Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Кафедра системного программирования

Захаров Глеб Игоревич

Реализация многогипотезного фильтра Калмана высокоточного позиционирования

Курсовая работа

Научный руководитель:
ст. преп. Смирнов М. Н.

Санкт-Петербург
2018

Оглавление

Введение

Системы позиционирования с дециметровой точностью используются в различных областях, таких как геодезия, градостроительство, картография, сельское хозяйство, прокладка железнодорожных автомобильных магистралей, мостов и других сооружений на дорогах, нефте- и газопроводов, линий электропередач и связи, а с недавнего времени используется и в беспилотных автомобилях.

Суть технологии — в уточнении сигнала, получаемого с навигационных спутников систем GNSS (Global Navigation Satellite System, например таких как ГЛОНАСС, GPS, Beidou, QZSS и Galileo), за счет их обработки наземными аппаратно-программными комплексами - сетями референтных станций или другого статичного приемника в режиме «базы». Второе называется относительным позиционированием.

В относительном позиционировании с дециметровой точностью участвуют два приемника: база и ровер. Пока первый приемник неподвижно стоит на точке и принимает сигналы со спутников, обрабатывает и отсылает по определенным каналам связи свое местоположение и поправки, второй приемник может стоять неподвижно (режим *static*) или двигаться (режим *kinematic*). Если приемники находятся на расстоянии меньше 10-20 км, главные ошибки GNSS сигналов, такие как тропосферные, ионосферные ошибки, смещения тактового сигнала и орбитальные ошибки спутников становятся одинаковы.

По сигналу, полученному от спутника, можно узнать фазовый сдвиг. Если бы приемник «знал» целое количество длин волн, которое помещается между ним и спутником, ошибка измерения расстояния в среднем составляла бы всего 5 мм. Поэтому приемнику нужно разрешить задачу *фазовой или целой неопределенности.*

Компания Emlid [16] занимается разработкой дешевых приемников точного позиционирования. Благодаря свободному ПО, их приемники дешевле аналогов в разработке на 1-2 порядка. На приемнике Emlid Reach [15] для разрешения неопределенности используется пакет программ rtklib [10] с открытым исходным кодом. Для разрешения фазовой

неопределенности в rtklib реализован фильтр Калмана. Это рекурсивный алгоритм, который по получаемому вектору измерений предсказывает вектор состояний физической системы. В данном случае вектор состояний - это позиция и скорость приемника, а также фазовые невязки для всех спутников.

Относительное позиционирование не избавляет полностью от всех ошибок, из-за чего фильтр Калмана может сходиться к неправильным результатам. Значит, и неопределенность разрешится неправильно. Ошибка может достигать нескольких метров.

Во многих областях применения точного позиционирования (таких как геодезия, беспилотные автомобили и строительство), неточность даже в 1 дециметр может сыграть существенную роль. По статистике пользователей rtklib, примерно 5% тестов получаются неточными. Поэтому возникла идея сделать фильтр более надежным.

Таких ошибок можно избегать путем добавления новых измерений от датчиков. Однако точные датчики (например, двух- или трехчастотные антенны) стоят больших денег и их почти не получается применить не компаниям, а обычным пользователям или исследовательским группам. Поэтому задачу уточнения фильтра нужно решать программным способом за счет ресурсов приемника.

Это можно сделать, так как сигналы с приемников поступают с маленькой частотой (5-10 Гц), поэтому у приемника остается много процессорного времени.

Одно из способов решения задачи — это *многогипотезный* фильтр Калмана. Алгоритм запускает несколько фильтров с разными предположениями о характере движения и с разными настройками, и на их основе выбирает лучший вектор состояний. Отдельный вопрос, как определить, что значит «лучший» вектор и как проверить улучшение точности алгоритма, так как в этой предметной области узнать, что решение неточное можно только по косвенным признакам.

1. Цель и постановка задачи

Итак, нужно реализовать многогипотезный фильтр для приемника Emlid Reach и проверить его эффективность в точности определения позиции приемника на тестах с плохими условиями. Для этого нужно:

1. разработать набор тестов, по которому можно определить, насколько точным получается алгоритм;
2. автоматизировать тестирование алгоритмов позиционирования для библиотеки rtklib;
3. выбрать метрику для оценки алгоритмов;
4. разработать многогипотезные фильтры с разными критериями выбора вектора состояний;
5. сделать эксперимент, сравнить результаты и выбрать наилучшую стратегию;
6. протестировать алгоритмы на целевом приемнике Emlid Reach.

2. Обзор

В [14] подробно объясняется принцип действия фильтра Калмана в библиотеке rtklib. В книге [5] написано, как реализуется фильтр для задач точного позиционирования.

Далее будет объяснен принцип действия фильтра и разрешения фазовой неопределенности, а затем будут приведены существующие решения.

2.1. Глоссарий

2.1.1. Описание фильтра и нахождение точной позиции

Приемник получает сигнал от базы и других спутников с заданной частотой. Время и вся информация из такого сигнала называется *эпохой*. Из эпохи приемник может узнать *псевдодальность*, то есть примерное расстояние до спутника и примерное количество длин волн до спутника. Из этих данных с помощью формулы для вычисления двойных разностей, может быть получена позиция ровера относительно базы. *Двойными* разности называются потому, что считаются для ровера и базы между 2-мя спутниками, один из которых называется «опорным». Формулы для двойных разностей такие:

$$\begin{aligned} \Phi_{rb}^{jk} &= \rho_{rb} + \lambda(B_{rb}^{j} - B_{rb}^{k}) + \varepsilon_{\Phi} \\ P_{rb}^{jk} &= \rho_{rb}^{jk} + \varepsilon_{P}, \end{aligned}$$

где Φ_{rb} — примерное расстояние в длинах волн между ровером и базой, P_{rb} — псевдодальность между ровером и базой, ρ_{rb} — настоящее расстояние, B_{rb} — разность количества длин волн от одного спутника (j или k, например) между базой и ровером, ε — шум сигнала.

