Задача:

Оценивание векторов положения, скорости и ускорения с помощью фильтра Калмана и реализация на Android.

Теоретическая часть

Фильтр Калмана

Фильтр Калмана решает общую проблему оценки состояния управляемого процесса с дискретным временем, описываемого линейным стохастическим разностным уравнением.

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}, \quad (1.1)$$

Где

$$z_k = Hx_k + v_k. \quad (1.2)$$

Случайные величины представляют собой шум процесса и измерения (соответственно). Предполагается, что они независимы (друг от друга) и с нормальной плотностью распределения

$$p(w) \sim N(0, Q), \quad (1.3)$$

$$p(v) \sim N(0, R). \quad (1.4)$$

Матрица в разностном уравнении (1.1) связывает состояние на предыдущем временном шаге и состояние на текущем шаге, при отсутствии управляющей функции или технологического шума.

Алгоритм для фильтра Калмана

Фильтр Калмана оценивает процесс, используя форму управления с обратной связью: фильтр оценивает
состояние процесса в определенный момент времени, а затем получает обратную связь в виде (зашумленных) измерений. Как таковые,
уравнения фильтра Калмана делятся на две группы: уравнения обновления времени и уравнения измерения.
Уравнения обновления времени отвечают за прогнозирование вперед (во времени), оценки текущего состояния и ковариации ошибок для получения априорных оценок для следующего временного шага.
Уравнения обновления измерений отвечают за обратную связь, т.е. за внедрение нового измерения в априорную оценку для получения улучшенной апостериорной оценки. Уравнения обновления времени также можно рассматривать как уравнения-предсказатели. Их можно рассматривать как уравнения корректора.
Действительно, окончательный алгоритм оценки
напоминает алгоритм предиктора-корректора для решения численных задач, как показано ниже.
на рисунке 1.

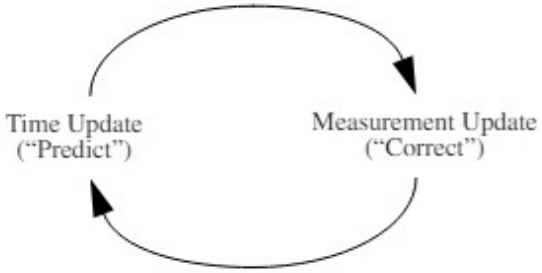

рис 1. Непрерывный цикл дискретного фильтра Калмана. Обновление времени прогнозирует оценку текущего состояния вперед во времени. Обновление измерений корректирует прогнозируемую оценку с учетом фактических измерений на данный момент.

Конкретные уравнения для обновления времени и измерений представлены ниже в таблице 1 и таблице 2.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \qquad (1.9)$$

$$P_k^- = AP_{k-1}A^T + Q \qquad (1.10)$$

Таблица1. Уравнения обновления по времени дискретного фильтра Калмана

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \qquad (1.11)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \qquad (1.12)$$

$$P_k = (I - K_k H)P_k^- \qquad (1.13)$$

Таблица 2. Уравнения обновления измерений дискретного фильтра Калмана.

Первой задачей во время обновления измерений является вычисление коэффициента усиления Калмана. Обратите внимание, что
уравнение, данное здесь как (1.11), совпадает с (1.8). Следующим шагом будет реальное измерение процесса.
Чтобы получить, а затем сгенерировать апостериорную оценку состояния путем включения измерения
как в (1.12). И снова (1.12) — это просто (1.7). Последний шаг – получить апостериорную оценку ковариации ошибок с помощью (1.13).
После каждого обновления пары времени и измерения процесс повторяется с предыдущим апостериорным оценки, используемые для прогнозирования новых априорных оценок. Эта рекурсивная природа является одной из
очень привлекательных особенностей фильтра Калмана — они значительно расширяют возможности практической реализации.

осуществимо, чем (например) реализация фильтра Винера, который предназначен для работы со всеми данными непосредственно для каждой оценки. Фильтр Калмана вместо этого рекурсивно
обуславливает текущую оценку всех прошлых измерений.

