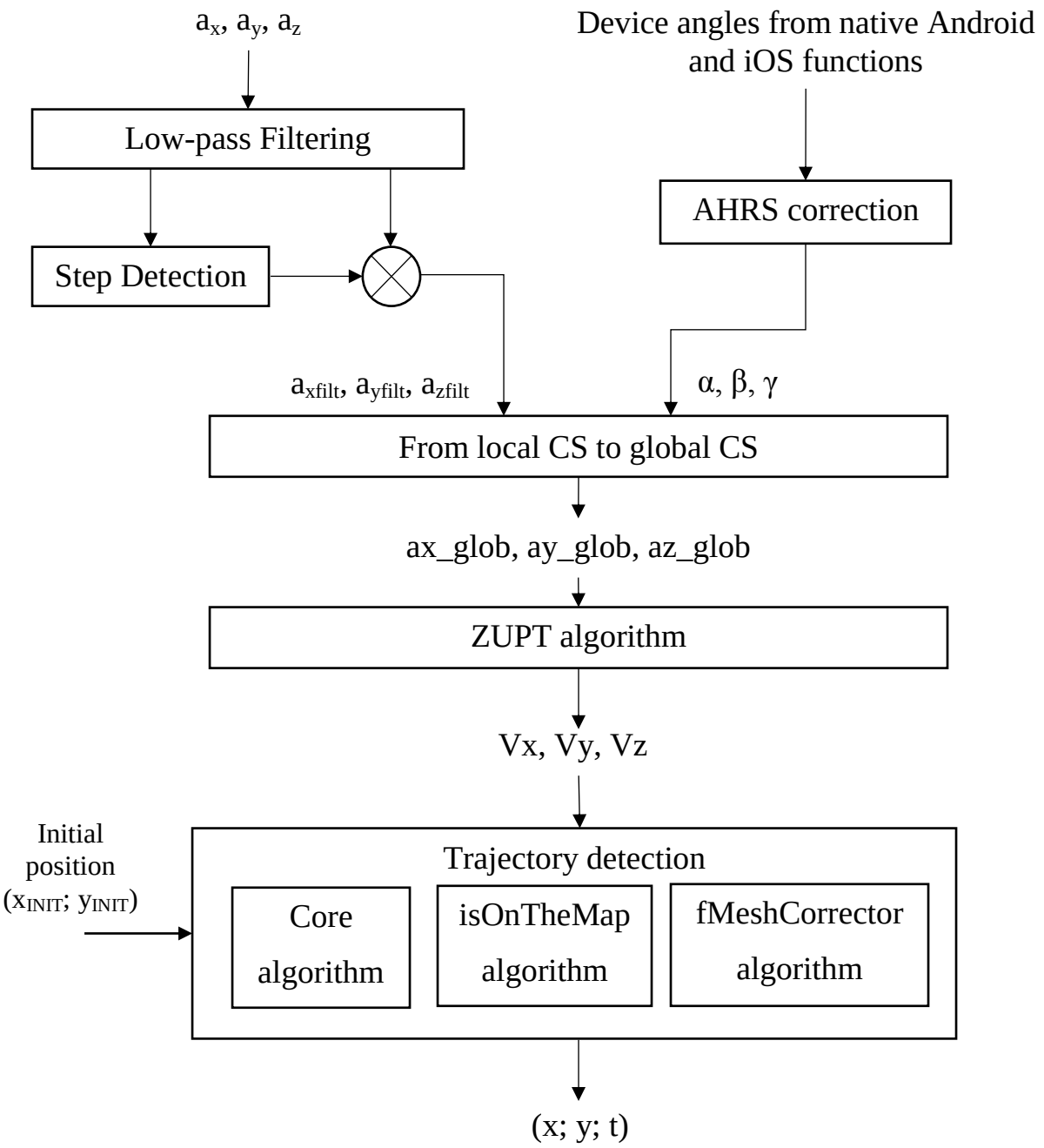
Описание алгоритма навигации по сенсорам

ОБЩАЯ СХЕМА



1. Low-pass filtering

Фильтрация данных акселерометра низкочастотным фильтром Баттерворта для удаления шумов и более эффективного обнаружения паттернов шагов пользователя.