Фильтр Калмана - рекурсивный алгоритм для оценивания динамически меняющегося вектора состояния $\hat{x}$ с помощью неточных измерений y в момент времени t_k.

Алгоритм работает в два этапа. На этапе прогнозирования фильтр

Калмана экстраполирует значения переменных состояния, а также их неопределенности. На втором этапе, по данным измерения (полученного с некоторой погрешностью), результат экстраполяции уточняется. Благодаря пошаговой природе алгоритма, он может в реальном времени отслеживать состояние объекта (без заглядывания вперед, используя только текущие замеры и информацию о предыдущем состоянии и его неопределенности).

Далее написаны формулы сначала этапа прогнозирования, а потом уточнения. В них $(+)$ обозначает состояние после коррекции фильтра с помощью измерения, $(-)$ обозначает состояние до коррекции, P — ковариационная матрица вектора x, а $h(x)$, $H(x)$ и R — вектор модели измерений, его матрица частных производных и ковариационная матрица ошибок измерений соответственно, F и Q — матрицы перехода и ковариации системы.

Этап прогнозирования:

$$\begin{aligned} \hat{x}_k(-) &= F^k_{k-1}\hat{x}_{k-1}(+) \\ P_k(-) &= F^k_{k-1}P_{k-1}(+)(F^k_{k-1})^T + Q^k_{k-1} \end{aligned}$$

Этап коррекции:

$$\begin{aligned} \hat{x}_k(+) &= \hat{x}_k(-) + K_k(y_k - h(\hat{x}_k(-))) \\ P_k(+) &= (I - K_k H(\hat{x}_k(-)))P_k(-) \\ K_k &= P_k(-)H(\hat{x}_k(-))(H(\hat{x}_k(-))P_k(-)H(\hat{x}_k(-))^T + R_k)^{-1} \end{aligned}$$

Здесь вектор состояний

$$x = (r^T, v^T, B_1^T, B_2^T, B_5^T)^T,$$

где r — позиция приемника, v — скорость, а $B_i = (B_i^1, \ldots, B_i^m)$ — одиночные разности от $k = 1, \ldots, m$ спутника между ровером и базой с частотой L_i.

Вектор измерений выглядит так:

$$y = (\Phi_1^T, \Phi_2^T, \Phi_5^T, P_1^T, P_2^T, P_5^T)^T,$$

где $\Phi_i = (\Phi^{12}_{rb,i}, \ldots, \Phi^{1m}_{rb,i})$ — двойные разности между спутниками 1 и m примерных чисел длин волн между ровером и базой, а P, соответственно двойные разности псевдодальностей. $H(x), h(x), R(x), Q(x), F(x)$ определяются исходя из этих векторов.

2.1.2. Разрешение фазовой неопределенности

Когда фильтр Калмана получил вектор x с помощью прогнозирования, можно разрешить фазовую неопределенность [14]. Сначала вектор x трансформируется в вектор двойных разностей. Затем ищется такой вектор $\hat{N} = (N^1, \ldots, N^m)^T$, где N^i — целое, который минимизирует

$$F(N) = (N - \hat{N})^T Q_N^{-1} (N - \hat{N}),$$

где Q_N — ковариация B_i в терминах двойных разностей.

Для решения этой проблемы в rtklib используется алгоритм LAMBDA [12] и его модификация MLAMBDA [1]. Алгоритм трансформирует линейное пространство, чтобы сократить количество решений, а затем с помощью поиска по дереву ищется решение, которое удовлетворяет соотношению $F(N_2)/F(N_1) > R_{threshold}$, где N_2 — второе из подходящих решений, а $R_{threshold}$ — константа.

Если в текущей эпохе такое решение N_1 находится, его называют *fix*, иначе *float*.

Как только fix нашелся, rtklib переходит к режиму *hold*, в котором вектор измерений y начинает считаться с помощью $\hat{N}$ путем добавления искусственных измерений. Такой способ расчета хорошо подходит для режима kinematic, например, для езды автомобиля по прямой.

Так как MLAMBDA — эмпирический алгоритм, при неправильной сходимости фильтра Калмана решение может быть неправильным, значит, и в режиме hold позиция будет неправильной. И наоборот, если неопределенность разрешилась правильно, то в режиме hold будут более точные измерения.

2.1.3. Многогипотезный фильтр Калмана

Многогипотезный фильтр Калмана - это параллельный алгоритм, который запускает несколько фильтров Калмана с разными настройками. Количество гипотез может быть постоянным или нет. Как написано в книге [5], из всех фильтров - *гипотез* - можно выбрать наилучший fix, если точно понятно, какая из гипотез лучше, остальные при этом отбрасываются. Для этого перед фильтром нужно реализовать алгоритм, который отклоняет все неподходящие гипотезы на этапе коррекции.

Взвешенный fix принимает все векторы измерений, взвешенных с какими-то вероятностями p_i, где $\sum p_i = 1$, но при этом рассчитывает только 1 вектор состояний. Примером может послужить Probalistic Data Association Filter (PDAF) [2], у которого этап прогнозирования такой же, как у фильтра Калмана, а коэффициент Калмана — K_k, в случае фильтра, становится многими коэффициентами $K_{i,k}$, и на этапе коррекции появляется взвешенная сумма этих коэффициентов.

