Межрегиональная ассоциация
экспериментальной психологии

Московский институт психоанализа

ЛИЦО ЧЕЛОВЕКА: ПОЗНАНИЕ, ОБЩЕНИЕ, ДЕЯТЕЛЬНОСТЬ

Под редакцией

К. И. Ананьевой, В. А. Барабанщикова, А. А. Демидова

Москва
Когито-Центр
2019

УДК 159.9
ББК 88
Л 65

Л 65 **Лицо человека: познание, общение, деятельность** / Отв. ред. К. И. Ананьева, В. А. Барабанщиков, А. А. Демидов. – М.: Когито-Центр–Московский институт психоанализа, 2019. – 568 с.
ISBN 978-5-89353-572-3

УДК 159.9
ББК 88

Коллективная монография, подготовленная ведущими отечественными специалистами, посвящена обсуждению широкого круга вопросов, касающихся изучения лица человека. Лицо человека – уникальный феномен, возникающий на перекрестье природных и социокультурных детерминант и находящий свое отражение в различных сферах человеческого бытия и практики. Главные темы, которые затрагиваются в данном издании: исследование лица в контексте междисциплинарных связей и отношений, распознавание эмоциональных состояний и индивидуально-психологических качеств человека по выражению его лица, особенности межличностного восприятия в процессах социального взаимодействия, расовые и этнические детерминанты межличностного познания и др. Книга ориентирована на специалистов из различных областей научного знания и общественной практики, интересующихся феноменом человеческого лица.

ISBN 978-5-89353-572-3

Глава 4

ОРГАНИЗАЦИЯ ИССЛЕДОВАНИЙ МЕЖЛИЧНОСТНОГО ОБЩЕНИЯ, ОПОСРЕДОВАННОГО ВИДЕОКОММУНИКАЦИЕЙ

А. В. Жегалло, И. А. Басюл

Ранее проводившиеся нами исследования восприятия лица человека (Барабанщиков, Жегалло, Куракова, 2016) основывались на парадигме викарного общения (общение с заместителем). В качестве стимульного материала использовались фото- и видеоизображения натурщиков. При этом регистрация движений глаз выполнялась с привязкой к изображениям, демонстрировавшимся на экране компьютера, что обеспечивало относительно высокое качество записей глазодвигательной активности и возможность последующего анализа с привязкой к рассматриваемому изображению. Недостатком данной парадигмы является значительная ограниченность исследуемого диапазона выражений лица коммуникантов и низкая экологическая валидность эксперимента в целом.

Оптимальным в плане экологической валидности является исследование межличностного общения в реальной коммуникационной ситуации. Известный нам на сегодняшний день опыт таких исследований (Фёдорова, 2017; Kibrik, Fedorova, 2018) показывает возможность полипозиционной видеорегистрации и качественной индивидуальной аудиозаписи реплик каждого из участников. Регистрация движений глаз в реальной коммуникационной ситуации предполагает использование мобильных устройств. За последние годы достигнуто значительное сокращение размеров регистрирующего айтрекингового оборудования при одновременном улучшении его технических характеристик. В то же время доступное оборудование

1 Исследование выполнено при финансовой поддержке РНФ, проект № 18-18-00350.

ориентировано скорее на изучение взаимодействия человека с окружающей средой, воспринимаемой полным широким полем зрения, чем на исследования межличностного восприятия. Так, у устройства Tobii Pro Glasses 2 угол зрения камеры сцены составляет 82°×52°; таким образом диагональный угол зрения составляет 97°. У айтреке-ров Pupil Labs в зависимости от комплектации угол зрения камеры сцены по диагонали может составлять 60°, 90°, 100°. Принимая размеры лица человека анфас 18×25 см, получаем угловые размеры на расстоянии 1 м – 10°×14°; на расстоянии 2 м – 5°×7°. Таким образом, фактическая пространственная разрешающая способность айтрекера значительно уменьшается.

