Мой первый околонаучный проект: инерциальный трекер

Даниил Барков

Осень 2022

Оглавление

Введение

Не буду рассказывать, когда и почему я захотел написать программу для работы с инерциальным сенсором, а начну с того момента, когда я начал изучать линейную алгебру и MATLAB.

Мне открылись новые возможности и методики, и создать прототип программы удалось очень быстро. К тому же у меня есть тяга к (относительно) научному познанию, и через такую работу я могу потренироваться. Текст же поможет структурировать мысли и сохранить математические выкладки.

Описание

Я в процессе разработки электронного девайса с набором датчиков, который умеет измерять ускорения, угловые скорости, магнитные поля, и давление (10 степеней свободы). Он обрабатывает эти данные и выдаёт набор чисел, которые отправляются на компьютер по USB. Там их считывает программа, написанная на матлабе и выдаёт такую картинку. При запуске программа определяет свой наклон при помощи акселеро-

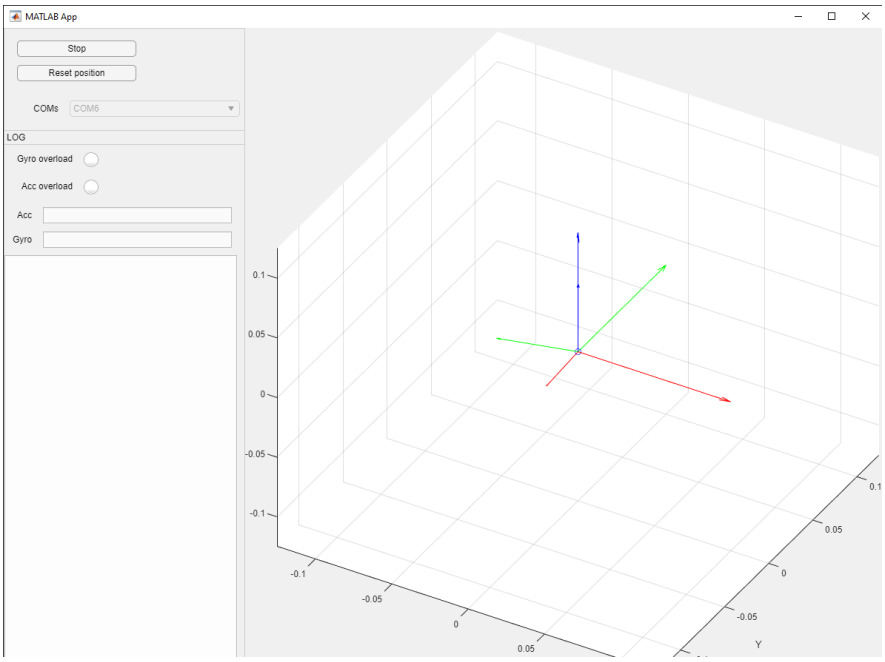

Рис. 1: Моё первое десктоп приложение

метра, азимут - по магнитометру, затем хитрым образом суммирует повороты по осям, компенсирует дрейф гироскопов и получает набор векторов x, y, z.

Цели и задачи

Пока что план такой - написать библиотеку, функции которой будут принимать массив данных с условного датчика, измерять время между приёмами и возвращать необходимые данные:

- трёхмерный базис, векторы которого сонаправлены с датчиком ($B_{(3\times3)}$)

- скорость относительно земли ($\overrightarrow{V}$)
- ускорение относительно земли ($\overrightarrow{A}$) и относительно датчика ($\overrightarrow{A'}$)
- азимут (θ)
- высота над уровнем моря или хотя бы над местом старта (alt)
- время между измерениями акселерометра и гироскопа (τ), магнитометра (τ_m) и барометра (τ_{alt})
- угловые скорости ($\varphi_x,\ \varphi_y,\ \varphi_z$, то есть $\overrightarrow{\varphi}$)
- крен ($\theta_r - roll$) и тангаж ($\theta_p - pitch$) для удобоваримости

Затем подготовлю шаблон для настройки микроконтроллера (MK) и сделаю библиотеку для работы с конкретной моделью датчика.