Еще один способ — многогипотезный фильтр Калмана. В отличие от предыдущих, у многогипотезного фильтра существует несколько векторов состояний и несколько гипотез, которые берутся из банка. Когда через некоторое количество итераций фильтра становится понятно, какие из гипотез точны, а какие нет — тогда неточные гипотезы исключаются из банка и расчетов. Эта техника изначально была применена в компьютерном зрении, и называлась MHT [11]. Перед использованием таких фильтров нужно ответить на вопросы, перечисленные ниже.

1. При каких условиях исключать гипотезы?

2. В каких условиях добавлять гипотезы?

3. Как отделить точные гипотезы от неточных?

Эти вопросы нужно решать с помощью знаний о предметной области.

На этапе коррекции банк делится на $n * l$ гипотез и *выходную* гипотезу. Затем все n векторов измерений комбинируются и считается l векторов состояний. Затем сразу после коррекции количество гипотез

уменьшается до l штук путем *сливания (merging)* или простым удалением гипотез. Точный подход различается в разных реализациях. Обычно гипотезы с наименьшим весом сливаются со своими соседями в пространстве измерений, или самые маловесящие гипотезы просто удаляются.

На выход подается 1 из векторов состояний, или они взвешенно комбинируются.

2.1.4. Из-за чего фильтр не сходится

Для того, чтобы считать одиночные разности в векторе состояний и измерений в rtklib, лучше всего брать измерения от базы и ровера в одну эпоху, т. е. в один момент времени [14]. Однако, приемники не идеально синхронизированы из-за различных часовых невязок. Кроме того, частота прихода эпох от приемников обычно различается: например, 5 Гц для ровера и 1 Гц для базы. Rtklib для контроля одиночных разностей выбирает последнее измерение базы, меньшее или равное времени эпохи ровера. Разность времен между базой и ровером называется *возрастом базы* или *age of differential*. С ростом разности времен, точность решения постепенно уменьшается из-за дрейфа часов приемников и ионосферного слоя.

Иногда сигнал от базового приемника может пропадать, и если возраст базы больше нескольких секунд, точное решение в отдельных условиях (при плохой погоде, например) не сможет найтись. Также возраст базы может быть постоянный при приеме поправок по медленным каналам, по радио, например.

Другой случай — если антенна приемника видит не все небо, например, при поездках на узких улицах, некоторые сигналы отражается от зданий, и получаются outlier'ы. В избавлении от outlier'ов сейчас ведется работа [7].

С помощью многогипотезных фильтров можно сгладить эти плохие условия. Поэтому возникла идея создать фильтр для минимизации возраста базы и для уменьшения вреда от outlier'ов.

2.1.5. Тестирование

Из-за трудоемкости записи данных и трудности воспроизведения плохих условий, трудность тестирования возрастает. В статье [13] указано, что можно эмулировать сигналы от приемников. Также можно сохранять реальные тесты и накапливать базу тестов.

2.2. Существующие решения уточнения фильтра

На данный момент реализация многогипотезного фильтра в библиотеках с открытым исходным кодом для точного позиционирования не найдена. Так, например, в [4] реализован расширенный фильтр Калмана, который аппроксимирует реальный шум гауссовым.

Существует реализация многогипотезного фильтра Калмана для кубка роботов [8]. В работе сравнивается многогипотезный и взвешенный фильтры, 1-ый в среднем точнее на 9 см. Значит, для большей точности нужно считать несколько векторов состояний.

В еще одной работе по кубку роботов [9] создали фильтр с динамическим добавлением множества гипотез, их общее количество может быть любым. В нашей работе нельзя создавать в реальном времени много разных гипотез, так как целевая платформа может не справиться с вычислениями (один цикл фильтра работает в среднем 10-20 мс., а сигнал приходит каждые 100 мс.). Поэтому количество одновременных подсчетов фильтра должно быть ограничено до 5.

Есть работа, в которой запускаются 2 фильтра Калмана [3], и берется среднее из их результатов. Результат получается меньше по дисперии. Значит, даже удвоение фильтра с разными настройками может дать результат.

Для навигации внутри помещений [6] также можно использовать многогипотезный фильтр. В работе гипотезы создаются каждый раз, когда приемник меняет направление движения (поворачивается на 90 или 180 градусов), а если несколько раз приемник повернул не туда, то гипотеза удаляется. В спутниковой навигации проверить надежность гипотезы можно только спустя какое-то время. При неправильном fix

решении фазовые невязки постепенно будут расходиться.

С другой стороны, создавать гипотезы будет сам rtklib, так как каждый цикл он будет пытаться разрешить фазовую неопределенность. Остается только воспользоваться разными настройками.

3. Автоматизация создания тестового набора

Для тестирования системы существует инструмент rtklib-testing-suite, который умеет создавать идеальные тесты - *логи*, зашумлять их с помощью случайных выборочных шумов, гауссовских или настоящих, но записанных из другого лога, а затем строить по ним статистику, такую как среднюю величину невязки, процент «точного» определения fix позиции (то есть меньше 10 см), ошибку наименьших квадратов в плоскости и по высоте, а также другие характеристики. Этот проект написан на языке Python и до сих пор находится в разработке.

Так как мы знаем настоящую точку нашего искусственного теста, мы без труда можем сравнить результаты зашумленного теста с этой позицией. К сожалению, это работает только для static тестов.

