sqrt{a_x^2(i) + a_y^2(i) + a_z^2(i)}, \quad (1)$$

где $i \in [1; I]$ – момент времени, для которого посчитана норма ускорения.

Алгоритм является адаптивным, т.е. его параметры автоматически уточняются в процессе работы. Начальные значения параметров задаются при запуске алгоритма (приложения).

Сильные стороны алгоритма

1. Детектирование шага происходит в точках действительного начала и окончания шага, а именно, в те моменты времени, когда текущее значение $|\vec{a}(i)|$ превышает среднее значение $a_{av}(i)$, при положительном текущем значении производной $|\vec{a}(i)|$ по времени;
2. Критерии, используемые в алгоритме, пересчитываются с разной частотой (в каждый момент регистрации показаний сенсора или в момент каждого задетектированного шага), но в конечном итоге они самостоятельно подстраиваются к конкретному пользователю;
3. Возможно (в будущем) применение данного алгоритма позволит вернуться к ZUPT алгоритму оценки длины шага, поскольку шаг детектируется полностью от нуля к нулю.

Слабые стороны алгоритма

1. Одним из критериев детектирования шага является амплитуда сигнала ускорения, A(s), которая пересчитывается каждый задетектированный шаг пользователя s. Иногда этот критерий может вести себя неадекватно. В будущем возможно его либо доработаем, либо заменим другим, например, на пороговое значение нормы ускорения;
2. Все же необходимо задавать начальные значения параметров алгоритма;
3. Алгоритм в строгом понимании не является алгоритмом обработки в реальном времени – для определения того, что шаг был, необходимо дождаться окончания детектирования шага.

Описание алгоритма

При отсутствии движения пользователя значение нормы ускорения, отфильтрованное низкочастотным фильтром Баттерворта, флуктуирует вокруг своего среднего значения. Когда же пользователь начинает перемещение, появляются характерные синусообразные колебания $|\vec{a}(i)|$.

При совершении шага норма ускорения сначала возрастает, достигая своего пикового положительного значения, затем уменьшается, достигая своего минимального уровня, и в итоге возвращается к среднему значению, образуя полный цикл шага.

При получении каждого нового значения $\vec{a}(i)$ (с частотой регистрации) алгоритм обновляет среднее значение ускорения $a_{av}(i)$ по следующей формуле:

$$a_{av}(i) = (1 - w_{av}) \cdot a_{av}(i-1) + w_{av} \cdot |\vec{a}(i)|, \quad (2)$$

где $a_{av}(i)$ – среднее значение $|\vec{a}(i)|$ на i-й момент времени; при инициализации $a_{av}(0)$ равно 1,1 грав. ед.

w_{av} – вес, с которым учитывается текущее измеренное значение нормы ускорения в среднем значении нормы ускорения (больший вес приводит к большим колебаниям среднего значения a_{av}); по умолчанию равен 0,01.

Таким образом, алгоритм постепенно сам уточняет среднее значение нормы ускорения не зависимо от того, насколько точно было задано начальное значение.

Регистрация начала и конца шага происходит при выполнении нескольких условий, основным из которых является следующее (в блок-схеме названо "StepCondition"):

$$\frac{|\vec{a}(i)|-|\vec{a}(i-1)|}{t(i)-t(i-1)} > 0 \text{ и } |\vec{a}(i)| > a_{av}(i) \text{ и } |\vec{a}(i-1)| < a_{av}(i), \quad (3)$$

где $t(i)$ – *timestamp* i-го значения вектора ускорения.

Дополнительным условием для регистрации начала шага является выполнение условия $t_{start} = 0$, где t_{start} – момент начала регистрации потенциального шага. При инициализации данный параметр равен 0. При выполнении данного условия и условия (3) алгоритм выбрасывает флаг STEP_START (см. блок схему).

Окончание шага и его верификация, т.е. подтверждение того, что зарегистрированное синусообразное колебание действительно является шагом, происходит при одновременном выполнении условия (3) и условий:

$$t_{start} \neq 0 \text{ и } a_{max}(s) - a_{min}(s) > \mathrm{C_A} \cdot \mathrm{A}(s-1), \quad (4)$$

где $a_{max}(s)$, $a_{min}(s)$ – максимальное и минимальное значения нормы ускорения на текущем шаге s;

$s \in [1; S]$ – номер текущего детектируемого шага;

A(s-1) – текущее значение амплитуды шага (на момент текущего детектируемого шага s), рассчитанное с учетом предыдущих (s-1) шагов; при инициализации $\mathrm{A}(0)$ равно 0,2 грав. ед.

