САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Моисеенко Евгений Александрович

Реализация системы определения направления на источник звука на контроллере ТРИК

Курсовая работа

Научный руководитель:
ст. преп. Я.А. Кириленко

Санкт-Петербург
2015

Оглавление

Введение

Современные робототехнические контроллеры имеют вычислительные ресурсы, позволяющие реализовывать алгоритмы машинного обучения и классификации, работающие в режиме реального времени. С помощью этих методов возможно создание интерактивных интеллектуальных систем, собирающих данные с помощью различных датчиков и оперативно реагирующих на изменение окружающий среды.

В контексте создания подобной системы задача определения положения источника звука может быть интересна как сама по себе, так и как часть более сложного алгоритма. Например, робот может использовать данные о положении диктора для наведения камеры на говорящего и дальнейшего распознавания лица человека.

Робототехнический контроллер ТРИК, используемый кафедрами системного программирования и теоретической кибернетики СПБГУ, предоставляет платформу для реализации роботов, предлагающую разработчику доступ к широкому набору периферийных устройств. Контроллер имеет два входа для микрофонов, что позволяет реализовать алгоритм определения направления на источник звука в плоскости. Цель данной работы - разработка алгоритма, поддерживающего эту функциональность.

1. Постановка задачи

- Реализация алгоритма направления на источник звука по сигналам с двух микрофонов
- Адаптация приложения для платформы ТРИК
- Исследование результатов работы датчика

2. Описание алгоритма

2.1. Детектор направления на источника звука

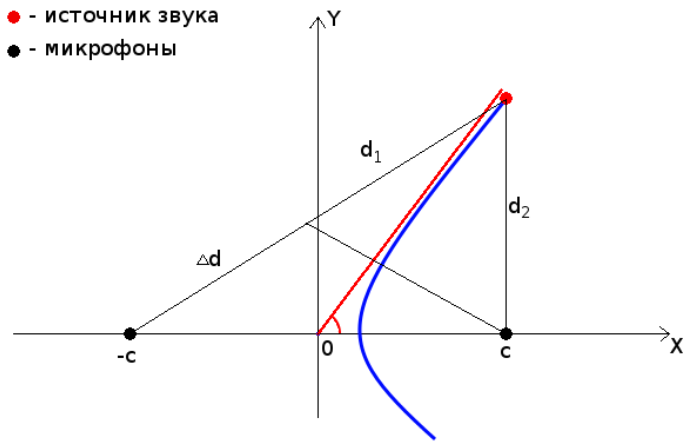

Рис. 1: Схематичное изображение положения микрофонов и источника звука

Рассмотрим систему координат, связанную с парой микрофонов (1). Микрофоны расположены на оси OX, на одинаковом расстоянии от центра оси равном c. Система принимает звуковой сигнал, который расположен на расстоянии $d1$ от первого микрофона и $d2$ от второго. По показаниям микрофонов можно определить разность расстояний до источника звука: $|d1 - d2| = \Delta d$. Геометрическое место точек, разность расстояний от которых до двух фиксированных точек равна константе - гипербола. Уравнение гиперболы в декартовых координат выглядит следующим образом:

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

Параметр a задает половину разности расстояний до фокусов ($2a = \Delta d$), расположенных в точках $(c, 0)$ и $(-c, 0)$. Знак a определяет к какому фокусу ближе ветвь гиперболы, на которой расположен источник сигнала. Параметр b определяется из соотношения $c^2 = a^2 + b^2$

Пусть $a > 0$. Рассмотрим ассимптоту гиперболы, определяемую уравнением $\frac{x}{a} = \frac{y}{b}$. Расстояние от фокуса $(c, 0)$ до этой асимптоты равно b, из чего можно сделать вывод что отрезок оси OX длины c, отрезок асимптоты длины a и перпендикуляр из фокуса к асимптоте образуют прямоугольный треугольник. Угловой коэффициент асимптоты определяется соотношением:

$$\phi = arccos(\frac{a}{c}) \ , \phi \in [0, \frac{\pi}{2}]$$

Разность расстояний до источника звука в отсчётах вычисляется при помощи корреляционной функции (2). Корреляционная функция двух дискретных сигналов по