Для того, чтобы полнее проверить алгоритмы, мы решили реализовать тестирование на реальных kinematic данных. Идея в том, чтобы проверять наш алгоритм только на тех участках лога, где мы уверены в точности решения фильтра Калмана. Например, в те моменты, когда приемник перед началом движения долго стоит на месте и при этом позиция, к которой сошелся фильтр, не меняется. Для такого тестирования мы ввели *опорные* и *плохие интервалы* — это такие отрезки времени, на которых мы уверены в решении или точно знаем, что оно неправильное. Теперь мы можем сравнивать текущую позицию какого-то эталонного решения с нашим решением на опорных интервалах, и смотреть на поведение сходимости фильтра на плохих интервалах.

Также существовала проблема медленного подсчета большого количества тестов, т. к. это трудоемкий процесс. В постобработке на компьютере часовая запись обрабатывается около 1 минуты. Раньше в этом проекте только расчет решения запускался параллельно, 200 часовых тестов он обрабатывал 50-60 минут, а затем считал статистику в одном процессе еще 50-60 минут. Я реализовал параллельный расчет статистики, теперь расчет статистики занимает примерно в 3 раза меньше времени. Кроме того, я реализовал опцию continue, которая позволяла

прервать программу и потом продолжить расчет с последнего незаконченного теста.

UML-диаграммы подпрограммы, отвечающей за подсчет статистики до и после реализации функций, представлены на рис. 1 и 2.

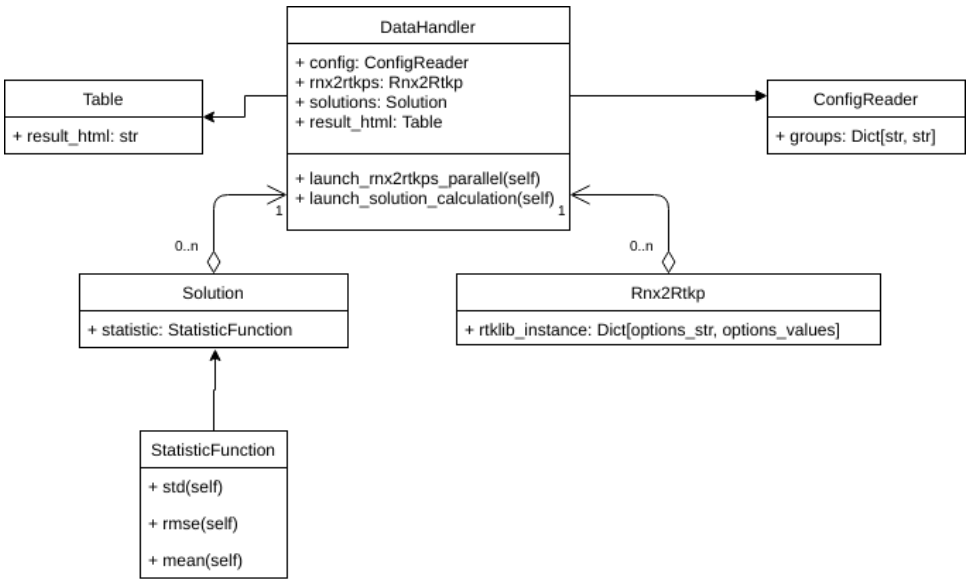

Рис. 1: UML-диаграмма «до».

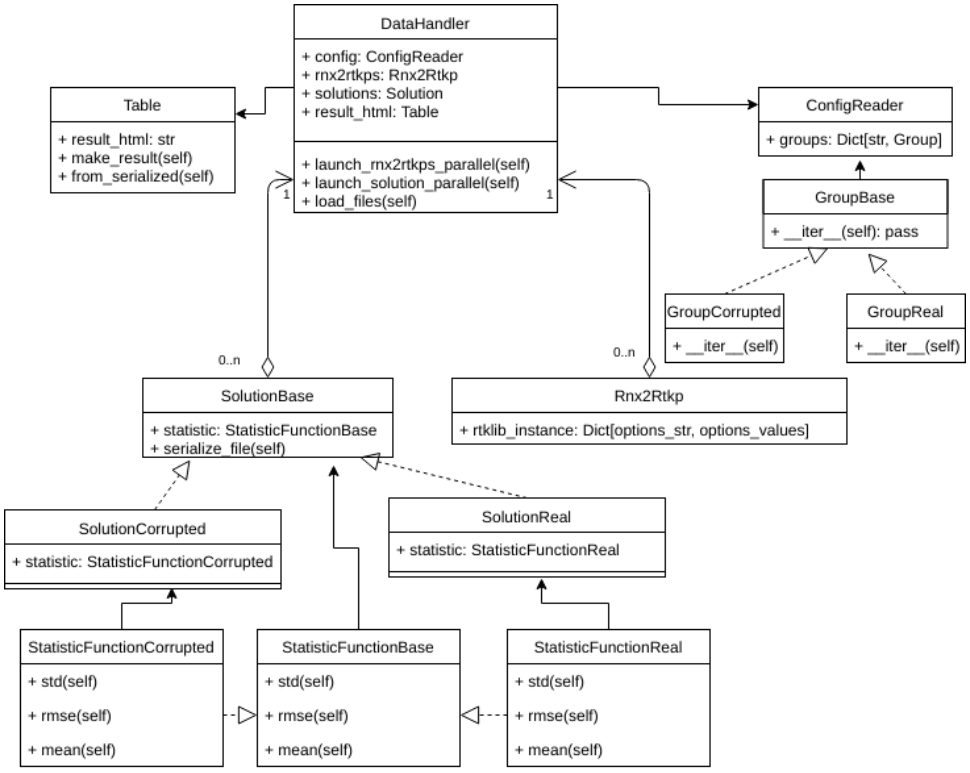

Рис. 2: UML-диаграмма «после».