Электронная начинка

Для проверки алгоритмов подготовил элетронную схему с датчиками и микроконтроллером. Главное устройство тут - это модуль 10 DOF IMU sensor (B) от Waveshare, включающий в себя

- 3-х осевой акселерометр (до ± 156 м/с^2)
- 3-х осевой гироскоп (до ± 2000 °/с)
- 3-х осевой магнитометр (до ± 4800 мкТл)
- барометр ($300 \sim 1100$ гПа, т.е $+9000 \sim -500$ м)
- термометр (даже)

Считыванием и обработкой данных занимается МК STM32F103C8T6, в народе Blue Pill. Работает на 72 МГц и не имеет ускорителя вычислений с плавающей точкой (FPU), поэтому считает относительно медленно. Он собирает по I^{2}C данные с акселерометра и гироскопа каждые $\tau = 4500$ мкс, а магнитометра - $\tau_m = 15000$ мкс. С барометром я пока не работал. Результаты вычислений отправляются на ПК через USB-UART конвертер, где их ловит матлаб и выводит их на экран в виде векторов и индикаторов перегрузки.

Алгоритм работы и программная реализация

Термины и определения

Перед описанием алгоритма расскажу как и что я назвал. Как рассказал выше, датчик выдаёт такой набор величин:

- ускорение ($\overrightarrow{A'}$)

- давление (P)
- угловые скорости $\overrightarrow{\varphi'}$
- интенсивность магнитного поля $\overrightarrow{M'}$

Также измеряются $\tau,\ \tau_m\ \ \tau_{alt}$.

На акселерометр всегда (если он не падает) действует сила тяжести, которую он также регистрирует своими внутренними грузиками - $\overrightarrow{G'}$. По-хорошему он должен иметь значение $\overrightarrow{G'} = (0, 0, -9.8)$, но это когда датчик правильно понимает совою ориентацию. Поэтому оригинальную силу тяжести назову отдельно - $\overrightarrow{G}$.

Частый термин здесь - базис $B_{(3\times3)}$. Под ним я подразумеваю три ортонормированных вектора датчика, как-то ориентированных в пространстве. Например, $\overrightarrow{z}$ направлен вверх, $\overrightarrow{x}$ - на север. Получится единичная матрица матрица. Если $\overrightarrow{x}$ смотрит на запад, получится так:

$$B_{(3\times3)} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Также часто будут использоваться матрицы поворота:
вокруг $\overrightarrow{z}$:

$$\mathrm{rotz}(\phi) = \begin{pmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

вокруг $\overrightarrow{x}$:

$$\mathrm{rotx}(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{pmatrix}$$

вокруг $\overrightarrow{y}$:

$$\mathrm{roty}(\phi) = \begin{pmatrix} \cos(\phi) & 0 & -\sin(\phi) \\ 0 & 1 & 0 \\ \sin(\phi) & 0 & \cos(\phi) \end{pmatrix}$$

вокруг произвольного $\overrightarrow{l}$:

$$\mathrm{rotl}(\phi, \overrightarrow{l}) = \begin{pmatrix} c + (1-c)l_x^2 & (1-c)l_x l_y - sl_z & (1-c)l_z l_x + sl_y \\ (1-c)l_x l_y + sl_z & c + (1-c)l_y^2 & (1-c)l_z l_y - sl_x \\ (1-c)l_x l_z - sl_y & (1-c)l_y l_z + sl_x & c + (1-c)l_z^2 \end{pmatrix}$$

где

$$c = \cos(\phi),\ s = \sin(\phi),\ \overrightarrow{l} = (l_x, l_y, l_z),\ |\overrightarrow{l}| = 1$$

Математика

Тут начинается самое интересное. Преимущественно разработчиков, например летательный аппаратов, интересуют углы крена θ_r и тангажа θ_p. Их можно вычислять простым способом. На акселерометр действует $\overrightarrow{G}$, который как-то проецируется на a_x, a_y, a_z,

получается $\overrightarrow{A'}$.