определению равна:

$$corr(f, g)[n] = \sum_{m=-\infty}^{+\infty} f[m]g[n+m]$$

Смещение отсчёта, на котором достигается максимум корреляционной функции, от 0 принимается за разность расстояний в отсчетах. Остается лишь перевести это значение в метры по формуле:

$$a = \frac{pv}{2F}$$

Где a - разность расстояний в метрах, p - разность расстояний в отсчётах, $v = 330\frac{m}{s}$ - скорость звука, F - частота дискретизации.

Так как разность расстояний Δd не может превышать $2c$, можно предположить, что и максимум корреляционной функции будет находиться в ограниченном диапазоне $[-L, L]$, где константа L зависит от расстояния между микрофонами c. Из этого можно сделать вывод, что корреляционную функцию следует вычислять лишь для отсчётов из отрезка $[-L, L]$.

Описанный выше алгоритм позволяет найти значение угла между осью, на которой расположены микрофоны, и прямой соединяющей центр оси и источник звука. Угол лежит в диапозоне $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Алгоритм не разделяет переднюю (относительно микрофонов) и заднюю полуплоскость, так как по рассмотренным измерениям невозможно однозначно восстановить угол.

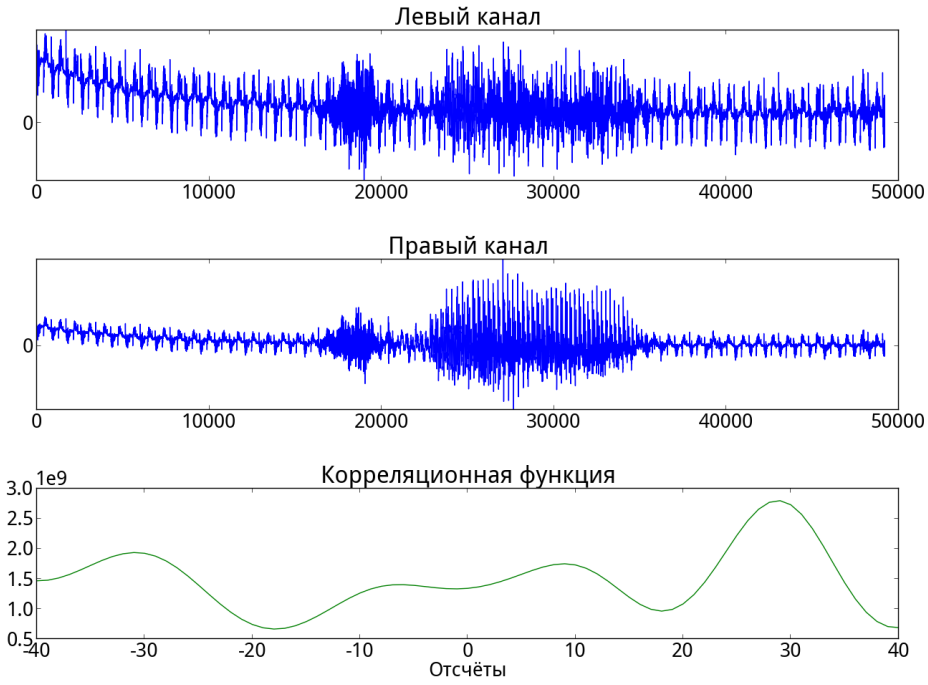

Рис. 2: Значение корреляционной функции для сигналов с двух микрофонов. Диктор находится под углом $-90°$ относительно оси микрофонов

2.2. Выделение активных фрагментов звукового потока

Помимо определения направления на источник звука, датчик также должен уметь отличать участки входного сигнала, содержащие полезный сигнал, от участков, содержащих только фоновый шум. Проблема обнаружения голосовой активности (англ. VAD - voice activity detection) - часто встречающаяся в цифровой обработке сигналов задача. Для классификации речевого фрагмента как активного/неактивного используются различные параметры звукового сигнала, такие как энергия сигнала, zero-crossing rate, спектральная плоскостность, пик-фактор и другие. В данной работе рассматривается алгоритм, основанный на кратковременной энергии сигнала (англ. short-term energy).