Для реализации опции continue был выбран модуль стандартной библиотеки Python pickle. Он сериализует всю статистику о решениях (классы, наследующие абстрактные классы SolutionBase и StatisticFunctionBas и затем при прерывании программы все подсчитанные результаты можно будет воспроизвести.

Класс Table, который создает таблицу с результатами в формате html, был переписан с помощью библиотеки pandas. Это было необходимо для последующего удобного добавления опций расчета, т. к. для kinematic и static тестов должны были рассматриваться разные характеристики. В классе ConfigReader существовал словарь разных «групп» тестов — зашумленные тестовые данные, созданные из одного «идеального» лога. Однако для kinematic тестов используется другая структура директорий, поэтому пришлось абстрагировать интерфейс доступа к группам. Теперь каждый новый класс «групп» тестов должен реализовать несколько методов, в том числе итерацию сквозь все тесты и проверку на правильность расположения файлов в директории.

Еще пришлось абстрагировать класс Solution, потому что он сравнивал тесты с 1 эталонной точкой и не умел работать с «опорными» интервалами. Соответственно, пришлось абстрагировать и класс StatisticFunction, который может считать различную статистику для решения.

Также для создания искусственного лога пользователю необходимо было написать скрипт, в нем указывалось, как именно нужно зашумлять этот лог. Такая система не рассчитана на создание большого количества тестов, т. к. для создания 50 тестов нужно написать 50 скриптов по 2-3 строки. Поэтому я написал скрипт, который по входным опциям (например, гауссовский шум по фазе 0.5, 1 и 2 см и синусоидальный шум по псевдодлине 1, 2 и 3 м) генерирует их сочетания и автоматически создает скрипты для последующего зашумления.

Также в rtklib была добавлена симуляция большого возраста базы. Это было необходимо для проверки одного из многогипотезных фильтров.

Для метрики был выбран процент «точных» (меньше 10 см) решений.

4. Многогипотезные фильтры

Во-первых, было реализовано 2 способа проверки точности fix решения. fix считается хорошим, если он держится достаточно долго по времени и фазовые невязки не увеличиваются по времени. Для решения первой задачи была создана очередь - буфер истории, в которой сохраняется тип решения, позиция и скорость, а также все фазовые невязки для всех спутников.

Процент fix решений считается как N_{fix}/N_{all}, и если это число больше C_1, которое можно установить, то fix решение считается надежным по длительности.

Для проверки фазовых невязок считается RMSE ошибка, которая есть корень из суммы квадратов всех невязок поделенной на их количество. Если эта величина больше какого-то барьерного значения, то гипотеза отбрасывается как неправильная. Обычно так происходит из-за специфики добавления целых длин волн к вектору измерений. Когда решение начинает расходиться, вектор измерений перестает быть похожим на тот, который предлагает rtklib в режиме hold, и фазовые невязки постепенно становятся большими.

На рис. 3 можно увидеть алгоритм работы в виде блок-схемы.

Затем было реализовано 2 многогипотезных фильтра. Первый, по задумке, должен решать проблему многолучевости в городах: выбраны n гипотез с увеличивающимся углом обзора спутников. Для них всех считаются вектора состояний. Пока они float, выбирается среднее арифметическое из всех гипотез. Как только для какой-то гипотезы фазовая неопределенность разрешена, это fix решение начинает проверяться по длительности и фазовой неопределенности. Если такой фикс оказался надежным, то выбирается он. Если нет, то решение остается float, и выбирается среднее из оставшихся фильтров. Если найдены 2 и более надежных fix гипотез, то берется среднее из них. Таким образом, банк гипотез состоит из fix и float гипотез, все из них считаются (то есть $l = n$ в терминах многогипотезного фильтра), а выбирается взвешенное среднее из всех фильтров с добавлением или удалением гипотез.

