

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»»

Факультет информатики, математики и компьютерных наук

Программа подготовки бакалавров по направлению

09.03.04 Программная инженерия

Окунёков Николай Степанович

КУРСОВАЯ РАБОТА

Разработка системы управления моделью колёсно-гусеничного танка

Научный руководитель

Скулкин Сергей Павлович

Нижний Новгород, 2021

| | |
|--|----|
| Введение | 3 |
| Постановка задачи | 4 |
| Обзор существующих решений | 6 |
| Использованные инструменты | 8 |
| Архитектура приложения | 10 |
| Общее описание | 10 |
| Управление двигателями | 10 |
| Расчет оставшегося заряда аккумулятора | 13 |
| GPS и ГЛОНАСС | 14 |
| LTE-модем | 15 |
| Wi-Fi | 16 |
| Передача изображения с камеры | 17 |
| Фронтенд | 18 |
| Тестирование | 20 |
| Заключение | 21 |
| GitHub | 22 |
| Источники | 22 |

Введение

С началом пандемии COVID-19 и массовым переходом сотрудников предприятий на удаленную работу, у многих людей появилась потребность в возможности дистанционного мониторинга удаленных объектов. В качестве очевидного решения представляется использование камер видеонаблюдения и сигнализаций, но подобные устройства предполагают стационарную установку, что затрудняет обеспечение полного покрытия территории. Таким образом, появилась идея использования автономных устройств, способных самостоятельно передвигаться по территории и обеспечивать наилучшие возможности для мониторинга и удаленного оповещения. Наиболее удобным из подобных решений является модель колёсно-гусеничного танка, способная обеспечить хороший уровень проходимости.

Для управления такой моделью необходимо программное обеспечение, которое было бы способно обеспечивать удобное удаленное управление ею наибольшего количества устройств. Также, учитывая большой набор потенциальных конфигураций похожих устройств, а также сценариев их применения, важно, чтобы это ПО было открытым, универсальным, поддерживало наибольшее количество способов управления моделью и легко масштабировалось. Таким образом, было решено разработать и сконфигурировать собственный набор программ, а также создать тестовую модель для проведения его испытаний.

Постановка задачи

Передо мной стояла задача разработать программное обеспечение для управления моделью колёсно-гусеничного танка с любого современного устройства, на котором доступны веб-браузер и интернет, а также саму модель для демонстрации работы ПО. ПО должно являться набором различных программ, скриптов и интерфейсов, позволяющим реализовать удобный и доступный интерфейс как для управления моделью, так и для отслеживания показаний ее телеметрии.

При разработке я ориентировался на нижеследующие показатели:

- Портируемость. Возможность с минимальными изменениями переиспользовать ПО на других моделях. Например, на моделях автомобилей
- Удобство использования и понятность интерфейса, практичный дизайн
- Минимальные требования к устройству клиента, поддержка наибольшего возможного количества интерфейсов управления
- Низкое потребление ресурсов для возможности деплоя на малопроизводительные одноплатные системы
- Простота развертывания ПО

Основой тестовой модели является старая сломанная радиоуправляемая игрушка неизвестного китайского производителя, от которой, с доработками с моей стороны, было использовано шасси с моторами. Моторы через драйвер управления двигателями L298N подключены к одноплатному компьютеру Raspberry Pi посредством интерфейса GPIO. Второй такой же драйвер используется для управления вращением башни. Посредством GPIO к компьютеру также подключено реле, управляющее питанием фонаря и аналого-цифровой преобразователь, к которому, в свою очередь, подключен делитель напряжения. Курсовая камера (веб-камера Logitech), LTE-модем, модуль работы со спутниковыми системами GPS и ГЛОНАСС подключены к Raspberry Pi

посредством USB. Посредством mini-jack (3,5 мм) разъема подключен усилитель, обеспечивающий вывод звука на динамик, установленный в крышке шасси.

ПО должно обеспечивать корректную всех этих модулей, сбор телеметрии и информации о текущем состоянии устройства и предоставление их конечному пользователю посредством веб-интерфейса.