Энергия дискретного звукового сигнала определяется соотношением:

$$E = \frac{1}{n}\sum_{i=1}^{n} x[i]^2.$$

Входной звуковой сигнал делится на участки (фреймы) одинаковой длины, для каждого вычисляется его энергия. Затем величина энергии фрейма E_j сравнивается с пороговым значением E_t, если выполняется соотношение $E_j > E_t$, то фрейм считается активным. В реализации алгоритма для двух звуковых сигналов значение энергии фрейма полагалось равным максимуму из значений энергии первого и второго сигналов $E_j = max\{E_j^1, E_j^2\}$.

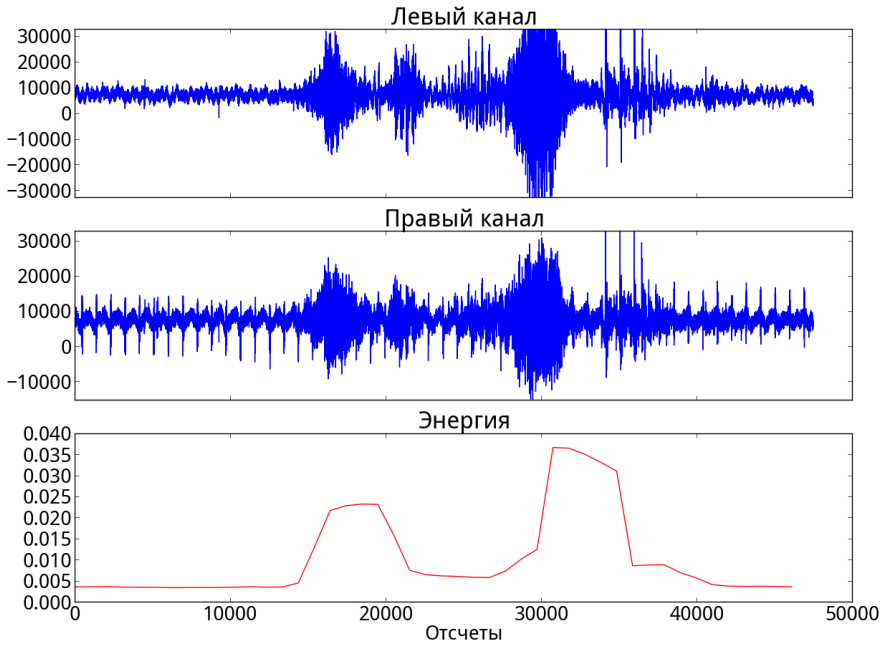

Рис. 3: График энергии фрейма для двух сигналов. Длина фрейма порядка 100 мс

На графике можно видеть зависимость энергии от входных сигналов. Значения

энергии взяты с шагом 1024 отсчёта и длиной окна 100 миллисекунд (т.е. значение на графике в точке равно значению энергии на предыдущих 100 миллисекундах сигнала). Частота дискретизации 44100 Гц.

Приведенный выше алгоритм был выбран для реализации так как он обладает такими преимуществами как относительная простота реализации и низкое потребление вычислительных ресурсов. К недостаткам следует отнести зависимость порогового коэффициента от уровня фонового шума и деградацию алгоритма при высоком уровне шума.

2.3. Фильтрация сигнала

Цифровой фильтр - это система, выполняющая преобразования над цифровым сигналом с целью выделения или подавления определенных частот. В общем случае, дискретный фильтр суммирует с весовыми коэффициентами некоторое количество входных сэмплов и некоторое количество предыдущих выходных отчетов [7]:

$$y[n] = b_0x[n] + b_1x[n-1] + \ldots + b_mx[n-m] - a_1y[n-1] - a_2y[n-2] - \ldots - a_ky[n-k]$$

Коэффициенты фильтра подбираются исходя из анализа входного сигнала, спектра шумов, полезного сигнала и других характеристик. Для того чтобы снизить влияние шума на показания датчика в работе был реализован цифровой фильтр.