На сегодняшний день приемлемым компромиссным решением нам представляется парадигма комбинированного эксперимента в диаде с опосредованием общения между участниками техническими средствами видеокоммуникации. Видеоизображение лиц участников эксперимента регистрируется компактными камерами, расположенными на верхней кромке экранов компьютеров. Изображение записывается и выводится на экран собеседника, занимая примерно половину площади экрана. Оставшаяся часть используется для вывода материала, относящегося к совместно решаемой задаче. Регистрация движений глаз выполняется стационарными айтрекерами, откалиброванными в координатах экрана. Угловой размер экрана по горизонтали не превышает 30°.

Основная проблема при проведении такого исследования состоит в высоких требованиях к видеосистеме. Видеоизображение должно одновременно записываться и передаваться на компьютер одного из участников, в программу, контролирующую экспериментальную процедуру. Доступные на рынке автономные видеокамеры обеспечивают высококачественную видео- и аудиозапись (Full HD, 50 fps и выше), но при этом не поддерживают передачу изображения на компьютер во время записи. Автономные камеры охранных систем поддерживают одновременную запись на SD-карту и трансляцию видео по сети, но при этом имеют относительно низкую частоту регистрации (как правило, не выше 30 fps). При этом стандарт передачи данных не документируется, а передача данных возможна лишь в специализированные приложения. Оптическая система на охранных камерах выбирается преимущественно исходя из требований максимальной компактности и ограничивается номенклатурой объективов с резьбой M12.

Наиболее перспективным решением нам представляются камеры, подключаемые к компьютеру по интерфейсам USB 2.0/USB 3.0. На сегодняшний день на рынке доступна широкая номенклатура

подобных устройств, отличающихся весьма высокими техническими характеристиками при умеренной цене. Рассмотрим несколько примеров. Камера Sony PlayStation Eye PS3 с сенсором OV7725 позволяет получить некомпрессированное изображение 640×480 60 fps и одновременно – высококачественную аудиозапись. Объектив с полем зрения 65°/75°. Камера ELP-USBFHD01M с сенсором OV2710 передает компрессированное (MJPEG) изображение 640×480 120 fps, 800×600 60 fps, 1280×720 60 fps. Объектив вариофокальный 5/50 мм с C/CS монтировкой. Камера KAYETON KYT-U400-MCS0660R01 с сенсором OV4689 передает компрессированное (MJPEG) изображение 1920×1080 MJPEG 60 fps, 1280×720 MJPEG 120 fps, 640×360 MJPEG 330 fps. Объектив вариофокальный 6/60 мм с C/CS монтировкой. Все три перечисленные камеры подключаются к компьютеру по интерфейсу USB 2.0, поддерживаются в Linux стандартными драйверами. Камеры ELP и KAYETON также поддерживаются в Windows, для Sony PlayStation Eye требуется коммерческий драйвер от Code Labs.

Оборудование, подключаемое по USB 3.0, позволяет получать некомпрессированное изображение высокого разрешения. Так, например, камера MDvision MD-SUA133GC-T поддерживает разрешение 1280×1024 240 fps. Обеспечить на практике стабильную работу данной камеры нам не удалось, поскольку для этого требуется компьютер с высокой скоростью записи на диск.

Камеры ELP и KAYETON стабильно работают под Linux (Lubuntu 18.04) под управлением ПО GStreamer. Поддерживается одновременная запись на диск без повторного сжатия, контроль видеоизображения и передача изображения по локальной сети. Размер сохраняемого на диск видео составляет 7 мбайт/мин (640×480 MJPEG 120 fps); 15 мбайт/мин (1280×720 MJPEG 120 fps).

GStreamer представляет собой OpenSource библиотеку для обработки мультимедиа, поддерживающую широкий диапазон вариантов обработки аудио- и видеопотоков. Гибкость обработки обеспечивается соединением между собой отдельных готовых элементов, реализующих захват данных из разных источников, их преобразование и вывод на конечные устройства. В простейшем варианте последовательность элементов объединяется в необходимом порядке в контейнер верхнего уровня (pipeline) и выполняется с помощью штатного приложения gst-launch-1.0. Командная строка для gst-launch-1.0 фактически представляет собой инструкцию, задающую порядок соединения отдельных элементов. Для решения более сложных задач имеются API, позволяющие использовать функционал GStreamer из стандартных языков программирования (в частности C и Python).