Входные параметры:

a_x, a_y, a_z	ускорения вдоль осей X, Y и Z устройства соответственно (в единицах гравитационного ускорения)
samplePeriod	период регистрации данных от сенсора (в секундах)
cutoffLow	частота среза для НЧ фильтра Баттерворта (Гц)
Порядок фильтра	8 для наибольшей скорости спада АЧХ

Выходные параметры:

$a_{x\ filt}$, $a_{y\ filt}$, $a_{z\ filt}$	Отфильтрованные значения проекций ускорения в системе координат устройства (в гравитационных единицах)

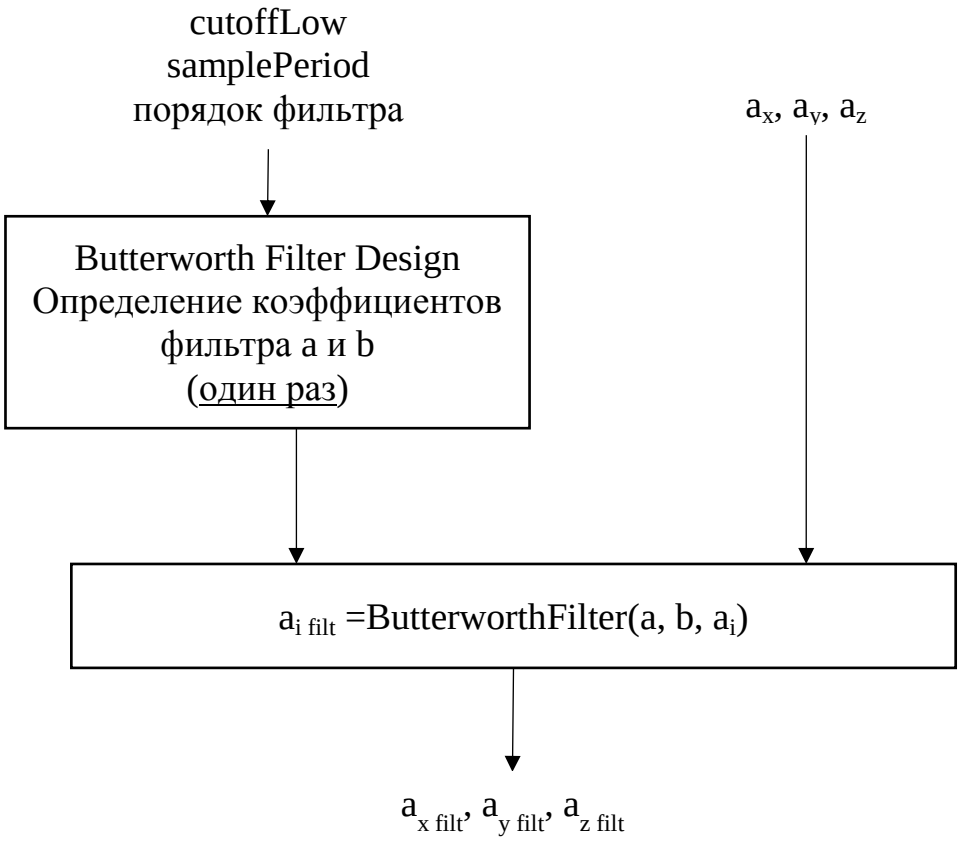
Значения по умолчанию

cutoffLow = 2.2 Гц (определено в результате исследований);

Рекомендации:

- Фильтр Баттерворта писать вручную не стоит, поскольку данный фильтр имеет достаточно много реализаций как на C, так и на C++.

Блок-схема


cutoffLow
samplePeriod
порядок фильтра
a_x, a_y, a_z
Butterworth Filter Design
Определение коэффициентов фильтра a и b
(один раз)
$a_{i\ filt}$ =ButterworthFilter(a, b, a_i)
$a_{x\ filt}$, $a_{y\ filt}$, $a_{z\ filt}$

2. Step detection algorithm

Алгоритм, который на основании отфильтрованного показания акселерометра и заданного порогового значения ускорения определяет, соответствует ли зарегистрированное значение ускорения статическому (пользователь неподвижен) или динамическому (пользователь идет) случаям.