3. Описание платформы

Контроллер ТРИК [6] имеет центральный процессор OMAP-L138 C6-IntegraTM DSP+ARM [3] компании Texas Instruments. Это гетерогенная архитектура, включающая процессор ARM общего назначения и процессор DSP, оптимизированный для обработки цифровых сигналов. Контроллер имеет оперативную память объемом 256 Mb, которую разделяют процессоры. ARM процессор работает под управлением операционной системы Linux.

Межпроцессорное взаимодействие осуществляется при помощи фреймворка Codec Engine [2], разработанного компанией Texas Instruments. К сожалению, текущая конфигурация фреймворка для ТРИКа позволяет запуск только одного Codec Engine алгоритма на стороне DSP. Такое ограничение не позволяет запустить на роботе одновременно, например, алгоритм обработки аудио и видеопотока. Конечно, можно обойти эту сложность и передавать Codec Engine серверу аудио и видео данные, а на стороне DSP алгоритма выполнять разделение этих данных и запуск нескольких задач. Однако, такой подход в конечном итоге приведет к написанию функциональности, аналогичной функциональности самого Codec Engine, сконфигурированного для запуска нескольких задач. Не факт, что самописное решение будет надежнее и удобнее варианта компании Texas Instruments.

По этой причине, а также исходя из соображений, что используемый алгоритм детекции направления на источник звука не потребляет значительных вычислительных ресурсов, было решено разрабатывать все компоненты системы для процессора ARM.

4. Реализация датчика

4.1. Программные датчики ТРИК

Программными датчиками будем называть датчики, которые, анализируя данные поступающие с перефирийных устройств, выдают некоторые значения. На данный момент на ТРИКе уже реализованы подобные датчики, например object sensor, анализирующий видеопоток и выделяющий в нем объекты заданного цвета. Все подобные сенсоры имеют некоторый общий интерфейс для взаимодействия с другими подсистемами ТРИКа.

Программный датчик работает как демон, который может быть запущен или остановлен запуском bash скрипта. Датчик пишет свои показания в именнованный канал (fifo), откуда они могут быть считаны другим приложением. Также допускается существованние второго fifo для передачи команд датчику во время работы.

Для интеграции с другими приложениями, разработанными для ТРИК, датчик направления на источник звука поддерживает такой же интерфейс.

4.2. Архитектура приложения

В качестве языка реализации был выбран C++, так как этот язык поддерживает ручное управление памятью и компилируется в машинный код, что может оказаться критически важным при разработке программы, работающей в режиме реального времени. Кроме того, C++ позволяет писать высокоуровневый абстрактный код. Также, на C++ написано множество отлаженных и оптимизированных библиотек.

Для хранения звукового потока был использован кольцевой буффер. Этот контейнер имеет фиксированный размер, выделяет память только при создании и позволяет записывать новые данные на место устаревших. Благодаря этим свойствам, данный тип контейнера идеально подходит для хранения и обработки постоянно поступающих данных. За основу кольцевого буффера был взят класс circular_buffer из библиотеки boost [1]. Для захвата данных с микрофонов использовался модуль QtMultimedia [4] из фреймворка Qt. QtMultimedia предоставляет платформонезависимый способ записи медиа данных в произвольный объект, поддерживающий интерфейс QIODevice.

Так как и детектор направления на звук, и фильтр сигнала, и vad-алгоритм, по сути, выполняют очень похожие действия, то есть принимают последовательность данных и, производя какие-то вычисления над ней, обновляют своё состояние, было решено наследовать реализации этих алгоритмов от общего интерфейса AudioFilter<Iterator>. AudioFilter<Iterator> является шаблонным классом, параметризуемым типом итератора. Пара итераторов задаёт последовательность (range) элементов. AudioFilter<Iterator> выполняет обработку (и, возможно, модификацию)

входной последовательности и обновляет свое состояние. Благодаря такой архитектуре, обработчики последовательности не зависят ни от типа контейнера, хранящего аудио данные, ни от типа его обхода.