Расширенный фильтр Калмана (EKF)
Как описано выше в разделе 1, фильтр Калмана решает общую проблему: оценить состояние управляемого процесса с дискретным временем, управляемого линейным стохастическим разностным уравнением. Но что произойдет, если оцениваемый процесс и (или) связь измерения с процессом нелинейна? Одни из самых интересных и успешных применений фильтрации Калмана были такими ситуациями. Фильтр Калмана, линеаризующий текущее среднее значение и ковариации называется расширенным фильтром Калмана или EKF. Используя ряд Тейлора, мы можем линеаризовать оценку вокруг текущей оценки. Использование частных производных функций процесса и измерения для вычисления оценок даже для нелинейных отношений

Предположим, что наш процесс снова имеет вектор состояния, но процесс теперь определяется нелинейным стохастическим разностным уравнением.

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}), \tag{2.1}$$

Где

$$z_k = h(x_k, v_k), \tag{2.2}$$

где случайные величины и снова представляют шум процесса и измерения, как в (1.3) и (1.4). В этом случае нелинейная функция в разностном уравнении (2.1) связывает состояние на предыдущем временном шаге в состояние на текущем временном шаге. Он включает в себя в качестве параметров
любую управляющую функцию и шум процесса с нулевым средним.

Чтобы оценить процесс с нелинейной разностью и соотношениями измерений, мы начинаем с новых основных уравнений, которые линеаризуют оценку (2.3) и (2.4),

$$x_k \approx \tilde{x}_k + A(x_{k-1} - \hat{x}_{k-1}) + Ww_{k-1}, \tag{2.5}$$

$$z_k \approx \tilde{z}_k + H(x_k - \tilde{x}_k) + Vv_k. \tag{2.6}$$

Где

- x_k и z_k – действительные векторы состояния и измерения.
- $\tilde{x}_k$ и $\tilde{z}_k$ – приближенные векторы состояния и измерения из (2.3) и (2.4).
- $\hat{x}_k$ – апостериорная оценка состояния на шаге k
- случайные переменные w_k и v_k представляют шум процесса и измерения, как в (1.3) и (1.4).

- A - матрица Якоби частных производных f по x, то есть

$$A_{[i,\,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0),$$

- H — матрица Якоби частных производных h по x

$$H_{[i,\,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\tilde{x}_k, 0),$$

- V — матрица Якоби частных производных по v,

$$V_{[i,\,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}(\tilde{x}_k, 0).$$

Теперь мы определим новое обозначение для ошибки прогнозирования:

$$\tilde{e}_{x_k} \equiv x_k - \tilde{x}_k, \tag{2.7}$$

и остаток измерения.

$$\tilde{e}_{z_k} \equiv z_k - \tilde{z}_k. \tag{2.8}$$

Используя (2.7) и (2.8), можем написать управляющие уравнения для процесса ошибок как

$$\tilde{e}_{x_k} \approx A(x_{k-1} - \hat{x}_{k-1}) + \varepsilon_k, \tag{2.9}$$

$$\tilde{e}_{z_k} \approx H\tilde{e}_{x_k} + \eta_k, \tag{2.10}$$

гдеε_k и η_k представляют новые независимые случайные величины, имеющие нулевое среднее значение и ковариационные матрицы, WQW^T и VRV^T с Q и R как в (1.3) и (1.4) соответственно.

Случайные величины (2.9) и (2.10) имеют примерно следующую плотность распределения:

$$p(\tilde{e}_{x_k}) \sim N(0, E[\tilde{e}_{x_k}\tilde{e}_{x_k}^T])$$

$$p(\varepsilon_k) \sim N(0, WQ_kW^T)$$

$$p(\eta_k) \sim N(0, VR_kV^T)$$

Полный набор уравнений EKF показан ниже в Таблице 3 и Таблице 4.

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0) \qquad (2.14)$$

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \qquad (2.15)$$

Таблица 3. Уравнения обновления по времени EKF.

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \qquad (2.16)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0)) \qquad (2.17)$$

$$P_k = (I - K_k H_k) P_k^- \qquad (2.18)$$

Таблица 4. Уравнения обновления измерений EKF.

Основная работа EKF такая же, как и у линейного дискретного фильтра Калмана, как показано на рисунке 2