Входные параметры:

accTH — пороговое значение ускорения (если ускорение больше порогового, то это динамический случай, если меньше – статический)

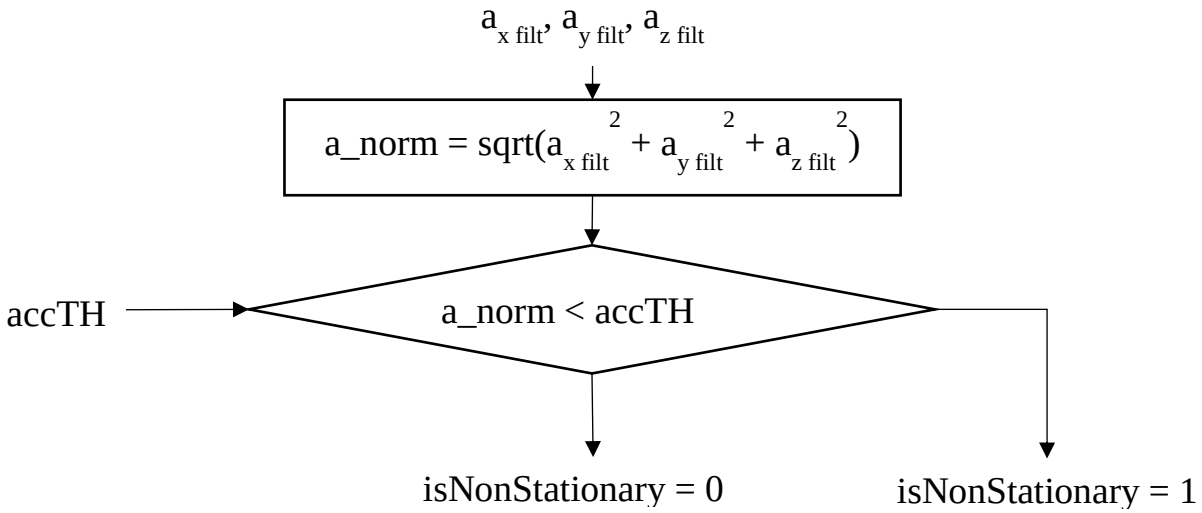
Выходные параметры:

isNonStationary — Признак движения:
1 – динамический случай (движение есть),
0 – статический случай (движения нет)

Значения по умолчанию

accTH = 1.035 грав. ед. (определено экспериментально);

Блок-схема



3. AHRS Correction

Под коррекцией AHRS понимаем корректировку углов, полученных с помощью устройства, относительно северного магнитного полюса. Корректировка осуществляется по следующей формуле:

$$\begin{cases} Pitch = Pitch \\ Roll = Roll \\ Yaw = -Yaw - 180 + MapOrientationAngle \end{cases}$$

Значения по умолчанию

MapOrientationAngle = 20 град. (для офиса IT-Jim)

MapOrientationAngle = 0 (для офиса Kaa Solutions) *уточню на днях*

(*в будущем этот параметр будет задаваться в json-файле карты в качестве одного из параметров; по сути – это количество градусов, на которое карта в сохраненном виде и при отображении на экране пользователя повернута относительно Севера*).

4. From Local CS To Global CS

Алгоритм, который преобразует зарегистрированные ускорения из локальной системы координат устройства в глобальную систему координат.

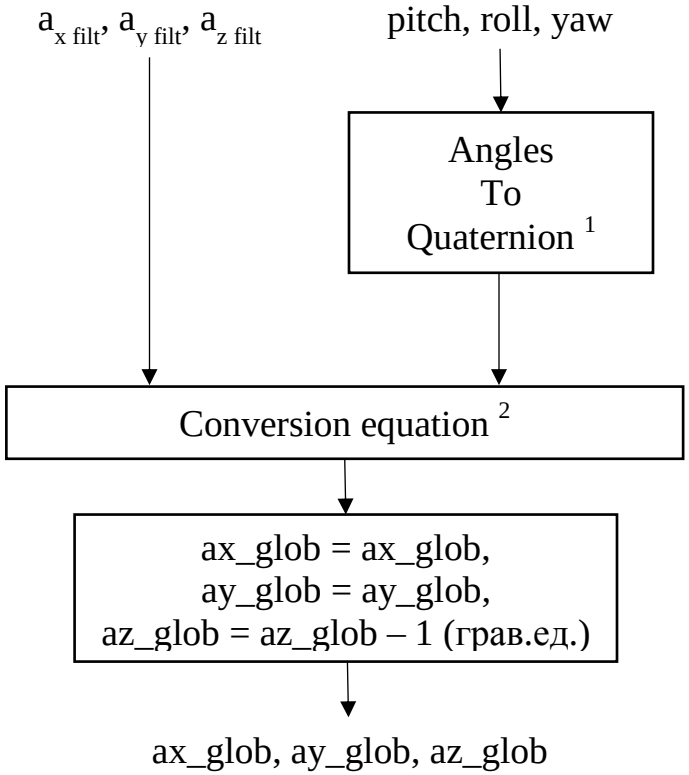
Входные параметры:

$a_{x\,filt}$, $a_{y\,filt}$, $a_{z\,filt}$	Отфильтрованные значения проекций ускорения в системе координат устройства (в грав. единицах)
Pitch, roll, yaw	углы после коррекции (в рад.)

Выходные параметры:

ax_glob, ay_glob, az_glob	Ускорения в глобальной системе координат

Блок-схема



[1] $$q = \begin{pmatrix} \cos\left(\frac{pitch}{2}\right) * \cos\left(\frac{roll}{2}\right) * \cos\left(\frac{roll}{2}\right) - \sin\left(\frac{pitch}{2}\right) * \sin\left(\frac{roll}{2}\right) * \sin\left(\frac{yaw}{2}\right) \\ \cos\left(\frac{pitch}{2}\right) * \sin\left(\frac{\text{roll}}{2}\right) * \sin\left(\frac{yaw}{2}\right) + \sin\left(\frac{pitch}{2}\right) * \cos\left(\frac{\text{roll}}{2}\right) * \cos\left(\frac{yaw}{2}\right) \\ \cos\left(\frac{pitch}{2}\right) * \sin\left(\frac{\text{roll}}{2}\right) * \cos\left(\frac{yaw}{2}\right) - \sin\left(\frac{pitch}{2}\right) * \cos\left(\frac{\text{roll}}{2}\right) * \sin\left(\frac{yaw}{2}\right) \\ \cos\left(\frac{pitch}{2}\right) * \cos\left(\frac{\text{roll}}{2}\right) * \sin\left(\frac{yaw}{2}\right) + \sin\left(\frac{pitch}{2}\right) * \sin\left(\frac{\text{roll}}{2}\right) * \cos\left(\frac{yaw}{2}\right) \end{pmatrix}$$

[2] Выражение для преобразования:

$$[ax\_glob, ay\_glob, az\_glob] = rotate([a_{x\ filt}\ \ a_{y\ filt}\ \ a_{z\ filt}],\ \hat{q})$$

где $\hat{q} = [q(1) \quad -q(2) \quad -q(3) \quad -q(4)]$;

rotate(a, $\hat{q}$):

result = quatProd(quatProd (q, [0 $a_{x\ filt}$ $a_{y\ filt}$ $a_{z\ filt}$]), $\hat{q}$);

result = [result (2) result(3) result(4)];

quatProd (a, b):

$$\begin{cases} ab(1) = a(1) * b(1) - a(2) * b(2) - a(3) * b(3) - a(4) * b(4) \\ ab(2) = a(1) * b(2) + a(2) * b(1) + a(3) * b(4) - a(4) * b(3) \\ ab(3) = a(1) * b(3) - a(2) * b(4) + a(3) * b(1) + a(4) * b(2) \\ ab(4) = a(1) * b(4) + a(2) * b(3) - a(3) * b(2) + a(4) * b(1) \end{cases}$$

5. ZUPT алгоритм

Алгоритм, который рассчитывает скорости с учетом статического и динамического этапов движения пользователя.

Входные параметры:

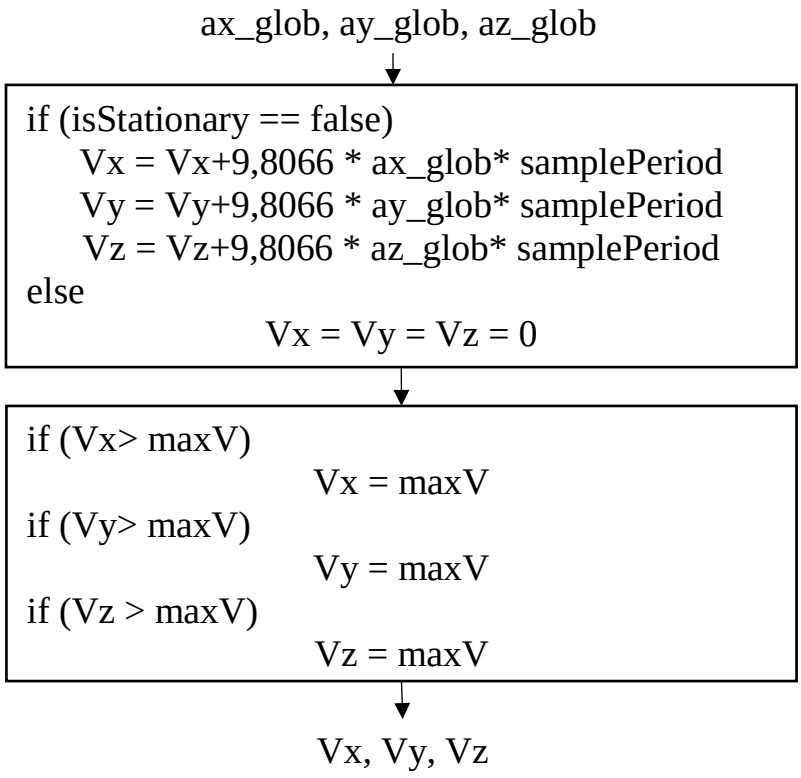
ax_glob, ay_glob,

az_glob	Ускорения в глобальной системе координат
samplePeriod	Период регистрации показаний сенсоров (в секундах)
isStationary	Признак движения
maxV	Максимальная скорость передвижения пользователя (по умолчанию равна 3,5 м/с)

Выходные параметры:

Vx, Vy, Vz	Скорости в глобальной системе координат

Псевдокод:



6. Trajectory detection алгоритм

Алгоритм определяет положение пользователя в данный момент времени и отображает его на карте.

Входные параметры:

Vx, Vy, Vz	Скорости в глобальной системе координат
samplePeriod	Период регистрации показаний сенсоров (в секундах)
adjCoef	Коэффициент, учитывающий неполноту регистрации шага по времени путем увеличения скорости
t_delta	Период выдачи показаний пользователю (в секундах)
mask	Бинарная маска помещения
pixelSize	Размер пикселя на карте
meshSteps	Шаг сетки на карте
masktable	Специальная таблица коррекции, записанная в виде вектор-столбца

Выходные параметры:

Pos	Положение пользователя [posX, posY, posZ]

Значения по умолчанию

adjCoef = 3 м

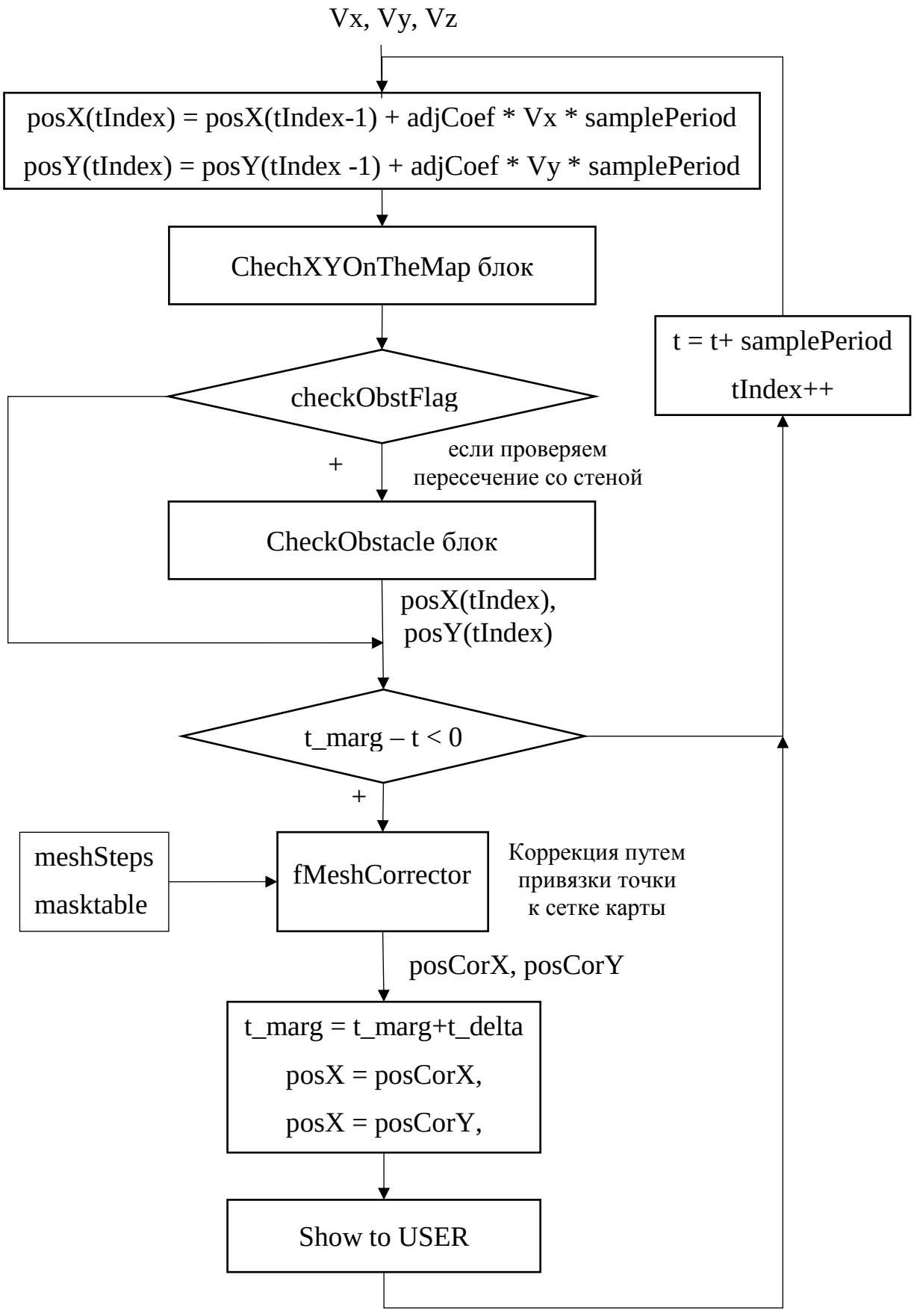
samplePeriod = 0.033 сек. (в Андройде между шагами может меняться)