Обзор существующих решений

На данный момент существует подобные решения существуют либо в виде части детских игрушек (например, ПО для управления игрушкой со смартфона через Wi-Fi), либо в виде проектов компаний по курьерской доставке.

Так, например, похожей функциональностью обладает ПО, встроенное в игрушку Happy Cow I-Spy Mini FPV Tank. Однако, оно обладает рядом существенных недостатков:

- ПО абсолютно закрыто для модификаций и представляет из себя законченное решение
- Для работы с моделью клиенту необходима установка дополнительного программного обеспечения, доступного на ограниченном ряде платформ
- Поддержка LTE, GPS и ГЛОНАСС отсутствует
- Ограниченный набор телеметрической информации
- Управление моделью возможно исключительно посредством виртуальных джойстиков на сенсорном экране. Иные способы управления не поддерживаются
- Отсутствие портируемости. ПО работает исключительно на игрушке, для которой было разработано

Также подобное ПО существует как часть проекта курьерского ровера Phantom Auto. Однако, оно также обладает некоторыми недостатками, среди которых:

- Закрытость ПО
- Ограниченный набор доступной телеметрической информации

- Для работы клиенту необходим специальный терминал, разработанный для данного ровера
- Отсутствие портируемости. ПО работает исключительно на ровере, для которого было разработано

Таким образом, возникла идея разработки собственного открытого набора ПО.

Использованные инструменты

ПО представляет из себя бекенд, реализованный на языке Python, и фронтенд, реализованный на HTML и чистом JavaScript. Также в состав ПО входит несколько bash-скриптов для служебных целей, таких как запуск стриминга с курсовой камеры. При разработке были использованы следующие инструменты:

- Клиент OpenSSH для удаленного подключения к компьютеру, установленному внутри тестовой модели.
- Текстовый редактор vim для редактирования скриптов и конфигурационных файлов
- Текстовый редактор nano для быстрых изменений в скриптах
- Система контроля версий git для хранения кода и ведения контроля версий ПО
- GitHub — git-хостинг
- Браузеры Google Chrome и Safari для тестирования и отладки фронтенда приложения

А также следующие библиотеки:

- flask_socketio — реализация WebSocket на Python 3
- RPi.GPIO — реализация работы с GPIO Raspberry Pi на Python 3
- smbus — реализация работы с SMBus на Python 3
- huawei_lte_api — реализация работы с Huawei HiLink API на Python 3
- netifaces — реализация работы с сетевыми интерфейсами Unix-подобных операционных систем на Python 3
- gpsd — реализация взаимодействия с демоном gpsd (получения данных с модулей GPS и ГЛОНАСС) на Python 3

- wifi — враппер для iwlist и /etc/network/interfaces на Python 3
- Google Maps API — реализация работы с Google Maps на JavaScript и HTML
- Twitter Bootstrap — CSS-фреймворк

Архитектура приложения

Общее описание

В качестве протокола для связи бекенда и фронтенда мною был использован WebSocket. Компьютер модели выступает в роли WebSocket-сервера, обрабатывая запросы от клиентов и регулярно посыпая всем клиентам телеметрическую информацию. Управление двигателями модели реализовано через библиотеку RPi.GPIO, посредством которой организована подача напряжения на определенные пины и изменение их рабочего цикла (используется широтно-импульсная модуляция). Опрос I²C датчиков, подключенных через аналогово-цифровой преобразователь, происходит по шине SMBus через библиотеку smbus. На тестовой модели присутствует лишь один I²C датчик — датчик текущего напряжения на аккумуляторе. ПО также собирает информацию о уровне сигнала WiFi (посредством чтения /proc/net/wireless), количестве подключенных спутников GPS и ГЛОНАСС, текущих координатах устройства и его вертикальной скорости через gpsd (посредством одноименной библиотеки). Также реализован сбор информации об уровне и типе сигнала LTE-модема (посредством работы с Huawei HiLink API через библиотеку huawei_lte_api), при этом предусмотрена проверка на предмет отсутствия подключенного модема через библиотеку netifaces.