Недостатком простейшего варианта реализации работы с камерами с использованием gst-launch 1.0 оказалась невозможность обеспечить синхронную запись видео и звука. Анализ проблемы показал, что в GStreamer поддержка режима аудио/видеозаписи с компрессией выполнена лишь в ограниченном объеме. Единственный стабильно работающий кодек – Theora, разработанный фондом Xiph.org как часть проекта Ogg. Принципиальным недостатком реализации данного кодека в GStreamer является однопотоковый режим работы, что приводит к невозможности видеозаписи в высоком разрешении даже на современных компьютерах.

Возможным решением данной проблемы является совместное использование библиотек GStreamer и ffmpeg в рамках программы на языке программирования C. Основное назначение GStreamer состоит в обеспечении передачи видеоизображения по локальной сети. Кроме того, средствами GStreamer выполняется захват изображения с камеры, контроль изображения, захват звука с микрофона. Библиотека ffmpeg используется для кодирования аудио- и видеопотоков, их объединения и записи в файл. Рассмотрим более подробно конкретные программные решения, реализующие указанный функционал.

Особенности использования библиотеки ffmpeg

Примеры программ, реализующих интересующую нас функциональность ffmpeg, содержатся в дистрибутиве исходного кода программы (папка doc/examples). Нас интересуют примеры кодирования и записи аудиопотока (файл encode_audio.c); кодирования и записи видеопотока (файл encode_video.c); совместной записи аудио- и видеоданных (файл muxing.c). Рассмотрим подробнее приемы реализации данных задач.

Для выполнения операции кодирования (encoding) используется один из кодеков, реализованных в библиотеке ffmpeg. В примере записи аудиопотока – это кодек с идентификатором AV_CODEC_ID_MP2. При запуске откомпилированного примера мы обнаружили, что система «не видит» имеющиеся кодеки. Для решения этой проблемы перед вызовом функции поиска кодека avcodec_find_encoder (AV_CODEC_ID_MP2) был добавлен вызов функции avcodec_register_all(), которая, согласно документации, выполняет регистрацию имеющихся в системе кодеков (codec), обработчиков (parser) и фильтров (filter).

В результате выполнения функции avcodec_find_encoder заполняется структура типа AVcodec. Поля name и long_name содержат

название кодека (при выполнении нами рассматриваемого примера – «mp2» и «MP2 (MPEG audio layer 2)». Поле supported_samplerates – набор поддерживаемых частот дискретизации аудиосигнала. Поле sample_fmts – набор кодов поддерживаемых форматов данных. Поле channel_layouts – набор кодов поддерживаемых расположений источников звука (моно, стерео и т. п.).

Помимо кодека для выполнения кодирования необходим также контекст кодирования (структура данных типа AVCodecContext), создаваемый функцией avcodec_alloc_context3(). Для дальнейшей работы нам важны следующие поля структуры: bit_rate – среднее количество бит данных, используемых для обработки в единицу времени, sample_fmt, sample_rate, channel_layout, channels – число каналов.

В разбираемом примере предполагается, что кодирование звука будет выполняться на частоте 44100 Гц или ближайшей к ней, поддерживаемой кодеком, битрейт составляет 64000 бит/с, число каналов – максимальное поддерживаемое кодеком, формат данных – AV_SAMPLE_FMT_S16 (16 бит данных со знаком на один сэмпл данных).

Для начала работы требуется предварительно выполнить настройку параметров и инициализацию кодека (функция avcodec_open2). После этого в контексте становится известным размер рабочей области данных (поле frame_size). Далее выполняется открытие на запись выходного файла (функция fopen).