$\mathrm{C_A}$ – настроечный параметр, равен 0,6.

Амплитуда шага с учетом предыдущих (s-1) шагов, A(s), вычисляется следующим образом:

$$\mathrm{A}(s) = \left(1 - w_{amp}\right) \cdot \mathrm{A}(s-1) + w_{amp} \cdot (a_{max}(s) - a_{min}(s)), \quad (5)$$

где w_{amp} – вес текущего значения амплитуды s-го шага; по умолчанию равен 0,05.

Кроме того, существует еще одно условие, которое определяет действительно задетектирован шаг или нет. Это временное условие, определяет длительность шага:

$$t(i) - t_{start} > C_T \cdot T_{stepAvg}(s-1), \quad (6)$$

где C_T – поправочный коэффициент, по сути запас на вариацию времени от шага к шагу; по умолчанию равен 1,3;

$T_{stepAvg}(s-1)$ – среднее время шага, рассчитанное по предыдущим (s-1) шагам:

$$T_{stepAvg}(s) = \left(1 - w_{step}\right) \cdot T_{stepAvg}(s-1) + w_{step} \cdot (t(i) - t_{start}), \quad (7)$$

где w_{step} – вес текущего значения времени s-го шага; по умолчанию равен 0,05.

$T_{stepAvg}(0)$ – начальное значение длительности шага, задаваемое при инициализации алгоритма, равно 1 сек.

Если длительность текущего шага превышает среднюю длительность шага, рассчитанную по предыдущим s-1 шагам, то в этом случае шаг не учитывается и алгоритм выбрасывает или флаг STEP_RESET, по которому значения всех параметров не пересчитываются, как при новом шаге, время шага сбрасывается в значение по умолчанию.

В случае, если верификация шага прошла успешно (выполнились условия (3)-(4) и (6)), детектор выбрасывает флаг STEP_END_START (так как условие конца шага совпадает после обнуления t_{start} с условием начала следующего шага), возвращает время окончания $t_{end}(s)$ шага и рассчитывает новое значение средней по предыдущим шагам амплитуды (5) и длительности шага (7), сбрасывает ряд параметров в их исходные значения:

$$t_{start} = 0, \quad a_{max}(s+1) = C_{MAX}, \quad a_{min}(s+1) = C_{MIN}, \tag{8}$$

где C_{MIN} – константа, требуется для корректного определения минимального значения $|\vec{a}(i)|$ на следующем шаге; по умолчанию равно 2 (произвольное положительное число; если будет близким к нулю, то есть вероятность пропустить минимум; потому решили сделать равным 2);

C_{MAX} – константа, требуется для корректного определения максимального значения $|\vec{a}(i)|$ на следующем шаге; по умолчанию равно 0 (произвольное положительное малое число; если будет больше нуля, есть вероятность пропустить максимум; потому решили сделать равным 0).

Если же шаг не был верифицирован (условия (3)-(4) или (6) не выполнились), то детектор выбрасывает флаг STEP_RESET, обнуляет t_{start} и восстанавливает остальные необходимые параметры в их исходные значения (см. блок-схему).

Ссылка на оригинал блок-схемы 1 - https://www.dropbox.com/s/zo8fsg3qwo2ryza/FelixStepDetector%231_1.png?dl=0

Ссылка на оригинал блок-схемы 2 - https://www.dropbox.com/s/ajk6orrnm5o64d5/FelixStepDetector%231_2.png?dl=0