Управление двигателями

При получении команды от пользователя посредством WebSocket, приложение, в зависимости от типа используемого устройства ввода (клавиатура, виртуальный геймпад на сенсорном экране, либо реальный геймпад), выбирает функцию, в которой обрабатывается эта команда. В начале обработки команды на движение блокируется обработка обновления информации о заряде батареи в процентах, так как при движении модели

наблюдаются просадки напряжения, и используемый мною метод оставшегося заряда в процентах может давать неточные результаты. Затем, в случае, если используется геймпад, программа, в зависимости от типа полученной команды, обновляет значение рабочего цикла для GPIO-пина контроля скорости мотора для правой гусеницы, либо пина контроля скорости для мотора левой гусеницы на значения модуля координаты джойстика геймпада (имеется проверка на случай, если значение координаты больше 50 или меньше -50). Затем подает напряжение на pin движения вперед, либо на pin движения назад, отвечающий за работу левого, либо правого мотора. Если координата равна нулю, то напряжение на пины, отвечающие за мотор гусеницы, перестает подаваться.

```
1 global blockpercentage
2     blockpercentage = True
3     coord = int(message)
4     if coord>50:
5         coord=50
6     if coord<-50:
7         coord=-50
8     p1.ChangeDutyCycle(abs(coord))
9     if coord>0:
10        GPIO.output(in1,GPIO.HIGH)
11        GPIO.output(in2,GPIO.LOW)
12    elif coord<0:
13        GPIO.output(in1,GPIO.LOW)
14        GPIO.output(in2,GPIO.HIGH)
15    else:
16        GPIO.output(in1,GPIO.LOW)
17        GPIO.output(in2,GPIO.LOW)
18        sleep(5)
19    blockpercentage = False
```

В случае получения команды от клавиатуры, программа задает значение рабочего цикла на 50, а далее при нажатии кнопки W, для движения вперед, включает напряжение на обоих пинах движения вперед. В случае нажатия кнопки D, для вращения налево, напряжение на пины подается в такой конфигурации, что правая гусеница движется вперед, а левая — назад, при нажатии кнопки A — наоборот. При нажатии кнопки S программа включает напряжение на обоих пинах движения назад. При отпускании кнопки (команда «/») программа прекращает подачу напряжения на все пины.

```
1 p.ChangeDutyCycle(50)
2 p1.ChangeDutyCycle(50)
3 blockpercentage = True
4 if message == 'w':
5     GPIO.output(in1,GPIO.HIGH)
6     GPIO.output(in2,GPIO.LOW)
7     GPIO.output(in3,GPIO.HIGH)
8     GPIO.output(in4,GPIO.LOW)
9 elif message == '/':
10    GPIO.output(in1,GPIO.LOW)
11    GPIO.output(in2,GPIO.LOW)
12    GPIO.output(in3,GPIO.LOW)
13    GPIO.output(in4,GPIO.LOW)
14    sleep(5)
15    blockpercentage = False
16 elif message == 'a':
17    GPIO.output(in1,GPIO.HIGH)
18    GPIO.output(in2,GPIO.LOW)
19    GPIO.output(in3,GPIO.LOW)
20    GPIO.output(in4,GPIO.HIGH)
21 elif message == 'd':
22    GPIO.output(in1,GPIO.LOW)
23    GPIO.output(in2,GPIO.HIGH)
24    GPIO.output(in3,GPIO.HIGH)
25    GPIO.output(in4,GPIO.LOW)
26 if message == 's':
27    GPIO.output(in1,GPIO.LOW)
28    GPIO.output(in2,GPIO.HIGH)
29    GPIO.output(in3,GPIO.LOW)
30    GPIO.output(in4,GPIO.HIGH)
```