Кодирование аудиопотока требует предварительного выделения памяти для исходных данных (функция av_frame_alloc) и результата кодирования (функция av_packet_alloc), После этого для фрейма, описывающего исходные данные, задаются параметры кодирования: формат, число и тип каналов, размер кадра. На основании заданной информации выделяется память под буфер, содержащий «сырые» данные. Далее данный буфер в цикле заполняется значениями, соответствующими гармоническому тональному сигналу. По заполнении буфера выполняется его кодирование (encoding).

Операция кодирования видеопотока выполняется аналогичным образом. В качестве аргументов программе encode_video передаются имя выходного файла и имя кодека. Список доступных для ffmpeg кодеков для записи данных можно посмотреть командой ffmpeg-encoders. Поскольку мы выполняем запись видеопотока, то нас интересуют кодеки с установленным признаком V (отбор может быть выполнен с использованием команды grep: ffmpeg-encoders | grep V). Перед поиском кодека так же, как и в случае кодирования аудиопотока, необходимо выполнить функцию avcodec_register_all. Следует отметить, что выбираемые далее параметры исходного видео совмес-

тимы не со всеми доступными кодеками. Для обеспечения максимально возможной эффективности работы рекомендуется использовать кодек libx264.

Тип данных для контекста, связанного с кодеком, устанавливается как AV_PIX_FMT_YUV420P. Цветовая модель YUV задает 3-компонентное кодирование изображения, разлагаемого на компоненту яркости Y и две цветоразностные компоненты U и V. Каждой точке изображения соответствует байт данных Y. Цветоразностные компоненты имеют общие значения для блока 2×2 точки. Выбор типа данных IUV420P в данном случае определяется тем, что он поддерживается имеющимися в составе библиотеки ffmpeg кодеками, обеспечивающими высокоэффективное сжатие с потерями (в частности H264). Буфер фрейма, предназначенный для исходного видеоизображения, содержит три массива данных, предназначенных для Y, U и V компонент изображения соответственно. По окончании кодирования видеоданных в выходной файл записывается конечная 4-байтная последовательность: 0, 0, 1, 0xb7.

При совместном кодировании аудио- и видеопотоков в качестве входного аргумента передается имя файла. Последующие аргументы рассматриваются как набор дополнительных параметров для ffmpeg. Кодеки, используемые для записи, выбираются автоматически функцией avformat_alloc_output_context2. В результате инициализируется структура AVFormatContext. Предварительно необходимо выполнить функцию av_register_all. При запуске примера с указанием выходного файла с расширением.avi использовались кодеки mpeg4 (видео) и mp3 (аудио). В случае выходного файла с расширением.mp4 – кодеки h264 (видео) и AAC (аудио). Структурная схема примера представлена на рисунке 1. Инициализация кодеков выполняется функцией add_stream. Помимо заданного кодека функция инициализирует также соответствующий поток (stream), содержащий, в частности, информацию о временной отметке, указывающей на текущий момент времени в потоке записываемых данных. Далее выполняются функции open_audio и open_video, в рамках которых выделяются блоки памяти для операций кодирования и задаются необходимые параметры.

Функция av_dump_format выводит детальную информацию о формате выходного потока. Вывод для случая .avi файла выглядит следующим образом:

```
Output #0, avi, to 'tst.avi':
Stream #0:0: Video: mpeg4, yuv420p, 352×288, q=2–31, 400 kb/s, 25 tbn
Stream #0:1: Audio: mp3, 44100 Hz, stereo, s32p, 64 kb/s
```

Открытие файла на запись и необходимые подготовительные операции выполняются функцией avio_open. Заголовок файла записывается av_format_write_header.

Собственно запись аудио- и видеопотока выполняется в цикле поочередным вызовом функций write_video_frame и write_audio_frame. Цикл завершается, когда обе функции вернут признак, указывающий, что записано необходимое число пакетов. Выбор того, какую из функций вызывать, принимается путем сравнения текущего времени в потоках функцией av_compare_ts. Если текущее время в видеопотоке меньше, чем в аудиопотоке, то выполняется запись очередного кадра видео, иначе – аудио. Частота записи видео составляет 25 кадров/с, таким образом, каждому кадру видео соответствует интервал продолжительностью 40 мс. Продолжительность временного интервала, соответствующего одному блоку аудиоданных, определяется его размером, при выполнении рассматриваемого примера она составляла 26,1 мс для кодека mp3; 23,2 мс для кодека aac.