$$\theta_p = \arcsin\left(\frac{a_x}{|\overrightarrow{A'}|}\right),\ \theta_r = -\arcsin\left(\frac{a_y}{|\overrightarrow{A'}|}\right).$$

Но тут возникает проблема. При поступательном ускорении оно складывается с силой тяжести:

$$\overrightarrow{A'} = \overrightarrow{A} + \overrightarrow{G},$$

и θ_r, θ_p меняются. В таком виде формулу использовать нельзя. Отсюда появляется потребность использовать гироскоп для вычисления поворотов. Однако надо задать начальное положение, которое можно получить написанным верхнему способом, оставив устройство смирно полежать несколько секунд.

Тут стоит отметить, что гироскоп микроэлектромеханический и гироскоп механический - работают по-разному. Второй - это маховик с осью на нескольких шарнирах, который старается сохранить свою ориентацию (или как-то так, надо разобраться). МЭМС гироскоп же возвращает какую-то среднюю скорость вращения вокруг оси за период измерения. Поэтому не получится узнать текущие наклоны, в любой момент обратившись к нему. Надо регулярно вычислять повороты и как-то суммировать их.

$$\overrightarrow{\phi} = \overrightarrow{\varphi}\tau = (\phi_x,\ \phi_y,\ \phi_z)$$

Но мне не нравится представление ориентации в виде трёх (+рысканье) углов, потому что оно понятно только оператору РУ самолётика. Да и сложно представить, что происходит с азимутом, когда нос смотрит вверх, какой угол равен 180° при полёте брюхом кверху - тангаж или крен. И почему. А компьютеру это не объяснить и подавно. Надо бы найти универсальное решение: опуститься к естественному представлению тел в нашем трёхмерном пространстве и обратиться к линейной алгебре.

Так вот. Надо как-то складывать повороты, которые (по крайней мере, в моём представлении) некоммутативные. Однако для малых углов этим можно пренебречь. Поворачиваю я так:

$$B_{(3\times3)k+1} = B_{(3\times3)k} \cdot \left(\mathrm{rotx}(\tau_k \cdot \varphi_{xk}) \cdot \mathrm{roty}(\tau_k \cdot \varphi_{yk}) \cdot \mathrm{rotz}(\tau_k \cdot \varphi_{zk})\right)$$

Оказалось, что если перемножать в таком порядке $rotx(\phi_x) \cdot B_{(3\times3)}$, то базис повернётся вокруг абсолютного вектора $\overrightarrow{Z}$. А если наоборот, то вокруг своего $\overrightarrow{z}$.

Все эти повороты начинаются с изначального положения, которое обязательно не вертикально и обязательно не на север. Надо это положение восстановить, и я воспользуюсь испробованным методом.

При включении МК делает пару сотен опросов акселерометра, усредняет показания, получает $\overrightarrow{A'}$, который должен равняться $\overrightarrow{G}$. Но выходит

$$\overrightarrow{G} \neq B_{(3\times3)} \cdot \overrightarrow{A'}.$$

Третье - это данность, первое даже не обсуждается. Неверным остаётся второй элемент. Надо его исправить.

Датчик какой-то стороной наклонён к земле, значит надо наклонить его обратно, то есть повернуть вокруг какой-то оси на угол отклонения. К тому же ось должна

проходить через 0 и быть нормальной к обоим векторам.
Ось вот такая:

$$\overrightarrow{l} = \frac{\overrightarrow{A'} \times \overrightarrow{G}}{|\overrightarrow{A'} \times \overrightarrow{G}|}$$

Такой угол:

$$\phi = \arccos\left(\frac{\overrightarrow{A'} \cdot \overrightarrow{G}}{|\overrightarrow{A'}| \cdot |\overrightarrow{G}|}\right)$$

Такой поворот:

$$B_{(3\times3)} = \mathrm{rotl}(\phi, \overrightarrow{l}) \cdot B_{(3\times3)}$$

Дабы уменьшить погрешности в будущем, МК перезапоминает силу тяжести:

$$\overrightarrow{G} = (0,\ 0,\ -|\overrightarrow{A'}|)$$

Аналогичным образом настраивается ориентация по азимуту, только в роли $\overrightarrow{G}$ выступает $\overrightarrow{N} = (1, 0, 0)$, а $\overrightarrow{A'}$ - это $\overrightarrow{M} = B_{(3\times3)} \cdot \overrightarrow{M'}$. С обнулённой вертикальной составляющей. А вращается вокруг $\overrightarrow{Z}$.