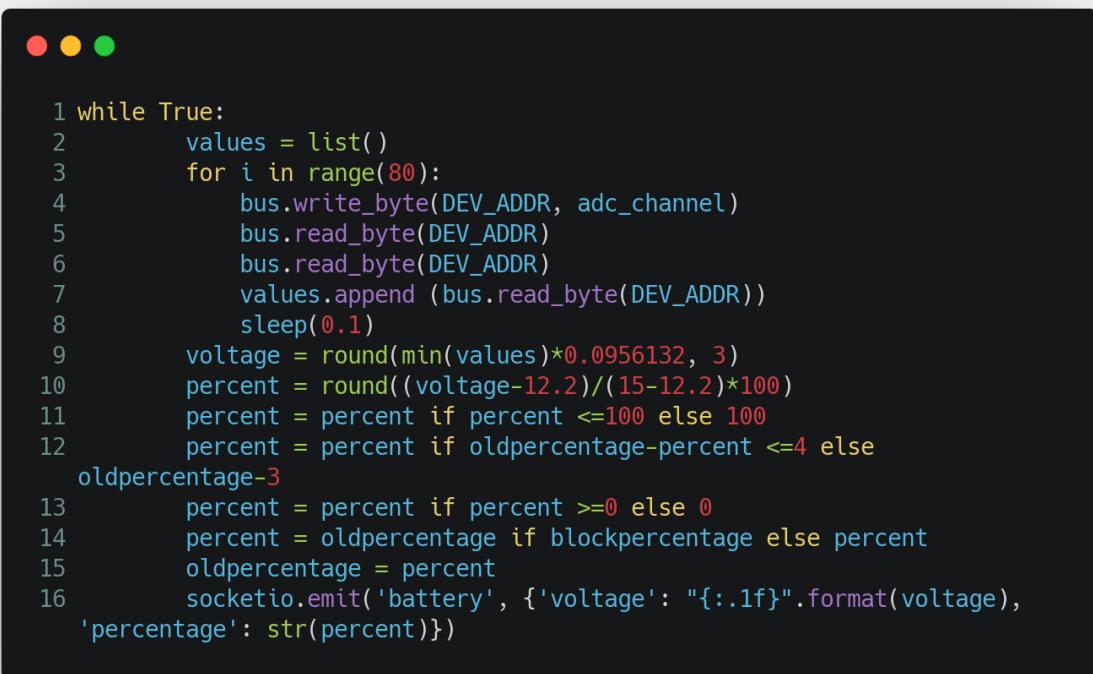
Расчет оставшегося заряда аккумулятора

Для предоставления пользователю информации об оставшемся заряде аккумулятора реализована обработка информации I²C-датчика напряжения. В результате опроса датчика программа получает значение от 0 до 255. Чтобы выяснить корреляцию получаемых значений с реальной информацией о напряжении аккумулятора, опытным путем был выяснен коэффициент для выполнения конвертации. Я тестировал датчик напряжения с батарейками формата АА, сопоставляя получаемые значения с показаниями мультиметра. Получив данные с одной, двух и трех последовательно подключенных батареек, я решил три уравнения вида $X * \text{(значение, полученное с датчика)} = \text{(напряжение, полученное с мультиметра)}$. Далее взял среднее значение X и использовал его в своей реализации.

Для расчета заряда аккумулятора в процентах я выяснил напряжение на полностью разряженном аккумуляторе (значение, при котором контроллер аккумулятора отключает систему в целях защиты от переразряда) и на полностью заряженном. Расчет процентов выполняется по формуле $((\text{напряжение аккумулятора}) - (\text{напряжение разряженного аккумулятора})) / ((\text{напряжение заряженного аккумулятора}) - (\text{напряжение разряженного аккумулятора})) * 100$.