Реализация алгоритма

1. На вход алгоритма передается текущий вектор ускорения и его таймстемп; на выходе алгоритма – один из флагов;
2. Типы флагов:
    a. STEP_NONE – ничего не происходит, устройство в стационарном положении;
    b. STEP_START – сработало условие начала шага;
    c. STEP_IN – данный флаг выбрасывается, когда детектор определил начало шага (STEP_START либо STEP_RESET_START), но шаг еще не закончился;
    d. STEP_END_START – сработало условие конца шага, но при этом эта же точка прошла по условию начала следующего шага;
    e. STEP_RESET – ранее сработало условие начала шага, но шаг оказался ложным (не прошел какую-либо проверку) и не должен быть учтен;
    f. STEP_RESET_START – ранее сработало условие начала шага, но шаг оказался ложным (не прошел какую-либо проверку) и не должен быть учтен; но при этом также сработало условие начала шага.
3. При получении флага STEP_NONE основная программа ничего не делает;
4. При получении флага STEP_START основная программа сбрасывает накопленные значения (ΔX; ΔY) в нуль (также обнуляются другие необходимые переменные) и для каждого последующего *timestamp* начинает определять и накапливать перемещения устройства, (ΔX; ΔY), но не выводить при этом смещение маркера на экран и не передавать их в PF;
5. При получении флага STEP_END_START ситуация соответствует случаю, когда пользователь сделал один шаг и продолжает идти дальше; накопленные за время текущего шага смещения (ΔX; ΔY) должны быть выведены на экран устройства (и учтены в других частях системы, в частности, переданы в PF); далее программа выполняет действия, как при получении флага STEP_START;
6. При получении флага STEP_RESET система понимает, что шаг был ложным, потому сбрасывает накопленные значения (ΔX; ΔY) в нуль и начинает ждать флага STEP_START; маркер пользователя на экране при этом никуда не перемещается; при данном флаге подразумевается, что начало нового шага по каким-либо причинам задетектировано не было;

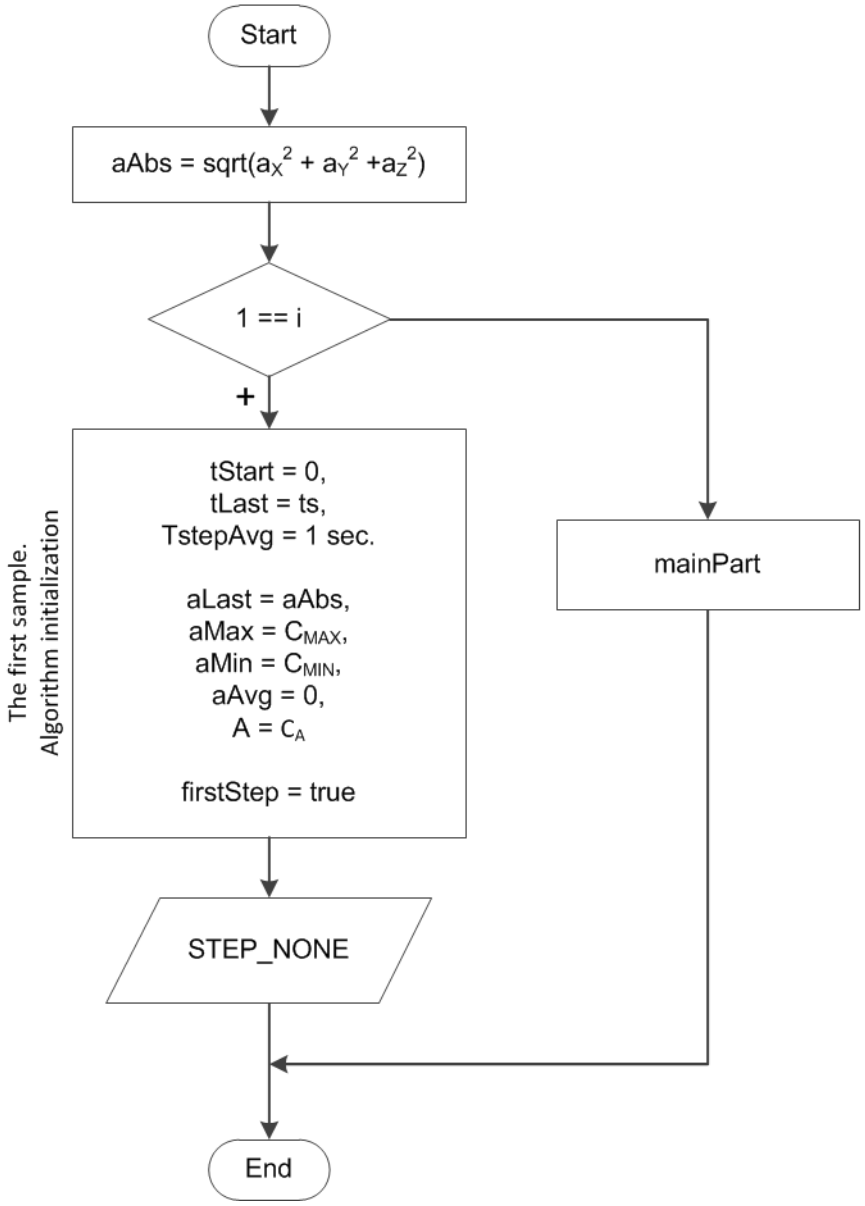
7. При получении флага STEP_RESET_START система понимает, что шаг был ложным, потому сбрасывает накопленные значения (ΔX; ΔY) в нуль; маркер пользователя на экране при этом никуда не перемещается; при данном флаге подразумевается, что также задетектировано начало нового шага и, следовательно, затем также выполняются действия, соответствующие флагу STEP_START;
8. При получении флага STEP_IN продолжается накопление значений (ΔX; ΔY) при отсутствии их вывода на экран; программа находится в процессе отслеживания шага пользователя, начало которого было определено раньше.