t_delta = 1 сек.

t = initTime (время инициализации) = 5 сек. (при старте)

t_marg = initTime + t_delta (при старте)

tIndex=0


Vx, Vy, Vz
posX(tIndex) = posX(tIndex-1) + adjCoef * Vx * samplePeriod
posY(tIndex) = posY(tIndex -1) + adjCoef * Vy * samplePeriod
ChechXYOnTheMap блок
checkObstFlag
+
если проверяем пересечение со стеной
CheckObstacle блок
posX(tIndex), posY(tIndex)
t_marg – t < 0
+
meshSteps
masktable
fMeshCorrector
Коррекция путем привязки точки к сетке карты
posCorX, posCorY
t_marg = t_marg+t_delta
posX = posCorX,
posX = posCorY,
Show to USER
t = t+ samplePeriod
tIndex++

ChechXYOnTheMap блок

Функция проверяет, находится ли данная точка в пределах карты. Если выходит за пределы, то производится коррекция координат.

ПСЕВДОКОД

```
if posX(tIndex) < 0
    posX(tIndex) = 0;
if posY(tIndex) < 0
    posY(tIndex) = 0;
end
if posX(tIndex) > mapLength
    posX(tIndex) = mapLength;
end
if posY(tIndex) > mapWidth
    posY(tIndex) = mapWidth;
end
```