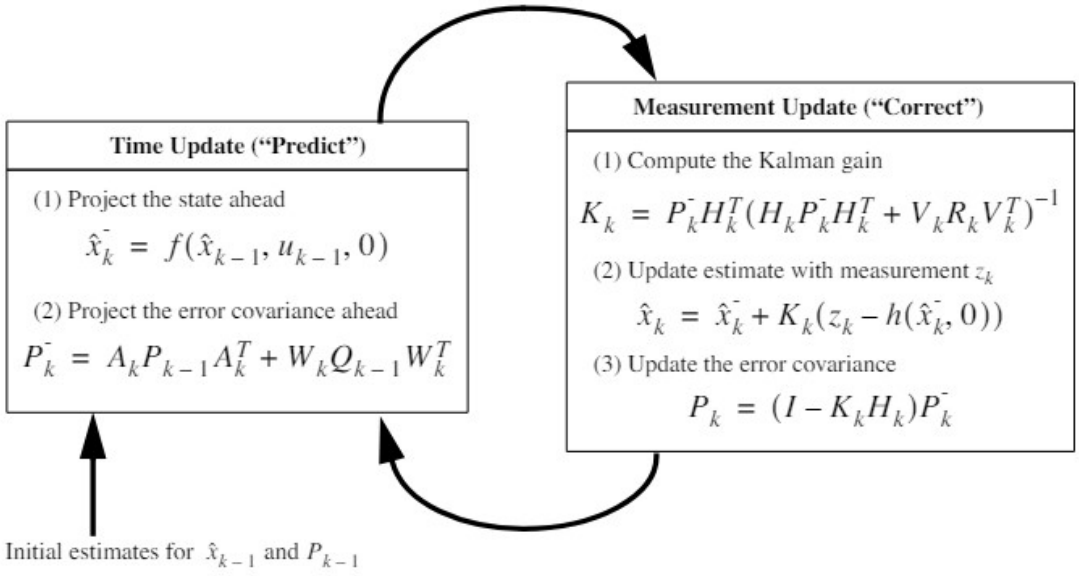

Рисунок 2. Полная схема работы расширенного фильтра Калмана

Практическая часть

Ardupilot

Реализация EKF была взята с Ardupilot - пакета программного обеспечения для автопилота беспилотных транспортных средств с открытым исходным кодом.

В пакете реализованы 4 фильтра, каждый из которых вычисляет 21,22,23,24 состояния системы соответственно.

4 параметра кватерниона

3 Компоненты скорости «Север», «Восток», «Вниз»

3 Компоненты положения «Север», «Восток», «Вниз»

3 Компоненты смещения дельта-угла IMU

2 Смещения дельта-скорости IMU по X и Y (включено только в вывод с 24 состояниями)

1 Смещение дельта-скорости IMU по Z (включены только в вывод состояний 22 и 23)

2 Компоненты скорости северного, восточного ветра

3 Компоненты магнитного потока севера, востока и земли

3 Компоненты фиксированного магнитного потока тела X,Y,Z (это противоположный знак смещениям компаса, используемым APM)

1 Смещение ландшафта вдоль оси вниз (включено только в вывод фильтра с 23 состояниями)

Реализация фильтров доступна как на Matlab так и на C++.

В ходе работы мы использовали EKF с двадцатью двумя состояниями, реализованном на C++.

Фильтр принимает данные с показаниями акселерометра, магнетометра, гироскопа, барометра и данные с GPS.

https://github.com/priseborough/InertialNav

Sensor Logger

Для получения данных было использовано приложение Sensor Logger

Sensor Logger — это бесплатный, простой в использовании кроссплатформенный регистратор данных, который регистрирует показания распространенных датчиков движения на смартфонах. После завершения записи можно экспортировать в

виде сжатого файла CSV, JSON, SQLite, KML или просмотреть в приложении с помощью интерактивных графиков. Поддерживаемые датчики и измерения включают в себя:

- Ускорение устройства (акселерометр; необработанный и откалиброванный)
- Вектор гравитации (акселерометр)
- Скорость вращения устройства (гироскоп)
- Ориентация устройства (гироскоп; необработанный и откалиброванный)
- Магнитный курс (магнитометр; необработанный и откалиброванный)
- Барометрическая высота (барометр)
- Координата GPS, высота, скорость и курс
- Аудио (микрофон)
- Громкость (микрофон)
- Изображения с камеры (только спереди и сзади, только передний план)
- Видео с камеры (только спереди и сзади, только передний план)
- Шагомер (по оценкам операционной системы)
- Частота пульса (требуется установленное приложение для часов)
- Wrist Motion (требуется установленное приложение для часов)
- Датчик освещенности (только Android)
- Аннотации (метка времени и необязательный сопровождающий текстовый комментарий)
- Уровень и состояние батареи устройства
- Уровень яркости экрана устройства

https://github.com/tszheichoi/awesome-sensor-logger/

Часть практического задания заключалась в считывании данных с датчиков телефона и последующий передачи их в фильтр. Необходимо было преобразовать данные логгера в вид, требуемый фильтром для обработки.