Несколько различных объектов, поддерживающих интерфейс AudioFilter<Iterator> могут быть объединены в цепочку фильтров с помощью класса AudioPipe<Iterator>. Единственной задачей этого класса является хранение списка фильтров и последовательное их приминение к входному потоку. Для обработки двухканального сигнала реализован аналогичный интерфейс StereoAudioFilter<Iterator>, единственным отличием которого является то, что он принимает на вход две последовательности.

Во время разработки и отладки программы, было полезно иметь возможность считывания аудио сигнала не только с микрофонов, но и из wav файлов. Для того чтобы код обработки данных не зависел от способа их получения, был введен интерфейс AudioStream. Для объединения поставщика аудио сигнала и последовательности фильтров служит класс TrikSoundController.

Так как система должна обеспечивать отклик в реальном времени, входной поток разбивается на фреймы одинаковой длины. Вся последовательность фильтров применяется к обрабатываемому фрейму, после чего все объекты-обработчики опрашиваются контроллером с запросом выдать новые показания. Длина фрейма подбирается достаточно малой - от 256 до 2048 сэмплов, что при частоте дискретизации 44100 гц равно 5-50 мс. Однако, такая последовательность может оказаться слишком короткой для принятия решения используемыми алгоритмами. Поэтому обработчики сохраняют промежуточные результаты своей работы и выдают новые показания на основе истории. VAD-алгоритм должен вычислять энергию для окна сигнала порядка 100 мс, поэтому соответствующий объект, при размере фрейма равном 1024 отсчётам, хранит результат предыдущих вычислений для 5 фреймов.

Часть вычислений, выполняемых датчиком, использует вещественные числа. Например, коэффицинты цифрового фильтра - вещественные числа. Из-за отсутсвия аппаратной поддержки вычислений с плавающей запятой на процессоре ARM, было решено использовать вычисления с фиксированной точкой. Для этой цели была использована библиотека Fixed Point Math Library [5].

4.3. Реализация детектора направления

Контроллер подает на вход детектора звуковые фреймы порядка 256-2048 отсчётов. От детектора ожидается значение угла. Однако, если вычислять величину для таких коротких фреймов, показания датчика будут сильно колебаться на каждом шаге. Для того чтобы сгладить кривую показаний, детектор хранит историю вычислений.

Воспользовавшись свойством корреляционной функции:

$$corr(f,g)[n] = \sum_{m=0}^{K} f[m]g[n+m] = \sum_{m=0}^{K_1} f[m]g[n+m] + \ldots + \sum_{m=K_{j-1}}^{K_j} f[m]g[n+m]$$

замечаем, что значение функции на длинном сигнале восстанавливается по значениям на более коротких.

Чтобы ответить на запрос, детектору необходим массив значений корреляционной функции на отрезке $[-L, L]$. Объект хранит список массивов размера $2L$ со значениями корреляционной функции для предыдущих N фреймов. Когда поступает запрос угла, детектор суммирует все хранящиеся массивы, чтобы получить значения корреляционной функции для сигнала длины $N * size_{fr}$, где $size_{fr}$ - длина фрейма. Затем находится максимум функции и вычисляется угол по формулам, описанным в 2.1.

5. Оценка работы датчика

5.1. Оценка показаний

После реализации датчика было проведено тестирование программы на контроллере в реальных условиях. Эксперименты проводились как в режиме работы датчика в реальном времени, так и на заранее подготовленных записях. Во всех тестах использовался следующий формат аудио:

- Частота дискретизации - 44100 Гц.
- Тип сэмпла - 16 битное знаковое целое.

По экспериментам, проведенным с аудио потоком захватываемым в реальном времени, можно в целом судить, что датчик распознает угол направления на звук корректно, хотя и с существенной погрешностью. Колебания сенсора могут составлять до 10° в обе стороны. Также было отмечено, что датчик плохо справляется с источниками звука, расположенными под углами 90° и -90°. Можно сделать вывод, что датчик способен распознать лишь сектор с дугой порядка 20°. Впрочем, для робота, который, услышав произнесенную диктором фразу, должен повернуться к источнику сигнала, такой точности может оказаться вполне достаточно.