Завершающая последовательность данных записывается функцией av_write_trailer. По завершении записи выполняется закрытие потоков функцией close_stream. Далее функцией avio_closep закрывается контекст ввода/вывода. В заключение выполняется освобождение контекста с помощью avformat_free_context.

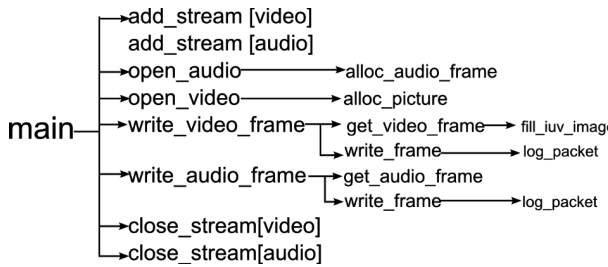

Рис. 1. Основная структура стандартного примера muxing.c

Особенности использования библиотеки GStreamer

Основные приемы работы с GStreamer наглядно показаны в стандартном примере[1]. В примере реализуется проигрывание аудиофайла в формате Ogg/Vorbis. Для начала работы с GStreamer выполняется его инициализация вызовом функции gst_init. Далее выполняется создание pipeline функцией gst_pipeline_new. Необходимые элементы создаются функцией gst_element_factory_make. Цепочка элемен-

1 URL: https://gstreamer.freedesktop.org/documentation/application-development/basics/helloworld.html.

тов включает: источник filesrc, выполняющий чтение файла с диска; демультиплексор oggdemux, выполняющий задачу извлечения потока из файла-контейнера; декодер vorbisdec, декодирующий аудиопоток; конвертер audioconvert; элемент, выполняющий воспроизведение звука autoaudiosink (рисунок 2).

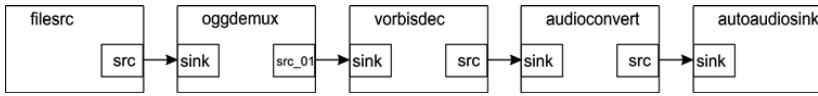

Рис. 2. Цепочка элементов (pipeline), выполняющая проигрывание аудиофайла

Указание на проигрываемый файл выполняется изменением свойства location элемента filesrc с помощью функции g_object_set. Обработку сообщений выполняет функция bus_call, подключаемая с помощью функции gst_bus_add_watch. Добавление в pipeline созданных элементов выполняется функцией gst_bin_add_many. Установление связей между элементами реализовано с помощью gst_element_link и gst_element_link_many. Связь между демультиплексором и декодером инициируется как динамическая, создаваемая во время выполнения программы, поскольку у демультиплексора может быть в принципе несколько выходных соединений (pad), создаваемых в ходе анализа контейнера. Функция g_signal_connect задает вызов функции on_pad_added в момент наступления события «pad-added», т. е. в тот момент, когда стало известно содержание файла-контейнера.

Воспроизведение запускается установкой состояния pipeline в GST_STATE_PLAYING, для чего используется функция gst_element_set_state. Основной цикл программы инициализируется функцией g_main_loop_new, далее запускается на выполнение функцией g_main_loop_run. По окончании воспроизводимого файла функция bus_call получает сообщение GST_MESSAGE_EOS и вызывает функцию g_main_loop_quit, завершающую основной цикл. После этого состояние pipeline устанавливается в GST_STATE_NULL. В завершение уничтожаются pipeline, обработчик сообщений и структура данных, управляющая основным циклом программы, для чего выполняются функции gst_object_unref, g_source_remove, g_main_loop_unref соответственно.

Рассмотренный пример компилируется без ошибок и выполняет поставленную задачу без каких-либо нареканий. Единственная потенциальная проблема – некорректное завершение работы по Cntrl+C.

