**Блок вторичной обработки - CLE**

Вторичная обработка реализована для системы с одним мастером-маяком и для любого числа ведомых маяков и любого числа меток (насколько позволяет темп излучения и aloha).

Блок вторичной обработки функционирует в две фазы:

* фаза синхронизации опорных маяков

Фаза, в которой ведомые опорные маяки синхронизуются по времени со шкалой мастера-маяка. На этом базируется принцип беззапросной архитектуры и без данной процедуры определение координат меток невозможно.

Для коррекции шкалы одного ведомого маяка относительно мастера необходимо знать время излучения сигнала мастера в шкале времени мастера, время приема в шкале времени ведомого маяка и геометрическое расстояние между маяками (фактически – координаты обоих маяков). Алгоритм синхронизации построен на базе линейного фильтра Калмана.

* фаза вычисления координат меток

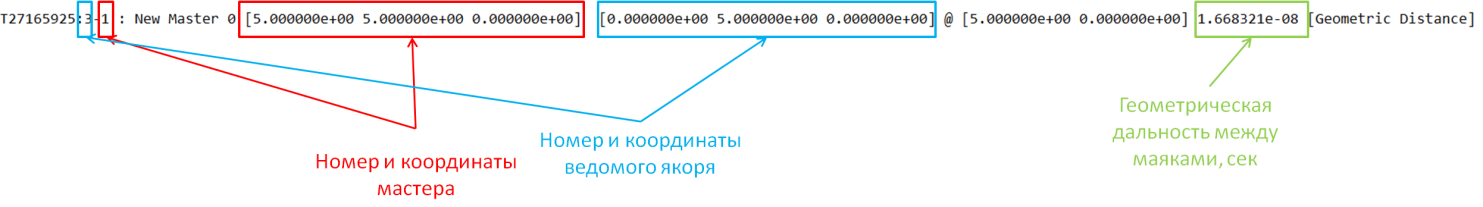
Непосредственно фаза, в которой по временам прихода blink-сигнала метки на все опорные маяки определяются ее координаты (задача мультилатерации). Это реализуемо только при условии синхронизации опорных маяков. Для определения координат используется метод наименьших квадратов.

У декавейва сделано в штатном режиме так: метки блинкуют 10 раз в секунду в динамике (в статике не блинкуют – проверяется по акселерометру), а синхронизация происходит с периодом 0,15 секунд. То есть мастер раз в 0,15 сек переключается с приемника на передатчик и не может в этим моменты принимать сигналы от меток. Остальные маяки (ведомые) – всегда приемники, принимают сигналы меток и синхросигналы мастера. У декавейва архитектура сделана таким образом, чтобы можно было масштабировать систему, делать субмастеры, зависящие от основного. В нашем случае пока реализовано только для одного мастера.

**Входная информация**

В блок вторичной обработки поступает 4 вида сообщений (на примере лога с ТТК):

* информация о новом маяке (New Master):



Это сообщение по идее отправляется один раз в начале работы системы при инициализации маяков для каждого маяка отдельно. Нужно для того, чтобы создать базу данных маяков, записать их номера (или ID) и координаты (без координат нельзя решить НЗ), а также расстояние между ведомым маяком и мастером в секундах (эта информация нужна для синхронизации).

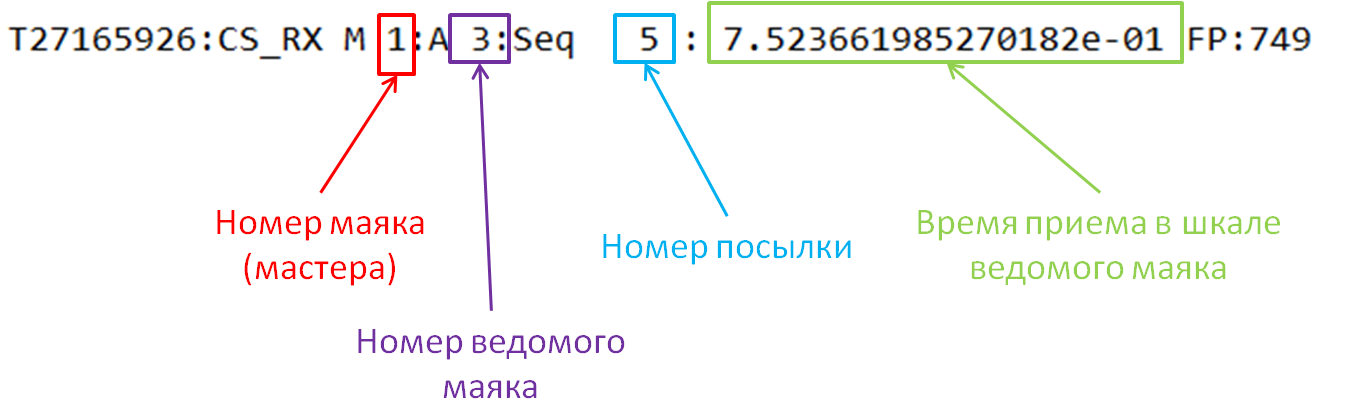
В ТТК информация о дальности является избыточной, так как ее можно вычислить и на этапе вторичной обработки. Кроме того, В ТТК информация о мастере передается в каждом сообщении такого типа, что тоже избыточно по идее.