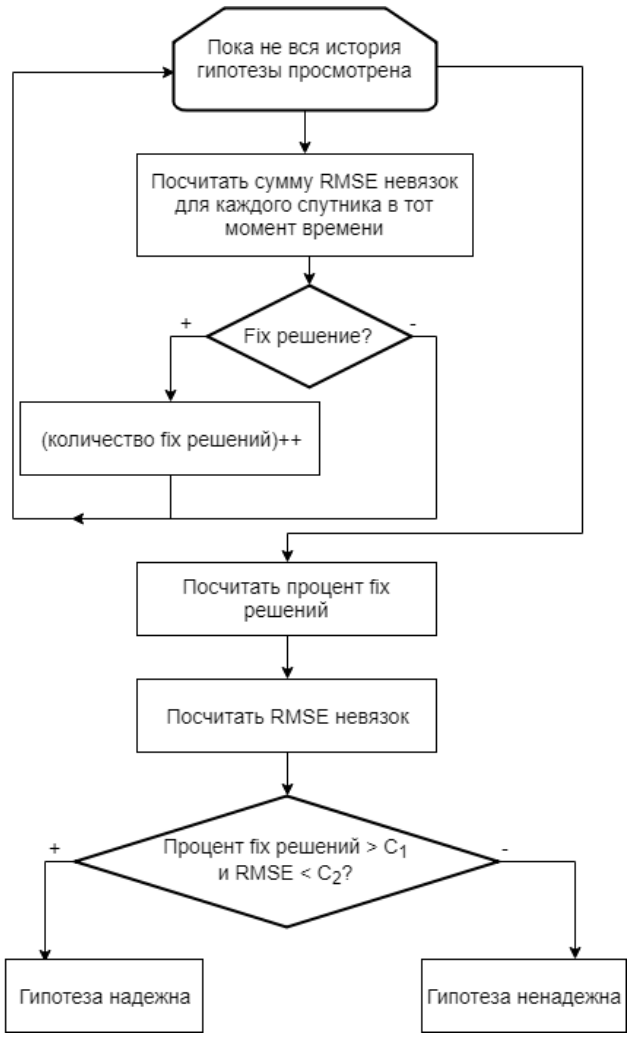

Рис. 3: Алгоритм проверки гипотезы на надежность.

Такой фильтр может помочь избавиться от многолучевости, так как обычно она происходит от низких спутников, сигнал от которых отражается от зданий. Однако, чем больше мы сужаем угол обзора, тем меньше используется спутников и, соответственно, тем меньше измерений получает фильтр. Решено было сделать 3 гипотезы с углами видимости 5, 15 и 25 градусов относительно горизонта. 15 градусов выбрано из стандартных настроек rtklib, 25 градусов — так как иначе будет слишком мало спутников.

На рис. 4 можно увидеть принцип работы фильтра. Пока все гипотезы — float решения или ненадежные fix решения, выбирается среднее из float позиций. Когда же хотя бы одна fix гипотеза проходит проверку на надежность, фильтр выбирает ее и удаляет остальные ненадежные

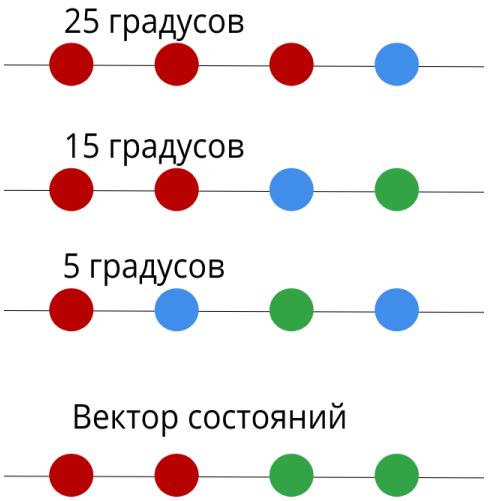

Рис. 4: Пример работы. Красные точки — float решения, синие и зеленые — ненадежные и надежные fix решения.

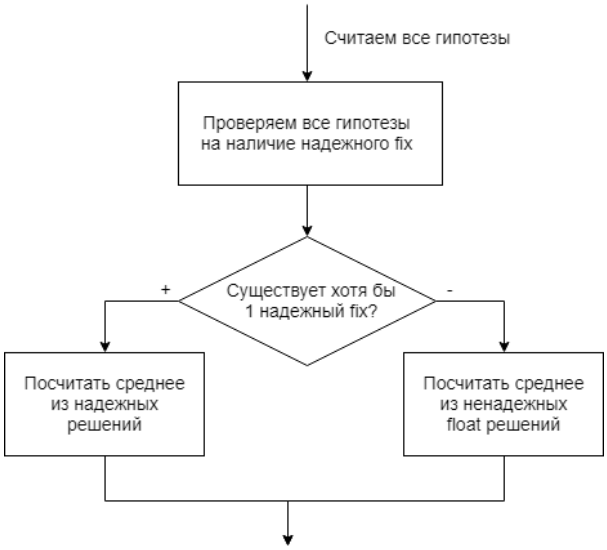

Рис. 5: Алгоритм работы фильтра.

Рис. 6: *Усредненный* фильтр.

гипотезы.

Второй фильтр, который получилось реализовать — *догоняющий*, он должен решать проблему с большим возрастом базы. Есть 2 гипотезы, одна из них считается по самым новым измерениям, как обычный фильтр Калмана, другая считается по последним измерениям базы. Для этого нужно было создать очередь из измерений ровера и базы. В момент, когда приходят новые измерения, вторая догоняющая гипотеза сравнивает новые базовые наблюдения с последними, и если они новее, то добавляет их в очередь базы. Таким образом, если пришли новые измерения базы, то они окажутся в очереди. Затем 2-ая гипотеза проверяет возраст базы для следующих измерений ровера в буфере путем выбора минимального возраста, и если он меньше какого-то барьерного значения, то 2-ая гипотеза делает шаг.

В момент, когда на 2-ой гипотезе появляется надежное fix значение, от нее отделяется 3-яя гипотеза, которая досчитывает догоняющую гипотезу до текущего времени, а затем подменяет вектор состояний 1-ой гипотезы на новый. Получается, что все предыдущие измерения фильтра считались с минимальным возрастом базы, поэтому их можно счи-

тать надежнее. В терминах многогипотезного фильтра банк получается из 2 гипотез, и в случае надежности fix решения выбирается 2-ая гипотеза, иначе 1-ая.