В этом случае программа завершается без выполнения фрагмента кода, следующего за основным циклом. В случае необходимой нам программы, выполняющей запись и трансляцию аудио- и видеопотоков такое поведение недопустимо, поскольку при остановке программы обязательно должна выполняться запись завершающей кодовой последовательности и закрытие выходного файла.

Для решения данной задачи вводится обработчик сигнала SIGINT, указывающего на остановку процесса пользователем. Установка функции обработчика выполняется путем вызова функции g_unix_signal_add. В результате функция – обработчик intr_handler будет вызываться при нажатии пользователем Ctrl+C. В качестве аргумента intr_handler получает адрес pipeline. Зная его, можно послать в pipeline событие (event), указывающее на завершение потока данных, для чего выполняется вызов gst_element_send_event (pipeline, gst_event_new_eos ()). Обработка полученного события выполняется в функции bus_call и сводится к вызову g_main_loop_quit, завершающему основной цикл программы. После этого выполняется программный код, следующий за вызовом основного цикла. Для корректного завершения работы невызывавшийся обработчик следует явно удалить путем вызова функции g_source_remove.

Реализация захвата аудио- и видеопотока, контроля и трансляции видеоизображения средствами GStreamer

Рассмотрим теперь реализацию в GStreamer интересующей нас задачи обработки потока данных с веб-камеры. Структура pipeline, реализующей работу с видеопотоком представлена на рисунке 3. Захват изображения с камеры реализуется элементом v4l2src, по умолчанию используется устройство /dev/video0. Через согласующий элемент videoconvert данные передаются на элемент-разветвитель tee. Выходные соединения tee формируются по запросу, путем вызова функции gst_element_get_request_pad. Каждому выходному соединению tee ставится в соответствие входное соединение одного из элементов queue, возвращаемое функцией gst_element_get_static_pad. Соединение выходного и входного элементов выполняется с помощью gst_pad_link.

Таким образом формируется три цепочки элементов. Первая, содержащая элементы queue и autovideosink, обеспечивает контрольное воспроизведение видео на экране компьютера. Вторая, включающая элементы queue, videorate, jpegenc, multipartmux, tcpserversink отвечает за передачу данных на клиентский компьютер. Элемент videorate выполняет снижение частоты передаваемых кадров до 20 кад./с, jpegenc –

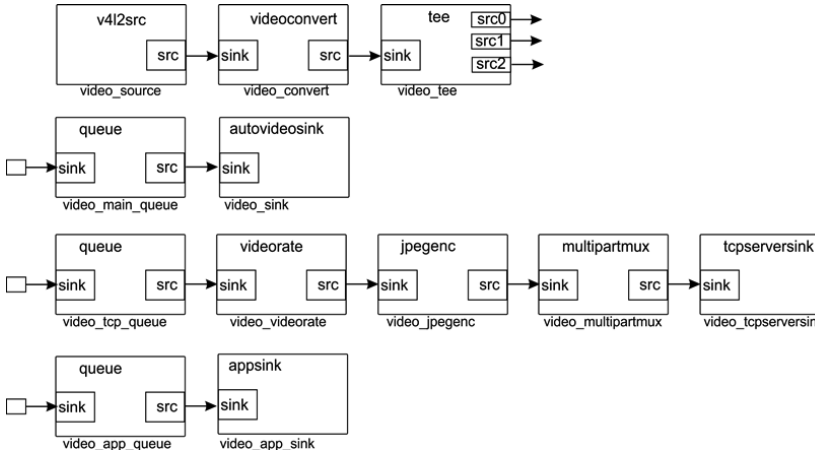

Рис. 3. Pipeline, выполняющая обработку видеопотока.