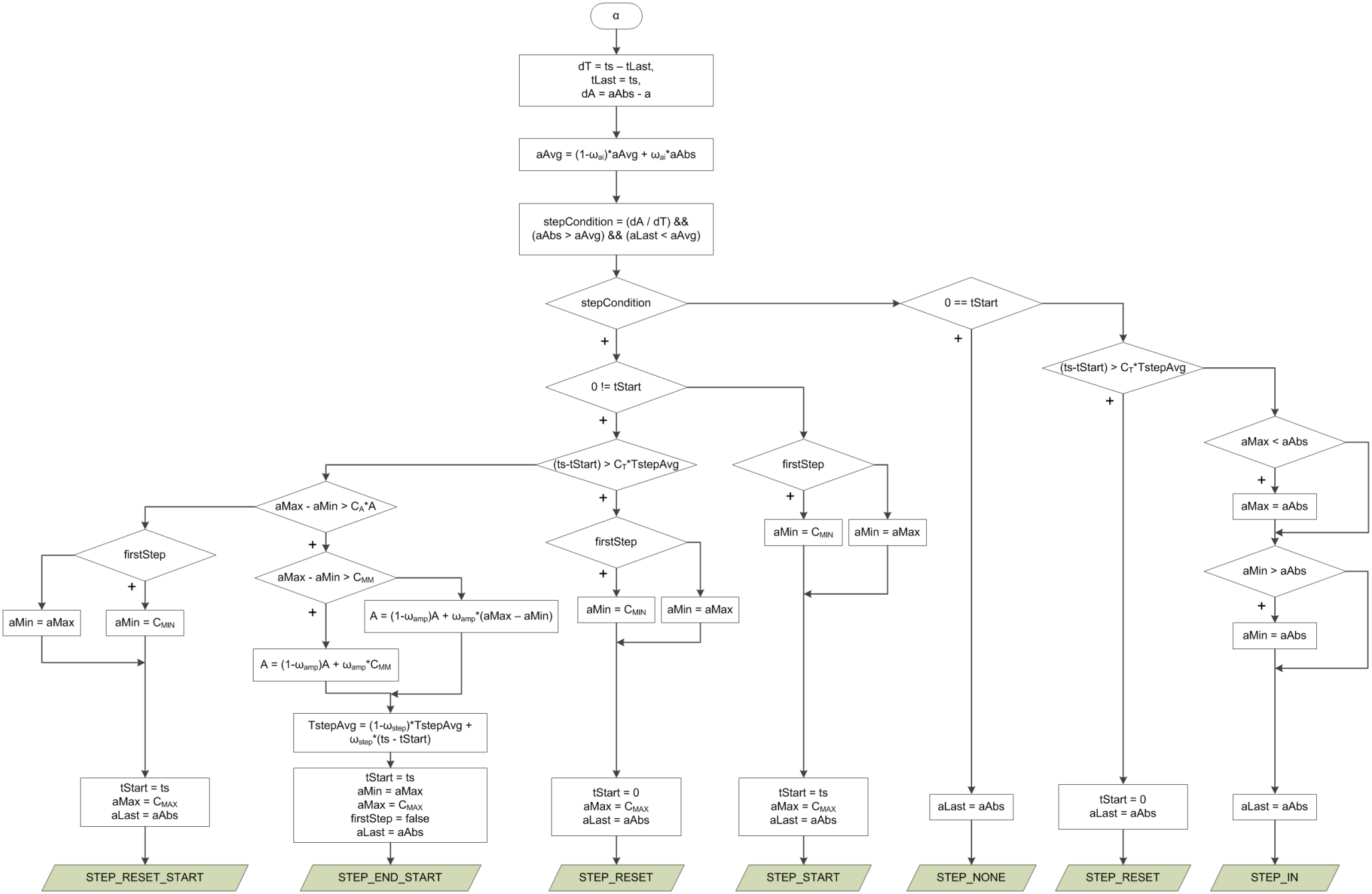
Константы и параметры алгоритма

№№	Обозначение	Описание	Значение по умолчанию
1	C_{MIN}	Требуется для корректного определения минимального значения $\|\vec{a}(i)\|$ на следующем шаге	Равно 2 (произвольное положительное число; если будет близким к нулю, то есть вероятность пропустить минимум; потому решили сделать равным 2);
2	C_{MAX}	Требуется для корректного определения максимального значения $\|\vec{a}(i)\|$ на следующем шаге	Равно 0 (произвольное положительное малое число; если будет больше нуля, есть вероятность пропустить максимум; потому решили сделать равным 0)
3	$a_{av}(0)$	Требуется при расчете среднего значения модуля ускорения в первой итерации	Равно 1.1 грав. ед.
4	w_{av}	Требуется при пересчете среднего значения модуля ускорения	Равно 0.01
5	A(0)	Требуется для расчета значения амплитуды шага в первой итерации	Равно 0.2 грав. ед.
6	w_{amp}	Требуется при пересчете усредненной амплитуды в конце каждого задетектированного шага	Равно 0.05
7	$T_{stepAvg}(0)$	Требуется для учета длительности шага при детектировании в первой итерации.	Равно 1 сек.
8	w_{step}	Вес ограничения по времени. Требуется при пересчете средней длительности шага.	Равно 0.05
9	C_T	Запас при проверке ограничения по времени. Требуется для учета возможностей особенности походки и учета приближения пользователя к преграде (шаги замедляются)	Равно 1.3