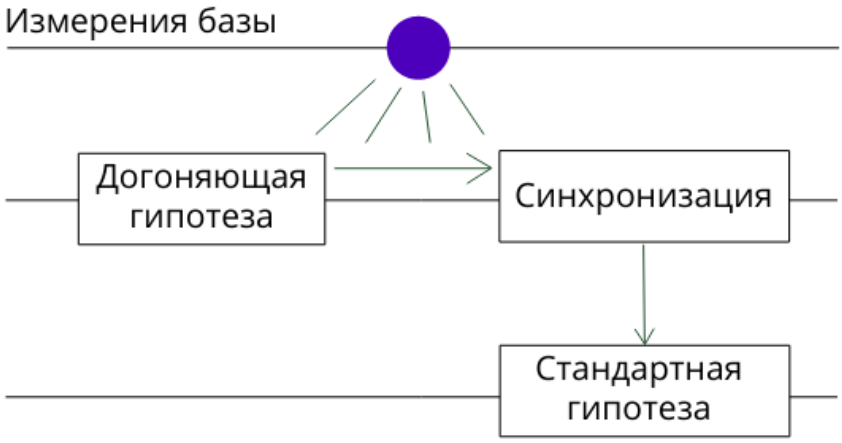

Рис. 7: Догоняющий фильтр. Точка — момент прихода базового сигнала.

На рис. 7 представлен пример работы догоняющего фильтра. В момент прихода базового сигнала догоняющая гипотеза начинает процесс синхронизации, «догоняет» стандартный фильтр, и копирует свои состояния в стандартную гипотезу. Теперь у стандартной гипотезы лучше состояния обновляемых матриц и усиления Калмана.

При маленьком возрасте базы процесс *синхронизации* 2 гипотез никогда не будет происходить, но при больших возрастах 2-ая гипотеза будет считаться с малой базой.

Для реализации использовался язык C, на котором написан rtklib. Для догоняющего фильтра была изменена структура rtk_input_data_t, которая использовалась для передачи входных данных гипотезам, теперь совершается «глубокое» копирование, и все данные о сигналах, приходящих со спутников, передаются гипотезе. Это было необходимо, потому что дальше фильтр «делит» этот сигнал на сигнал базы и

Рис. 8: Алгоритм работы догоняющего фильтра.

сигнал ровера. Затем фильтр кладет эти данные в 2 разных буфера истории типа rtk_input_queue_t. Эта структура позволяет обратиться по индексу к любому элементу, а также позволяет автоматически удалять последние элементы при добавлении новых.

Также были реализованы функции:

1. поиск лучшей базы (get_closest_base_index);
2. подсчет возраста базы для rtk_input_data (calculate_age);
3. нужно ли считать догоняющий фильтр? (will_make_step_hyp1);
4. нужно ли синхронизировать фильтры? (will_sync_hyps);
5. синхронизация фильтра (sync_hyps);
6. продолжение синхронизации дальше (sync_hyps_continue).

Для усредненного фильтра были реализованы функции calculate_average и проверка надежности решений (calculate_fixes).

Кроме того, для каждого фильтра нужно было реализовать функции интерфейса multihypothesis_strategy, который необходим для связи rtklib и мультигипотезных фильтров. Это функции rtk_multi_split, отвечающая за момент, когда гипотезы создаются, rtk_multi_qualify, которая позволяет проверить гипотезы по каким-то критериям и удалить ненужные, а также rtk_multi_merge, которая позволяет получать вектор состояний из состояний всех гипотез. В этих функциях и выполняются все функции, перечисленные выше.

Код догоняющего и усредненного фильтров доступен на GitHub[1] [2].

[1] https://github.com/Snyssfx/RTKLIB/tree/wip-test-running-filter

[2] https://github.com/Snyssfx/RTKLIB/tree/wip-lfs-test

5. Эксперимент

Нужно проверить, стали ли лучше многогипотезные фильтры по сравнению со стандартным фильтром Калмана, для этого нужно сравнить результаты тестирования по количеству точных fix решений на «хороших» и «плохих» тестах.

Для эксперимента были созданы 40 искусственных статических тестов, а также взяты из базы 20 реальных тестов с опорными интервалами. Тест проводился в режиме постобработки на рабочем компьютере с помощью rtklib-testing-suite. 40 искусственных тестов делились на 4 идеальных (стоящих в 1 точке), и каждый оригинальный зашумлялся 2, 3, 5 м по псевдодлине и 1, 2 и 3 см по фазовым невязкам. В 20 реальных тестах находились логи, записанные в пешем режиме, на машинах и квадрокоптерах. В качестве метрики был взят процент fix точек решения, в которых точность меньше 10 см. Тестовый набор поделится на «плохие» тесты, в которых стандартный фильтр неправильно находит местоположение больше, чем в 1.5% случаев, и на «хорошие» тесты, в которых стандартный фильтр точно определяет местоположение, в соотношении примерно 20 к 40. Эти числа будут отличаться, так как мы сравниваем многогипотезные фильтры с *разными* стандартными фильтрами.

Сначала мы запустили стандартный фильтр rtklib, затем новый фильтр и посчитали разность процентов точных fix решений для каждого теста. Затем нашли выборочное среднее, дисперсию и доверительный интервал для всех тестов.

Для догоняющего фильтра задержка возраста базы была выбрана 4.5 секунд, однако он был также протестирован с задержкой 0 секунд и для таких тестов работал как и стандартный фильтр. Усредненный фильтр тестировался без задержки базового сигнала. Доверительный интервал взят с доверительной вероятностью 0.99.

Результаты для «плохих» тестов (точность меньше 10 см)

Фильтры	Мат. ожидание (%)	Отклонение	Доверительный интервал (%)
Догоняющий	26.73	14.87	[19.35, 34.12]
Усредняющий	16.77	15.08	[7.6, 25.94]

Результаты для «хороших» тестов (точность меньше 10 см)

Фильтры	Мат. ожидание (%)	Отклонение	Доверительный интервал (%)
Догоняющий	0.73	1.08	[0.3, 1.16]
Усредняющий	2.83	3.96	[1.05, 4.6]

Таким образом, фильтры увеличивают количество точных fix решений на «плохих» и «хороших» тестах. На некоторых тестах прирост точности был большим (30-35%). Необходимо заметить, что распределение улучшения в процентах получается несимметричным, поэтому и такое среднеквадратичное отклонение.