Чтобы получить оригинальное ускорение, надо вычесть силу тяжести:

$$\overrightarrow{A} = \overrightarrow{A'} + \overrightarrow{G}$$

Теперь можно вращать датчик и любоваться его отображением в матлабе. Но вот ещё одна проблема. Гироскопы сложно откалибровать, и они очень шумные. К тому же от резких поворотов, у них случаются перегрузы, и все скорости выше порога обрезаются. От этого накапливаются ошибки, и вектора постепенно уплывают. Поэтому необходимо периодически корректировать ориентацию. Делается это описанным выше способом, но чтобы было плавнее, базис доворачивается не на весь угол за раз, а на какую-то подобранную долю угла (в моём случае - 0.02), а функция выполняется при $9.6 < |\overrightarrow{A'}| < 10$.
Про азимут я немного вру, т.к. магнитометр приходится калибровать, а я давно не делал это. Поэтому временно не использую компас, и если с ним возникнет путаница, не обращайте, пожалуйста, внимание.

На этом собственно математика в программе заканчивается. Сюда не вошли вычисление скорости и высоты, т.к. я ещё не разобрался.

Алгоритму есть куда улучшаться, но это прототип, и остались нерешённые проблемы, связанные с точностью вычислений и подготовкой устройства к работе.

Прототип

Изначально все чудеса происходили в матлабе. Мобильное приложение собирало показания с датчиков на телефоне в массив, сохраняло в облачный файл, который затем открывался в матлабовском скрипте. В нём я прототипировал алгоритм, что было очень удобно. Произошёл прув оф консепт.

Следующим шагом стало наблюдение в реальном времени. МК общался с датчиком и передавал на ПК сырые данные, которые обрабатывались в матлабовском скрипте.

Очень удобно было проводить там калибровку (расскажу позже). Но в радиоуправляемую подводную лодочку ноут не положишь, а к микрокомпьютерам доверия нет, тк даже гоношный ноут напрягается от такой нагрузки. Поэтому стал переносить вычисления на маленький МК .

Программа для ПК

Приложение принимает 10 байт, 9 из которых - это координаты трёх векторов, помноженные на 100 (векторы в длину меньше 1, а передавать single precision вчетверо дольше), а последний хранит 2 бита для индикаторов перегруза датчиков.

Пока что не делал проверку целостности и двустороннее общение, т.к. это демонстрационная программа. Но когда буду писать утилиту для настройки, обязательно займусь этим.

Программа для МК

Матричные операции

Алгоритм можно было перенести построчно. Но готовых функций работы с матрицами у меня не было. Первая мысль - воспользоваться C компайлером и получить готовый C код. Но это тоже оказалось муторно. Поэтому я решил написать свои матлаб-лайк матричные функции. Мне были необходимы:

- переменная, хранящая произвольный набор float чисел и информацию об их порядке
- умножение на число
- сумма двух матриц
- умножение двух матриц
- транспонирование
- определитель
- векторное произведение
- скалярное произведение
- нормирование вектора
- модуль вектора
- чтение элемента m_{ij}
- запись элемента
- создание матрицы $m \times n$
- создание единичной матрицы $n \times n$

- вычисление угла между векторами
- матрицы поворота

Матлабовского должно быть вот что:

- матрица выглядит как одна переменная
- функции проверяют возможность выполнения операции
- результат возвращается в виде числа, либо записывается в переменную, указанную в аргументах
- переменные для результатов не приходится готовить вручную, а функция сама распоряжается памятью для сохранения результата, записывает её параметры матрицы

Матрица у меня - это структура:

```
struct matrix {
    float *arr;
    uint8_t rows;
    uint8_t cols;
    uint8_t size; // rows*cols
};
```