Данные	InternalNav			Логи
GPS				
0	'LineNo'	есть везде		seconds_elapsed
1	'Status'	возможно фиктивный	Location	3
2	'TimeMS'			seconds_elapsed
3	'Week'		Location	0
4	'NSats'		Location	0
5	'HDop'		Location	0
6	'Lat'		Location	latitude
7	'Lng'		Location	longitude
8	'RelAlt'		Location	altitude
9	'Alt'		Location	0
10	'Spd'		Location	speed
11	'GCrs'			bearing
12	VZ'			0
13	'T'			

IMU				
0	'LineNo'	есть везде		seconds_elapsed
1	'TimeMS'			seconds_elapsed
2	'GyrX'		GyroscopeUncalibrated	x
3	'GyrY'		GyroscopeUncalibrated	y
4	'GyrZ'		GyroscopeUncalibrated	z
5	'AccX'		AccelerometerUncalibrated	x
6	'AccY'		AccelerometerUncalibrated	y
7	'AccZ'		AccelerometerUncalibrated	z

MAG				
0	'LineNo'	есть везде		seconds_elapsed
1	'TimeMS'			seconds_elapsed
2	'MagX'		MagnetometerUncalibrated	x
3	'MagY'		MagnetometerUncalibrated	y
4	'MagZ'		MagnetometerUncalibrated	z
5	'OfsX'			0
6	'OfsY'			0
7	'OfsZ'			0

NTUN				
0	'LineNo'	есть везде		seconds_elapsed
1	'TimeMS'	time		seconds_elapsed
2	'Yaw'			0
3	'WpDist'			0
4	'TargBrg'			0
5	'NavBrg'			0
6	'AltErr'			0
7	'Arspd'		Location	speed
8	'Alt'		Barometer	relativeAltitude
9	'GSpdCM'			0

ATT				
0	'LineNo'	есть везде		seconds_elapsed
1	'TimeMS'			seconds_elapsed
2	'Roll'		Orientation	roll
3	'Pitch'		Orientation	pitch
4	'Yaw'		Orientation	yaw

Таблица 5. Соответствие данных логгера и фильтра

Данные с логгера программно (Python) преобразовывались в данные для фильтра

В итоге для предоставленных тестовых данных фильтр выдал следующее решение

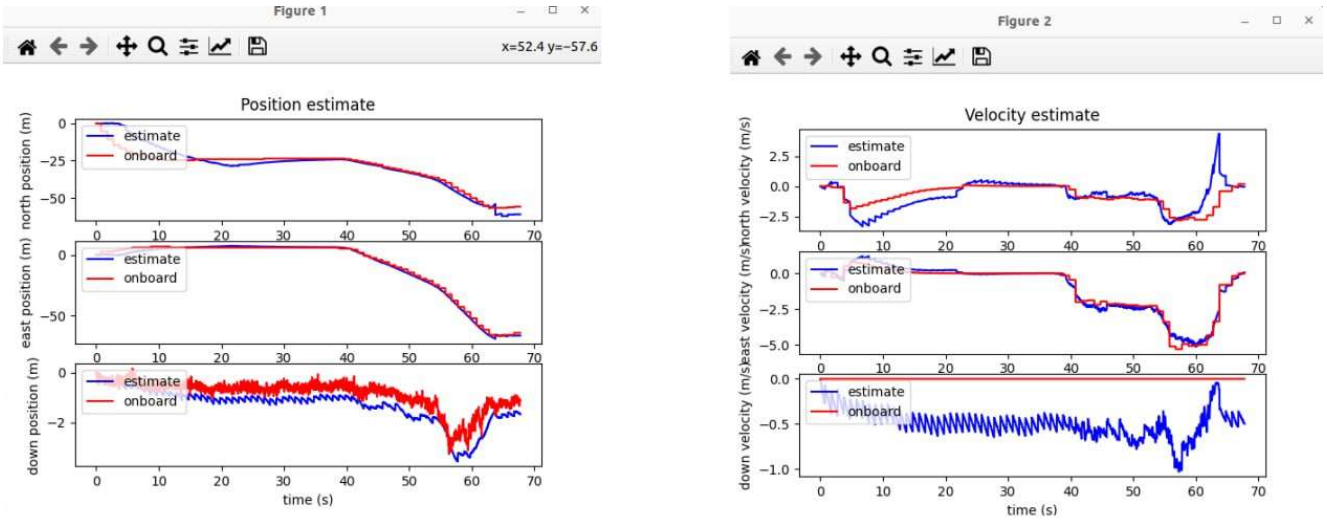

Рисунок 3. Решение, полученное фильтром

Реализация на Andriod

Задача заключалась в запуске кода фильтра на Android.