6. Тестирование на целевом устройстве

Для тестирования на целевом устройстве Emlid Reach было проведено несколько изменений. Зная, что в среднем один цикл работы фильтра занимает 10-20 мс., а сигнал приходит с частотой максимум 10 Гц, количество гипотез, считающихся в одном цикле было ограничено до 5. Это особенно актуально для догоняющего фильтра, так как теперь синхронизирующаяся 3-яя гипотеза может делать только 3 расчета за раз, то есть догонять 1-ую гипотезу она будет медленнее. Но ее продуктивность от этого не меняется, так как она все равно на каждом шаге выбирает лучшую базу, поэтому с приходом новой базы она выберет ее в следующем цикле.

Работа проверена на 2 записанных тестах с временными метками, то есть они проигрывались на устройстве как будто в реальном времени, а также был протестирован просто в реальном времени из офиса. Во всех случаях фильтр показал такой же результат, как и записанные из этих логов логи для постобработки.

Заключение

В ходе работы автором было сделано:

- разработан набор тестов, добавлены реальные тестовые данные;
- автоматизированна тестирующая система, добавлена параллельность в расчетах и скрипт, который помогает создавать много искусственных тестов;
- выбрана метрика для оценки алгоритмов (процент fix решений с точностью меньше 10 см);
- разработаны 2 фильтра: усредняющий с разными углами видимости спутников и догоняющий фильтр;
- проведен эксперимент и алгоритмы протестированы на целевом приемнике.

Список литературы

[1] Chang X. W., Yang X., Zhou T. MLAMBDA: a modified LAMBDA method for integer least-squares estimation // Journal of Geodesy. — 2005. — Dec. — Vol. 79, no. 9. — P. 552–565. — Access mode: https://doi.org/10.1007/s00190-005-0004-x.

[2] Dezert J., Bar-Shalom Y. Joint probabilistic data association for autonomous navigation // IEEE Transactions on Aerospace and Electronic Systems. — 1993. — Oct. — Vol. 29, no. 4. — P. 1275–1286.

[3] Drolet L., Michaud F., Cote J. Adaptable sensor fusion using multiple Kalman filters // Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113). — Vol. 2. — 2000. — P. 1434–1439 vol.2.

[4] GoGps. GoGps project. — 2017. — Access mode: https://github.com/goGPS-Project (online; accessed: 13.05.2018).

[5] Groves P.D. Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems, Second Edition:. GNSS/GPS. — Artech House, 2013. — ISBN: 9781608070053. — Access mode: https://books.google.ru/books?id=t94fAgAAQBAJ.

[6] Lategahn J., Ax T., Röhrig C. Multi hypothesis Kalman Filter for indoor pedestrian navigation based on topological maps // 2016 IEEE/ION Position, Location and Navigation Symposium (PLANS). — 2016. — April. — P. 607–612.

[7] Multivariate outlier detection based on robust computation of Mahalanobis distances. Application to positioning assisted by RTK GNSS Networks / Elena Giménez, Mattia Crespi, M. Selmira Garrido, Antonio J. Gil // International Journal of Applied Earth Observation and Geoinformation. — 2012. — Vol. 16. — P. 94 – 100. — Access mode: http://www.sciencedirect.com/science/article/pii/S0303243411002029.

[8] Qian Qian. Multi hyphothesis Kalman filter for robo cup. — 2017. — Access mode: https://dainamite.github.io/public/publication/qian_master_thesis.pdf (online; accessed: 13.05.2018).

[9] Quinlan Michael J., Middleton Richard H. Multiple Model Kalman Filters: A Localization Technique for RoboCup Soccer // RoboCup 2009: Robot Soccer World Cup XIII / Ed. by Jacky Baltes, Michail G. Lagoudakis, Tadashi Naruse, Saeed Shiry Ghidary. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. — P. 276–287.

[10] RTKLIB библиотека. — 2018. — Access mode: https://github.com/emlid/RTKLIB/tree/emlid_2.4.3 (online; accessed: 31.05.2018).

[11] Reid D. An algorithm for tracking multiple targets // IEEE Transactions on Automatic Control. — 1979. — Dec. — Vol. 24, no. 6. — P. 843–854.

[12] Teunissen P. J. G. The least-squares ambiguity decorrelation adjustment: a method for fast GPS integer ambiguity estimation // Journal of Geodesy. — 1995. — nov. — Vol. 70. — P. 65–82.

[13] Teunissen P. J. G., Giorgi G., Buist P. J. Testing of a new single-frequency GNSS carrier phase attitude determination method: land, ship and aircraft experiments // GPS Solutions. — 2011. — Jan. — Vol. 15, no. 1. — P. 15–28. — Access mode: https://doi.org/10.1007/s10291-010-0164-x.

[14] rtklib manual. — 2017. — Access mode: http://www.rtklib.com/prog/manual_2.4.2.pdf (online; accessed: 13.05.2018).

[15] Приемник «Emlid Reach». — 2018. — Access mode: https://emlid.com/reach (online; accessed: 31.05.2018).

[16] Сайт компании «Emlid». — 2018. — Access mode: https://emlid.com (online; accessed: 31.05.2018).