Типичная функция выглядит так:

```
int matrixSum(matrix* input1, matrix* input2, matrix* output) {
    if (input1->cols != input2->cols || input1->rows != input2->rows)
        return 1;
    if (output->cols != input2->cols || output->rows != input2->rows) {
        float* tmp = (float*) malloc(sizeof(float) * input1->size);
        for (int i = 0; i < input1->size; i++) {
            tmp[i] = input1->arr[i] + input2->arr[i];
        }
        matrixRenewArray(output, input1->rows, input1->cols, tmp);
    }
    else {
        for (int i = 0; i < input1->cols * input1->rows; i++) {
            output->arr[i] = input1->arr[i] + input2->arr[i];
        }
    }
    return 0;
}
```

Таким образом, можно на ходу перезаписывать матрицы, не заморачиваясь их размерами. Я прототипировал в Visual studio на C++. Но у МК память ограничена, а избегать её переполнения я ещё не научился. Поэтому сделал всем матрицам массив по 9 элементов. На удобстве это почти не сказалось, зато могу при компиляции прикинуть количество занятой RAM. Когда заведу гит, выложу все библиотеки туда.

Что касается производительности. Общение с датчиками почти не занимает процессорное время благодаря DMA и прерываниям. В основном цикле нет делэев а есть

отсчёт времени между операциями и статусы выполнения. Самым долгим делом пока являются описанные вычисления - до 900 мкс, что при периоде опроса в 4500 мкс оставляет много времени на другие задачи. К тому же речь идёт про один из самых простых МК этой серии. При использовании FPU, вычисления выполняются вдвое быстрее.

Проблемы

Расскажу о проблемах, связанных с работой датчика, МК и алгоритма

Корявые вычисления с плавающей точкой

Здесь я расскажу о том, то я много вычилсяю тригонометрические функции, которые по природе неидеальны.
О том, что действительные числа в формате float подразумевают накопление ошибки.
И о том, что я хочу периодически проверять его ортонормированность. Ну и как-то чинить базис в дальнейшем.

Пока что считаю главной проблемой ошибки вычислений, вызванных тем, что значения тригонометрических функций иррациональные, а они в свою очередь коряво интерпретируются двоичным форматом. Это приводит к тому, что базисы перестают быть ортонормированными. Кажется, это не очень существенная проблема, но (как мне кажется) в корне не решаемая. Однако я могу периодически фиксировать эту ошибку и исправлять базис, теряя при этом частичку информации.

Частота измерений датчика и интерполяция

Подумаю, какую интерполяцию использовать, и почему. (до прямых линий между точками) Надо решить, как и зачем интерполировать получаемые данные. Я пока что не проводил подробных анализов, но предполагаю, что период измерений достаточно большой, чтобы значения между точками сильно различались. Поэтому использовать формулу $\phi_k = \tau_k \cdot \varphi_k$ – неудачное решение. Для повышения точности можно хотя бы усреднять соседние измерения, то ест вот так: $\phi_k = \tau_k \cdot 0.5(\varphi_k + \varphi_{k-1})$

Шумы и фильтрация

Построю гистограмму шумов покоящихся датчиков на разных диапазонах измерений. Подумаю, нужно ли использовать фильтр. Чего это будет стоить (скорость и память). Подберу варианты.

Калибровка

Объясню необходимость калибровки, опишу текущий метод, расскажу, почему хочу поменять.

Планы на будущее

Расскажу о том, что хочу улучшить, и куда добавить придуманный алгоритм

В самом конце обращу внимание на название - я хочу сделать инерциальный трекер. То есть, устройство, которое может отслеживать своё перемещение. Это чуть интереснее, чем просто электронный гироскоп. Я конечно же не ожидаю от него точности, как от GPS, достаточно трёхмерной траектории за последние несколько секунд. Набор таких можно, например, прикрепить к одежде и использовать для передачи движения аватару в метавселенной (конечно же я не хочу таким заниматься). А можно использовать для стабилизации балансирующего робота.