Для того чтобы каким-то образом количественно оценить работу сенсора, было записано порядка 50 коротких фраз, произнесенных под разными углами. Для каждой записи вычислялось среднеквадратичное отклонение показаний датчика от ожидаемой величины. Также целью этого эксперимента было сравнение зависимости показаний детектора от длины истории (то есть длины окна, на котором вычисляется корреляционная функция). Интуитивно понятно, что данная длина должна совпадать с длиной ожидаемого голосового сигнала. При оценке результатов было установлено, что при короткой длине истории показания датчика сильно скачут на каждом шаге, а при увелечении длины сглаживаются и приближаются к прямой, равной значению угла.

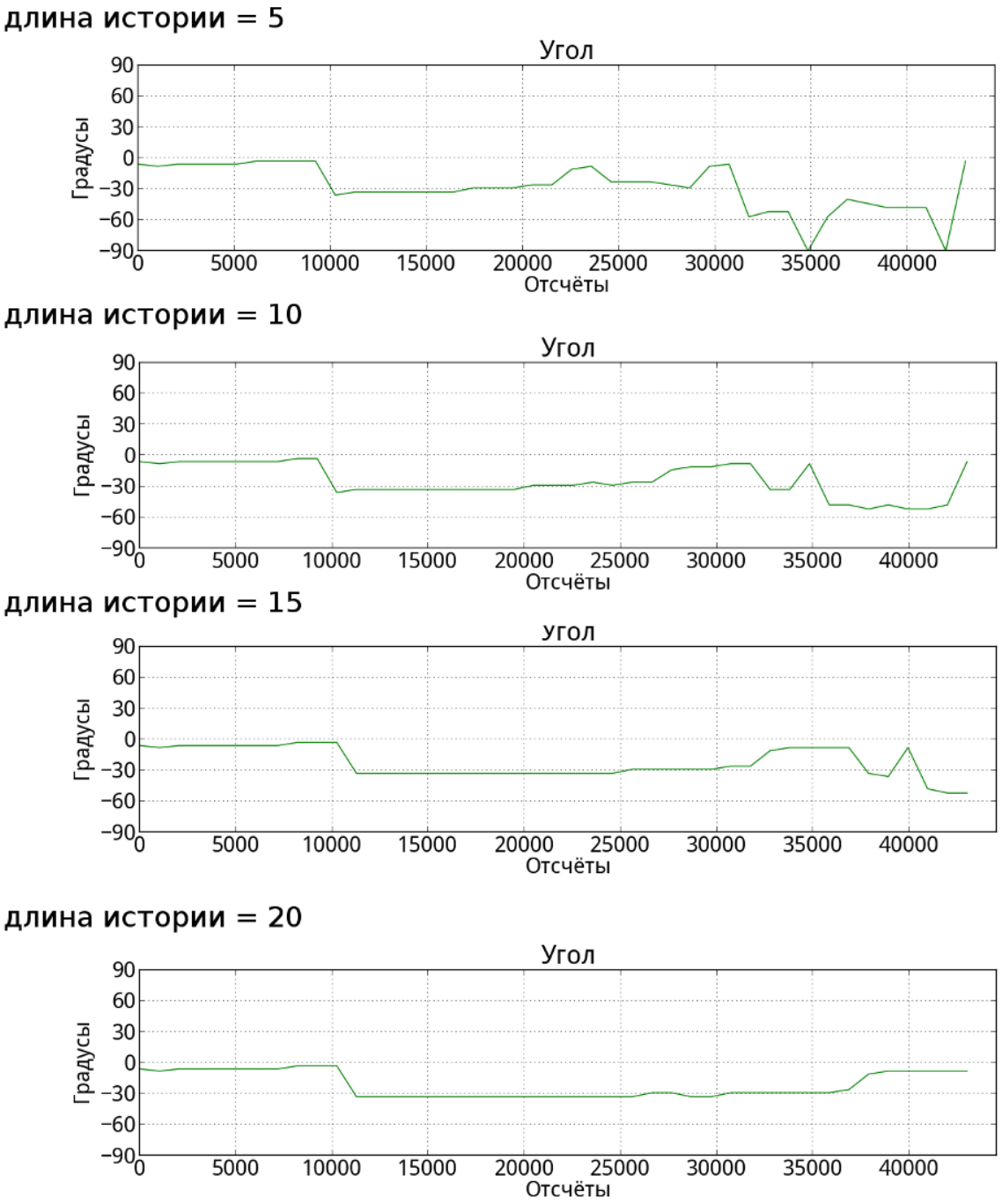

Рис. 4: Кривая показания датчика при различной длине истории. Диктор находится под углом $-30°$ относительно центра оси микрофонов

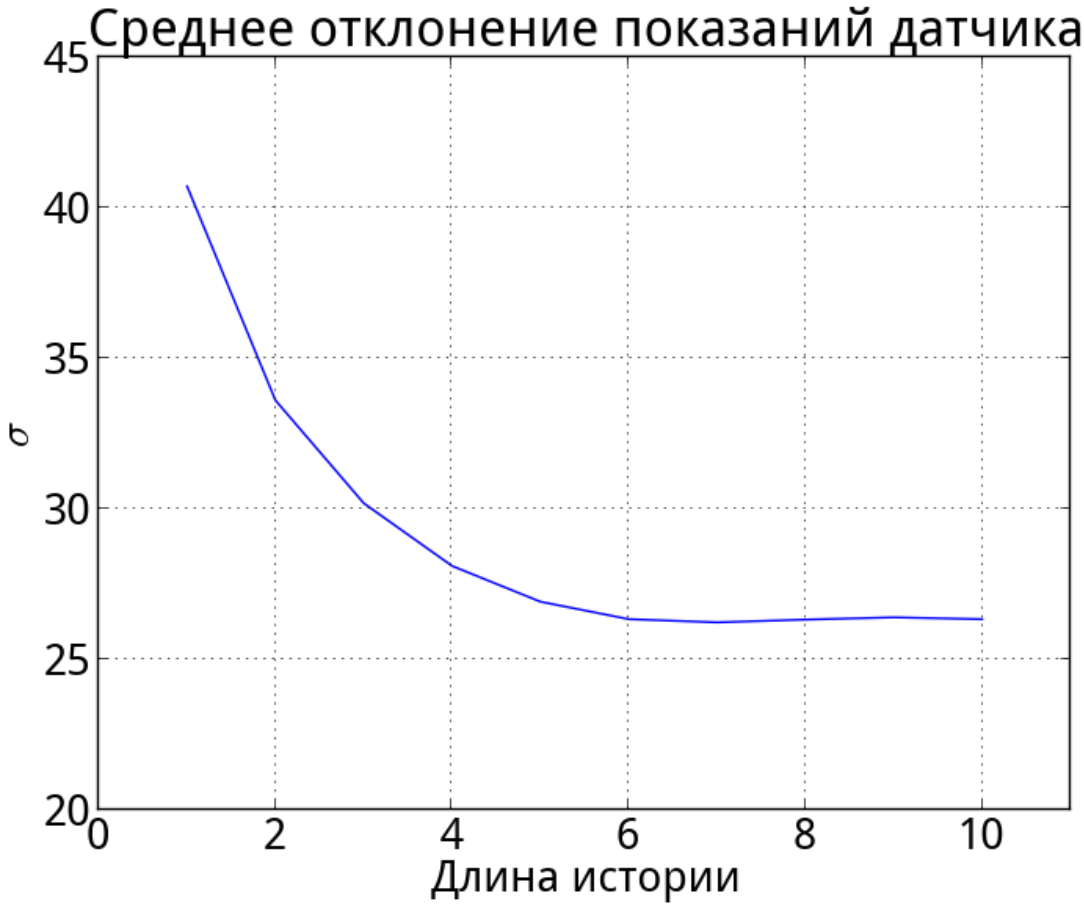

Рис. 5: Среднеквадратичное отклонение показаний от ожидаемого значения в зависимости от длины истории. На графике показано среднее значение по 52 тестам

Глядя на график зависимости отклонения показаний датчика от длины истории видно, что величина отклонения уменьшается при большей длине истории.