* синхронизационный пакет мастера (CS\_TX)



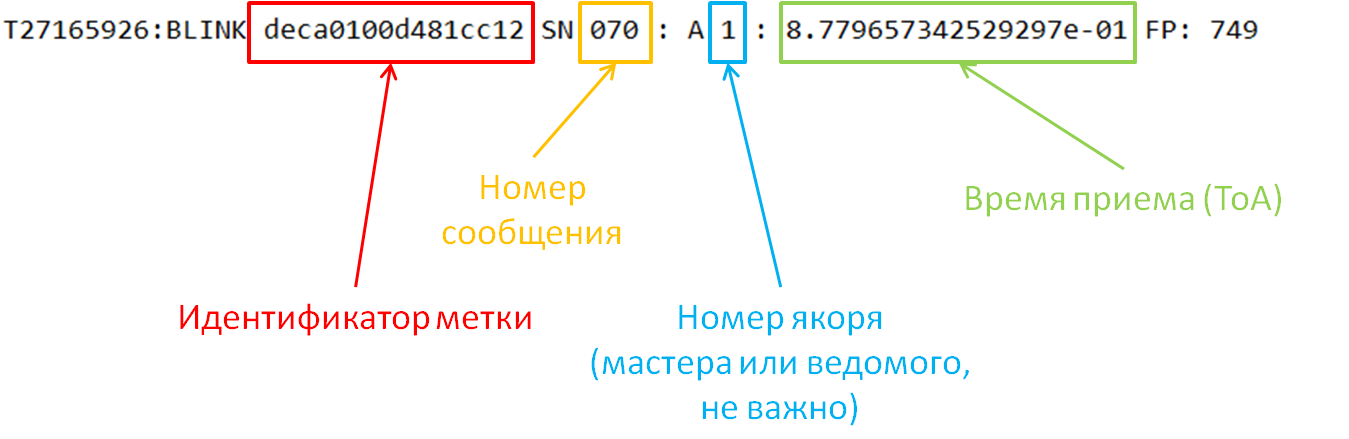
С определенным периодом (в ТТК 0,15 сек по умолчанию) мастер-маяк излучает такой синхроимпульс и отправляет в блок вторичной обработки сообщение, содержащее: номер мастер-маяка (на случай, если их больше одного), номер посылки (Seq), и время отправки в своей шкале времени.

* сообщение о приеме синхроимпульса от ведомого маяка (CS\_RX)



По принятии синхроимпульса от мастера-маяка ведомый маяк отправляет в блок вторичной обработки сообщение, содержащее: номер ведомого маяка, номер мастера, номер посылки (Seq), время приема синхроимпульса в шкале ведомого маяка. Номер посылки необходим для идентификации приема соответствующего синхроимпульса. Информация данного сообщения используется для синхронизации шкалы времени ведомого маяка со шкалой мастера-маяка.

* сообщение о приеме blink-сигнала метки



Сообщение которое каждый маяк (включая мастера) отправляет в блок вторичной обработки. Оно содержит номер маяка, номер посылки (это формирует метка), ID метки, и непосредственно время приема сигнала метки в шкале ведомого маяка.

Мое мнение: входные данные на первом этапе можно сделать как у декавейва, чтобы все работало без изменения. Что бы я изменил: вместо номеров якорей (они задаются у них в интерфейсе либо при создании конфигурационного файла пользователем, присваиваются номера и тд) использовал бы их ID, подобно таким, которые используются для меток (типа decaXXXXXXXXXXX). Это позволит избежать проблем с задания соответствия между номерами маяков и их айдишниками. По мне – это просто лишние избыточные сущности. По факту, информация которую я не выделил в сообщениях - не нужна (кроме идентификаторов сообщений – New Master, CS\_TX, CS\_RX, BLINK).

**Выходная информация**

Пока что на выходе: ID метки + ее координаты. Формат можно сделать любой. Можно добавить время, геометрический фактор, какой-нибудь показатель качества. Выдача информации происходит по мере ее поступления (координаты рассчитались – сразу на выход).

**Архитектура блока вторичной обработки**

Есть основной объект – **config**. В первую очередь он содержит в себе базу данных всех маяков и меток в виде списков. Пока список маяков пустой или в нем недостаточное их количество, никаких координат меток не будет, поэтому надо сперва его заполнить (это сообщения типа New Master).

Кроме этого, конфиг содержит разного рода необходимые параметры:

* dw\_unit в секундах
* T\_max величина переполнения таймера в секунлах
* cur\_seq – текущее значение номера синхронизационной посылки
* с – скорость света
* zone – это размер зоны (для проверки вылета решения за границы)

В блок вторички поступают сообщения в формате строк, они описаны ранее. Эти строки парсятся в объекте класса Message. Там им присваивается тип (type), который может принимать четыре значения (в соответствии с типами сообщений): “NM”, “CS\_TX”, “CS\_RX”, “BLINK”. Также есть флажок state, принимающий значение 0, если типа сообщения не совпадает ни с одним из заданных.