Примечание: В прямоугольных блоках указаны названия элементов GStreamer. Подписи под блоками – названия соответствующих переменных в описываемой программе

выполняет перекодирование изображения в jpeg, multipartmux – мультиплексирование потока, tcpservesink – передачу потока по сети. Для элемента videorate функцией g_object_set задается свойство rate. Для элемента tcpserversink – свойства host и port. Последняя, третья цепочка состоит из элементов queue и appsink. Задача appsink – передача данных в пользовательскую функцию, задаваемую с помощью g_signal_connect. Обработка аудиопотока сводится к захвату данных с элемента – источника pulsesrc и передаче их в пользовательскую функцию через appsink.

Пользовательские функции new_video_sample и new_audio_sample в качестве входного аргумента получают ссылку на элемент GStreamer appsink, передавший данные (GstElement *sink). Для получения данных выполняется вызов g_signal_emit_by_name (sink, «pull-sample», &sample), передающий в appsink запрос на получение данных. Полученные данные сохраняются в структуру GstSample *sample. Далее из данной структуры можно извлечь служебную информацию о формате данных: gst_sample_get_caps (sample) и собственно буфер данных: gst_sample_get_buffer (sample).

Поскольку GStreamer основан на объектной системе библиотеки Glib – GObject, для доступа к области памяти необходимо выполнить ее блокировку: gst_buffer_map (buffer, &map, GST_MAP_READ). По завершении работы с областью памяти блокировка снимается: gst_buffer_unmap (buffer, &map). В заключение работы дается за-

прос на освобождение памяти, содержащей полученные данные: gst_sample_unref (sample).

В рассматриваемом примере демонстрируется сама концепция использования appsink. Пользовательские функции не производят полезной работы, выполняя только вывод информации о типе получаемых данных и числе вызовов.

Интеграция функциональности записи аудио- и видеопотока в приложение

Итоговое приложение включает в себя элементы реализации, обсуждавшиеся выше: захват и обработка потоков средствами GStreamer, передача данных в пользовательские функции new_video_sample и new_audio_sample, кодирование и запись средствами ffmpeg. Структура итогового приложения приведена на рисунке 4.

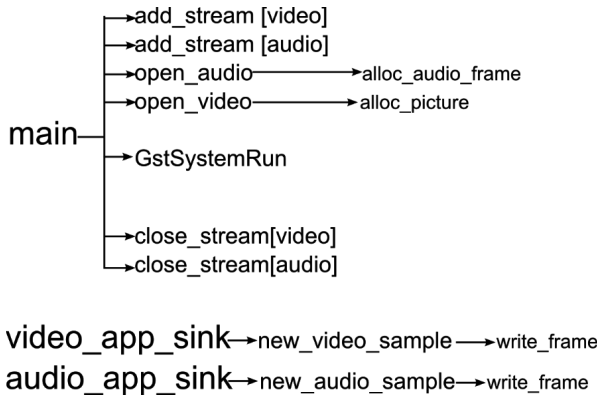

Рис. 4. Основная структура конечной программы, использующей библиотеки ffmeg и GStreamer

Основная проблема при реализации приложения состоит в организации передачи данных из элемента appsink gstreamer в кадр кодека ffmpeg. Для видеопотока данные, полученные после блокировки памяти функцией gst_buffer_map (buffer, &map, GST_MAP_READ), находятся в массиве map.data. Поскольку мы используем цветовую модель IUV, то данный массив включает полный кадр значений яркости Y, а затем два кадра половинного размера с информацией о значениях цветоразностных компонент U и V.

Область данных, куда должна быть перенесена информация – массивы video_st.frame->data [0] – компонента Y, video_st.frame->

data [1] – компонента U, video_st.frame->data [2] – компонента V. За один вызов передается информация о полном видеокадре.

Для аудиопотока данные в map.data при использовании двухканального формата S16LE содержат по два байта данных на каждый отсчет для каждого канала, т.е. один отсчет – 4 байта данных. Используемые в нашей реализации аудиокодеки ffmpeg имеют размер кодируемого блока 1152 сэмпла данных (mp3) либо 1024 сэмпла данных (AAC). Для того чтобы согласовать размер кодируемого блока данных с размером передаваемого appsink блока, с помощью функции g_object_set производится установка свойства blocksize элемента pulsesrc.