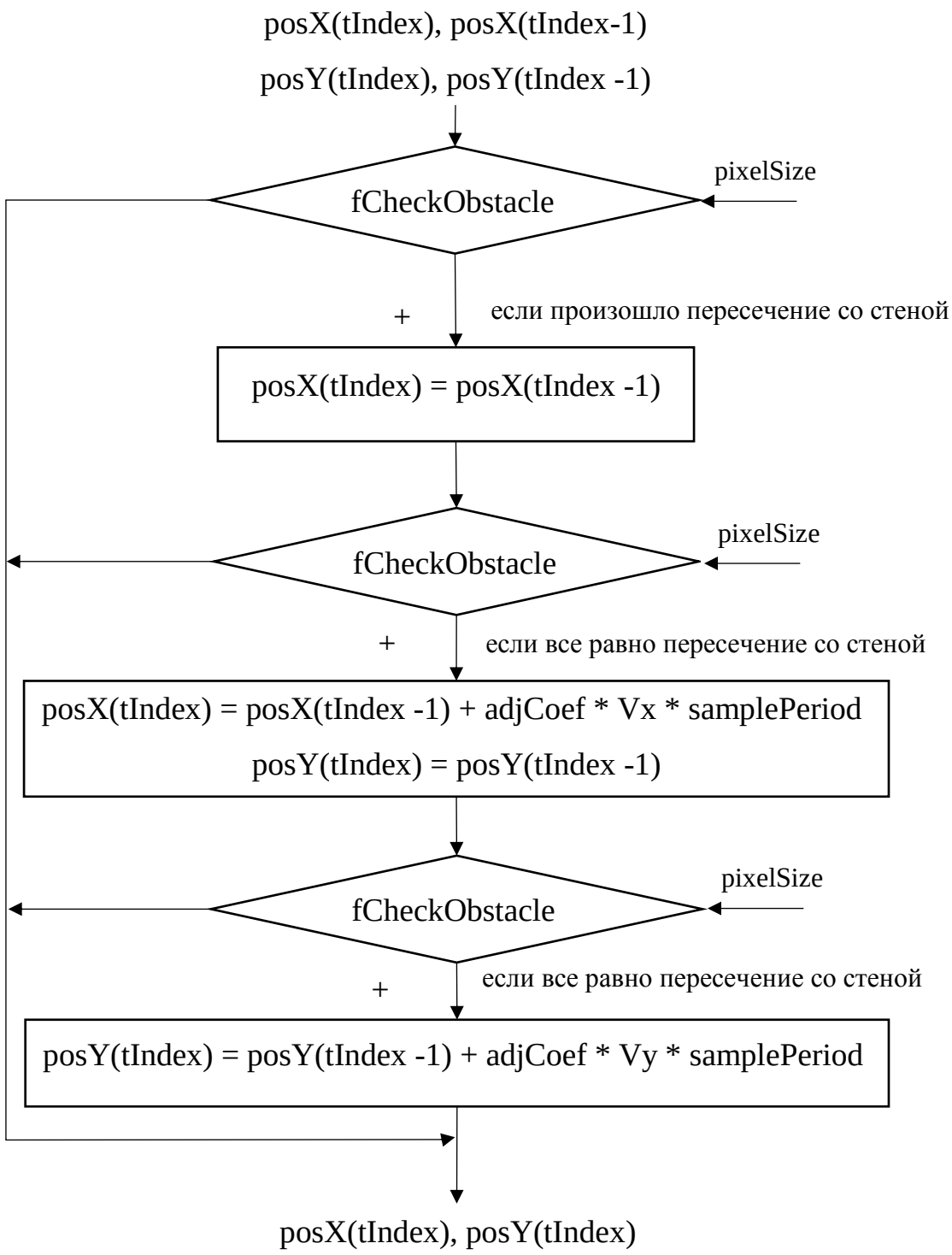
где `posX(tIndex)` и `posY(tIndex)` – соответствующие координаты положения в метрах;

`mapLength`, `mapWidth` – соответствующие размеры карты в метрах.

CheckObstacle блок

Если выставлен флаг "*checkObstFlag*", то блок проверяет, пересекает/попадает ли линия между предыдущим положением на карте и следующим рассчитанным положением на черную область (на маске). Если пересекает, то блок пытается корректировать текущее положение сначала по координате X, затем по координате Y. Если ни одна из корректировок не удалась и координаты X и Y все еще находятся в запрещенной зоне, то такое перемещение все-таки разрешается.

Функция "*fCheckObstacle*" строит прямую между предыдущей и текущей точками, определяет ее уравнение и проверяет каждую точку прямой на попадание в черную область. Возвращает true, если попадание произошло, и false – если прямая не пересекает препятствия.


posX(tIndex), posX(tIndex-1)
posY(tIndex), posY(tIndex -1)
fCheckObstacle
pixelSize
+
если произошло пересечение со стеной
posX(tIndex) = posX(tIndex -1)
fCheckObstacle
pixelSize
+
если все равно пересечение со стеной
posX(tIndex) = posX(tIndex -1) + adjCoef * Vx * samplePeriod
posY(tIndex) = posY(tIndex -1)
fCheckObstacle
pixelSize
+
если все равно пересечение со стеной
posY(tIndex) = posY(tIndex -1) + adjCoef * Vy * samplePeriod
posX(tIndex), posY(tIndex)