10	C_A	Множитель для амплитуды при верификации шага. Требуется для проверки условия (4).	Равно 0.6
11	C_{MM}	Параметр, который позволяет проверять случаи, когда амплитуда импульса текущего шага значительно больше предыдущих.	Равно 0.8

Блок-схема алгоритма

// Function implements Step detector proposed by Felix Sirenko.
// INPUT PARAMETERS:
// a - vector, 1x3, full acceleration value (a_X, a_Y, a_Z) from the device sensor, in grav.units
// ts - double, timestamp of received acceleration value, in sec.
// OUTPUT PARAMETER:
// STATUS_FLAG - int, current state of the detector.




α
dT = ts – tLast,
tLast = ts,
dA = aAbs - a
aAvg = (1-ωai)*aAvg + ωai*aAbs
stepCondition = (dA / dT) &&
(aAbs > aAvg) && (aLast < aAvg)
stepCondition
0 == tStart
0 != tStart
(ts-tStart) > CT*TstepAvg
aMax - aMin > CA*A
aMax - aMin > CMM
firstStep
aMin = aMax
aMin = CMIN
A = (1-ωamp)A + ωamp*CMM
A = (1-ωamp)A + ωamp*(aMax – aMin)
TstepAvg = (1-ωstep)*TstepAvg +
ωstep*(ts - tStart)
tStart = ts
aMax = CMAX
aLast = aAbs
tStart = ts
aMin = aMax
aMax = CMAX
firstStep = false
aLast = aAbs
tStart = 0
aMax = CMAX
aLast = aAbs
tStart = ts
aMax = CMAX
aLast = aAbs
aLast = aAbs
(ts-tStart) > CT*TstepAvg
tStart = 0
aLast = aAbs
aMax < aAbs
aMax = aAbs
aMin > aAbs
aMin = aAbs
aLast = aAbs
STEP_RESET_START
STEP_END_START
STEP_RESET
STEP_START
STEP_NONE
STEP_RESET
STEP_IN