5.2. Исследование скорости работы

Одно из главных требований к программе - способность с минимальной задержкой обновлять показания при возникновении звукового сигнала. Чтобы добиться этого, было решено разделять входной поток на короткие фреймы и выдавать новые значения сразу после обработки фрейма. Чтобы успевать обрабатывать аудио сигнал, датчик должен выполнять все вычисления за время меньшее, чем длина самого фрейма.

Мной были написаны бенчмарки, запускающие алгоритмы обработки сигнала (детектор угла, vad, фильтр) на случайных данных. Количество итерации каждого алгоритма равно 1000. Были протестированы фреймы размеров 256, 512, 1024 и 2048 сэмплов. Оценивалось среднее время работы в миллисекундах на каждой итерации и процент времени работы алгоритма от длительности фрейма (при частоте дискретизации 44100 Гц). Результаты замеров приведены в таблицах.

Таблица 1: Время работы детектора

Размер фрейма	Среднее время работы в миллисекундах	Среднеквадратическое отклонение	% от длины окна
256	0.5300	0.0042	9.13
512	1.500	0.0093	12.9
1024	3.4600	0.0147	14.9
2048	7.4200	0.0213	15.9

Таблица 2: Время работы vad

Размер фрейма	Среднее время работы в миллисекундах	Среднеквадратическое отклонение	% от длины окна
256	0.014	0.0008	0.24
512	0.029	0.0013	0.24
1024	0.057	0.0015	0.24
2048	0.12	0.0024	0.25

Глядя на замеры, можно сделать вывод, что лучшее отношение времени работы к длине фрейма достигается на окнах размером 256 сэмплов. Данный результат можно объяснить тем, что при короткой длине обрабатываемой последовательности данные лучше кэшируется процессором. Также видно, что суммарно все алгоритмы обрабатывают фрейм примерно за 25% от его длительности.

Таблица 3: Время работы фильтра

Размер фрейма	Среднее время работы в миллисекундах	Среднеквадратическое отклонение	% от длины окна
256	0.733	0.0045	12.63
512	1.494	0.0101	12.8
1024	3.044	0.0159	13.1
2048	6.33	0.0194	13.6

Результаты

В ходе работы был реализован датчик направления на источник звука, работающий в режиме реального времени на контроллере ТРИК. Датчик был протестирован, оценены его показания и потребление вычислительных ресурсов. Также были созданы вспомогательные утилиты для тестирования и визуализации результатов работы программы. В целом, работу сенсора можно признать удовлетворительной.

Исходный код проекта доступен по адресу https://github.com/EvgeniyMsk/sound_direction

Список литературы

[1] Boost.Circular Buffer. — URL: http://www.boost.org/doc/libs/1_57_0/doc/html/circular_buffer.html (online; accessed: 24.05.2015).

[2] Codec Engine. — URL: http://processors.wiki.ti.com/index.php/Codec_Engine (online; accessed: 24.05.2015).

[3] OMAP-L138 C6-IntegraTM DSP+ARM. — URL: http://www.ti.com/product/omap-l138 (online; accessed: 24.05.2015).

[4] Qt 4.8: QtMultimedia. — URL: http://qt.apidoc.info/4.8.5/qtmultimedia.html (online; accessed: 24.05.2015).

[5] Schregle Peter. Fixed point math library. — URL: http://www.codeproject.com/Articles/37636/Fixed-Point-Class (online; accessed: 24.05.2015).

[6] Робототехнический контроллер ТРИК. — URL: http://www.trikset.com/ (online; accessed: 24.05.2015).

[7] Сергиенко А.Б. Цифровая обработка сигналов. — ИД Питер, 2003.