В случае, если тип сообщения “NM”, поля объекта класса заполняются следующим образом: в поле Master записывается номер мастера маяка, в поле Anchor – номер ведомого маяка, в Master\_x, Master\_y, Master\_z – координаты мастера, в Anchor\_x, Anchor\_y, Anchor\_z – координаты ведомого маяка. В поле R записывается расстояние между маяками в секундах.

В случае, если тип сообщения “CS\_TX”, поля объекта записываются следующим образом: в поле Anchor – номер мастера, в поле Seq – номер посылки, в поле TimeStamp – время отправки.

В случае, если тип сообщения “CS\_RX”, поля объекта записываются следующим образом: в поле Anchor – номер маяка, в поле Seq – номер посылки, в поле TimeStamp – время отправки.

В случае, если тип сообщения “BLINK”, поля объекта записываются следующим образом: в поле Anchor – номер маяка, в поле SN – номер сообщения, в поле TimeStamp – время прихода (TOA), в поле ID – идентификатор метки.

Далее, в зависимости от типа сообщения, оно обрабатывается соответствующей функцией: **process\_NM, process\_CS, process\_BLINK.**

**process\_NM(mes, config)**

При приеме соответствующего сообщения создается объект класса **Anchor** со следующими полями:

number – номер маяка

x –

y –

z –

Range – расстояние до мастера в секундах (для ведомых)

sync\_flag – 1, если маяк синхронизирован с мастером (при создании объекта 0)

need\_to\_sync – 1, если при пришел синхронизационный пакет, но шаг синхронизации фильтром калмана еще не произведен, 0 - если синхронизация уже прошла либо не требуется (по умолчанию – 0)

X – вектор состояния для синхронизационного фильтра калмана (столбец 2х1)

Dx – матрица ошибок фильтрации для синхронизационного фильтра калмана (2х2)

T\_rec – время приема синхросигнала мастера (для ведомых маяков), время излучения для мастера

T\_tx – время излучения мастера на прошлом шаге синхронизации

master – 1 для мастера, 0 для ведомого

Соответствующий созданный объект добавляется в список config.anchors. Мастер в этом списке всегда первый.

**process\_CS(mes, config)**

1) Вначале проверяется соответствие номера посылки (Seq) текущему номеру посылки. Прием нового номера означает новый шаг синхронизации, а значит на предыдущем шаге все ведомые маяки синхронизировались, поэтому все флаги need\_to\_sync устанавливаются в положение 0.

2) Далее для маяка, от которого пришло сообщение, флажок need\_to\_sync поднимаем и записываем TimeStamp в поле T\_rec.

3) Шаг синхронизации для ведомого маяка фильтром калмана происходит только тогда, когда пришел синхронизационный пакет от данного маяка и от мастера маяка (флажки need\_to\_sync подняты и там и там). При выполнении данного условия делаем шаг синхронизации **CS\_filter** и опускаем флажок need\_to\_sync для ведомого маяка.

Шаг синхронизации происходит, когда флаг sync\_flag находится в состоянии 1, когда фильтры уже инициализированы (штатный режим работы). По умолчанию флаги sync\_flag для всех фильтров 0. Поэтому при первом приеме синхронизционных пакетов сперва происходит инициализация фильтра и поднимается флажок sync\_flag.

Все процедура на картинке:

**process\_BLINK(mes, config)**

Тут все просто. Когда приходит сигнал типа BLINK, мы сначала пробегаем по списку всех сопровождаемых меток в config.tags и смотрим, есть ли метка с таким ID в списке.

Если такой метки нет, то добавляем в список новую метку – объект типа **tag**, у которого есть следующие поля:

ID – идентификатор

SN – номер сообщения

measurements – список измерений в виде сообщений типа BLINK

x –

y –

ToT – время излучения

h – высота (не оценивается, известная изначально)

state – флаг, можно использовать как состояние метки (пока не используется)

file – объект лог-файла, куда пишутся координаты

Если метка с таким ID есть, то мы кладем в нее новое пришедшее измерение.

Методы:

add\_meas – добавление нового измерения в список

Если сменился SN, то считаем координаты для предыдущего SN, очищаем список измерений и после этого добавляем туда новое измерение. Если не сменился, то просто кладем новое измерение в список.

coords\_calc – расчет координат метки

Вызывается, когда сменился SN. Если измерений больше двух (для решения в 2Д методом МНК требуется минимум 3 измерения времени прихода), формируем массивы наблюдений PD и координат маяков SatPos, а также начальное приближение Init для МНК и вызываем функцию расчета координат solver\_pd, которая инвариантна к числу маяков.

Функция solver\_pd возвращает флаг b, равный true, если решение сошлось, и false, если по каким-то причинам решение не сошлось, а также вектор X (3х1) с координатами метки и временем излучения.

**Точка входа CLE** – в функции main вместо считывания файла подсовывать сообщения mes через какой либо интерфейс.

**Точка выхода CLE** – в функции add\_meas при условии флага = true (удачном расчете координат метки), отправляем эти координаты по соответствующим интерфейсам.