Так как фильтр реализован на C++ он не может быть скомпилирован вместе с кодом некоторого приложения для запуска на Android (т.к. для этого он должен был быть реализован на Java/Kotlin).

Для этого был использован Android NDK.

Android NDK — это набор инструментов, который позволяет реализовывать части приложения в собственном коде, используя такие языки, как C и C++.

В данном случае код фильтра на C++ вызывался из кода основного действия приложения, написанное на Java. Приложение считывало данные для фильтра из произвольного места в телефоне, запускало на них фильтр и писала в локальную папку приложения выходные данные.

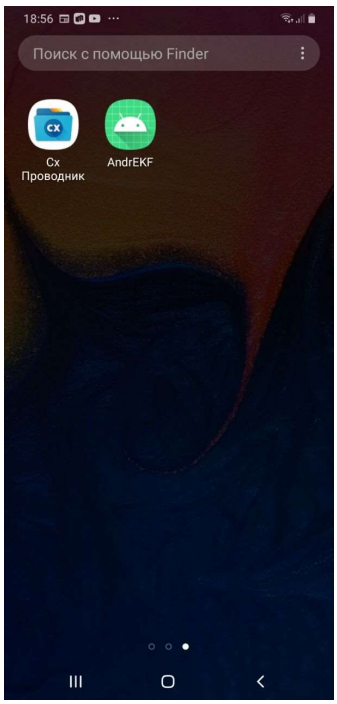

рис 4. Иконка приложения (AndrEKF)

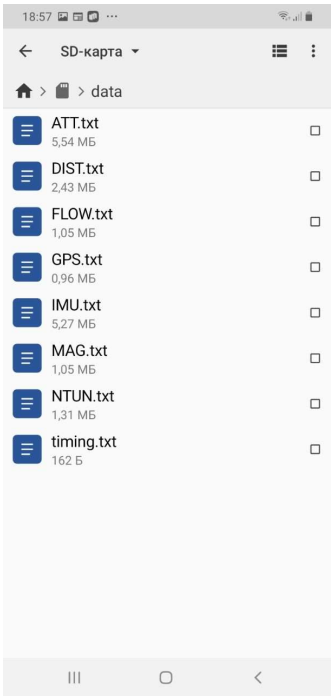

рис 5. Данные для фильтра

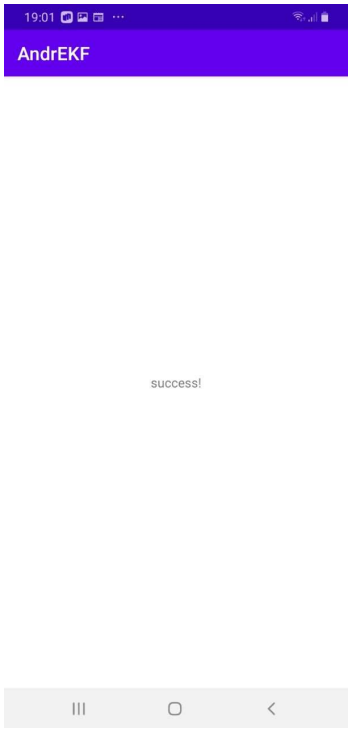

рис 4. Приложение, после окончания работы

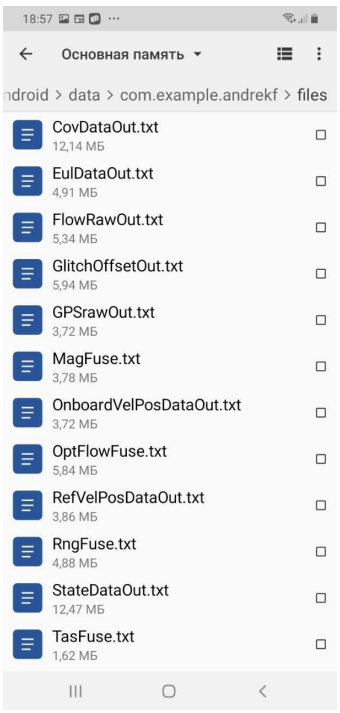

рис 5. Результат работы приложения