Выполненная на сегодняшний день реализация программы показывает принципиальную возможность интеграции библиотек GStreamer и ffmpeg. В то же время стабильность работы и производительность программы недостаточны для решения поставленной задачи одновременного кодирования высокоскоростного видео и стандартного аудиопотока. Анализ работы показывает, что для пары кодеков mpeg4 + mp3 выполнение программы происходит в несколько потоков, соответствующих имеющимся pipeline и queue GStreamer. В частности, кодирование видеоданных выполняется в одном потоке, создающем основную нагрузку. Синхронизация аудио- и видеопотока неудовлетворительная. На 4-ядерном процессоре Intel Core I5-4440 3.1 GHz уровень нагрузки составляет 20%. Для пары кодеков h264+ AAC кодирование видео (encoder libx264) выполняется в многопоточном режиме. На 4-ядерном процессоре Intel Core I5-4440 3.1 GHz уровень нагрузки составляет 40%. Синхронизация аудио- и видеопотока субъективно удовлетворительная. На момент написания статьи остается нерешенной проблема со стабильностью работы программы.

Реализация клиентской части

Клиентская часть реализуется на основе GStreamer. Необходимая pipeline содержит следующие элементы: tcpclientsrc – прием данных от элемента tcpserversink на передающей стороне; multipartdemux – демультиплексирование; jpegdec – декодировнаие jpeg, videoconvert – согласование форматов, appsink – получение данных и передача управления пользовательской функции. Функционал клиентской части реализован в виде класса на языке Python, обеспечивается интеграция с ПО PsychPy (Peirce, 2007). Таким образом, разрабатываемая система видеокоммуникации может быть интегрирована в психологические эксперименты, разрабатываемые с помощью PsychoPy.

Направления дальнейшей работы

Выполненная работа показывает принципиальную возможность эффективной интеграции библиотек GStreamer и ffmpeg, в результате чего полностью реализуются возможности Open Source кодека x264, реализуемого в рамках проекта VideoLAN. Нестабильность работы, предположительно, связана с неполным пониманием нами особенностей использования кодека. Ближайшее направление работы – повышение эффективности сетевой трансляции за счет передачи по сети видео, кодированного в H264, а не в MJPEG. Представляется крайне желательным увеличение размера кадра и частоты запоминаемого видео. Как отмечалось, уже сейчас доступны камеры с разрешением 1280×720 MJPEG 120 fps, однако перекодировка такого потока в H264 может оказаться крайне ресурсоемкой задачей даже на современном оборудовании.

Для сравнения укажем предельные характеристики видеозаписи среди массового оборудования по состоянию на начало 2018 г. Камера Sony DSC-RX10M4 позволяет вести видеозапись в разрешении Full HD 1920×1080 120 fps. YI 4K+ (Plus) Action Camera поддерживает видеосъемку в режимах 1280×720 240 fps; 1920×1080 120 fps. Недостатком последней является широкоугольный объектив, не позволяющий вести качественную съемку лица человека.

Исходный код программных решений, обсуждаемых в статье, доступен через профиль ResearchGate: https://www.researchgate.net/profile/Alexander_Zhegallo.

Литература

Барабанщиков В. А., Жегалло А. В., Королькова О. А. Перцептивная категоризация выражений лица. М.: Когито-Центр, 2016.

Фёдорова О. В. Распределение зрительного внимания собеседников в естественной коммуникации: 50 лет спустя // Е. В. Печенкова, М. В. Фаликман (ред.). Когнитивная наука в Москве: новые исследования. Материалы конференции 15 июня 2017 г. М.: Буки-Веди–ИППиП, 2017. С. 370–375.

Kibrik A. A., Fedorova O. V. Language production and comprehension in face-to-face multichannel communication // Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference “Dialogue 2018”. Moscow, May 30–June 2, 2018.

Peirce J. PsychoPy – Psychophysics software in Python // Journal of Neuroscience Methods. 2007. V. 162. P. 8–13.