Однако, при использовании этого метода может возникнуть проблема с тем, что при повышенной нагрузке на аккумуляторы (например, при работе двигателей) значение мгновенного напряжения на них проседает, а соответственно, формула расчета заряда может давать неверные результаты. Для защиты от этого мною была реализована блокировка обновления значения текущего заряда аккумулятора в процентах при условии работы двигателей. Разумеется, данный метод является несовершенным и не дает достаточной точности. Эту проблему можно решить добавлением в конфигурацию устройства дополнительных датчиков, таких как датчик тока, а

также датчиков, позволяющих определить текущую емкость аккумуляторов химическим методом. К сожалению, данные датчики отсутствуют в собранной мною тестовой модели, однако присутствует возможность для их добавления в будущие конфигурации: в систему легко интегрируются любые I²C-датчики.



```
1 while True:
2     values = list()
3     for i in range(80):
4         bus.write_byte(DEV_ADDR, adc_channel)
5         bus.read_byte(DEV_ADDR)
6         bus.read_byte(DEV_ADDR)
7         values.append (bus.read_byte(DEV_ADDR))
8         sleep(0.1)
9     voltage = round(min(values)*0.0956132, 3)
10    percent = round((voltage-12.2)/(15-12.2)*100)
11    percent = percent if percent <=100 else 100
12    percent = percent if oldpercentage-percent <=4 else
13        oldpercentage-3
14    percent = percent if percent >=0 else 0
15    percent = oldpercentage if blockpercentage else percent
16    oldpercentage = percent
17    socketio.emit('battery', {'voltage': "{:.1f}".format(voltage),
18 'percentage': str(percent)})
```

GPS и ГЛОНАСС

Также в интерфейсе присутствует отображение показателей текущей скорости движения модели, количества активных спутников GPS и ГЛОНАСС и положения модели на карте Google. Для получения этой информации мною была использована библиотека gpsd, реализующая интерфейс для работы с одноименным демоном операционной системы Linux. Библиотекой представляются данные о количестве спутников, вертикальной скорости, а также широте и долготе, на которых находится устройство в текущий момент. На фронтенде данные о скорости и количестве спутников выводятся на экран, а координаты, при помощи Google Maps API, отображаются на карте в отдельном

окне, открывающемся по нажатии соответствующей кнопки, которая активна только при уверенном приеме GPS.

GPS на тестовой модели реализован с помощью USB-модуля производства u-blox.



```
1 while True:
2     try:
3         packet = gpsd.get_current()
4         if packet.mode >= 2:
5             socketio.emit('gps', {'satellites': str(packet.sats),
6 'latitude': str(packet.lat), 'longitude': str(packet.lon), 'speed':
7 str(packet.hspeed)})
8         else:
9             socketio.emit('gps', {'satellites': str(packet.sats),
10 'latitude': 'N/A', 'longitude': 'N/A', 'speed': 'N/A'})
11     except:
12         socketio.emit('gps', {'satellites': '0'})
13     sleep(1)
```

LTE-модем

Для обеспечения удаленного управления вне зоны покрытия сетей Wi-Fi, тестовая модель оснащена LTE-модемом, подключенным по USB. Для тестирования используется модем производства Huawei с фирменной прошивкой HiLink. Для модемов на этой прошивке на языке Python реализована библиотека `huawei_lte_api`, обеспечивающая работу с их API. Однако получение данных с ее помощью затруднено некорректной работой при отсутствии подключенного модема — библиотека не содержит реализации исключений для этого случая и попросту зависает. Чтобы этого избежать, мною была реализована проверка работы интерфейса модема непосредственно перед обращением к API. За проверку отвечает библиотека `netifaces`, реализующая работу с сетевыми интерфейсами операционной системы Linux. Таким образом,

при условии, что модем подключен, данные об уровне сигнала и типе сети из библиотеки huawei_lte_api передаются на фронтенд.

Wi-Fi

Отображение текущего уровня сигнала Wi-Fi сети реализовано при помощи чтения файла /proc/net/wireless Linux-утилитой awk. Получаемое из файла значение умножается на 10 и делится на 7, в следствии чего вычисляется уровень сигнала в процентах. Также при помощи библиотеки wifi при получении команды scan от клиента реализовано получение списка доступных Wi-Fi сетей. Команда scan отправляется при нажатии на иконку Wi-Fi. При нажатии на эту же иконку открывается таблица с полученным списком Wi-Fi сетей с возможностью подключиться к ним. Со стороны бекенда эта возможность также реализована с помощью библиотеки wifi.

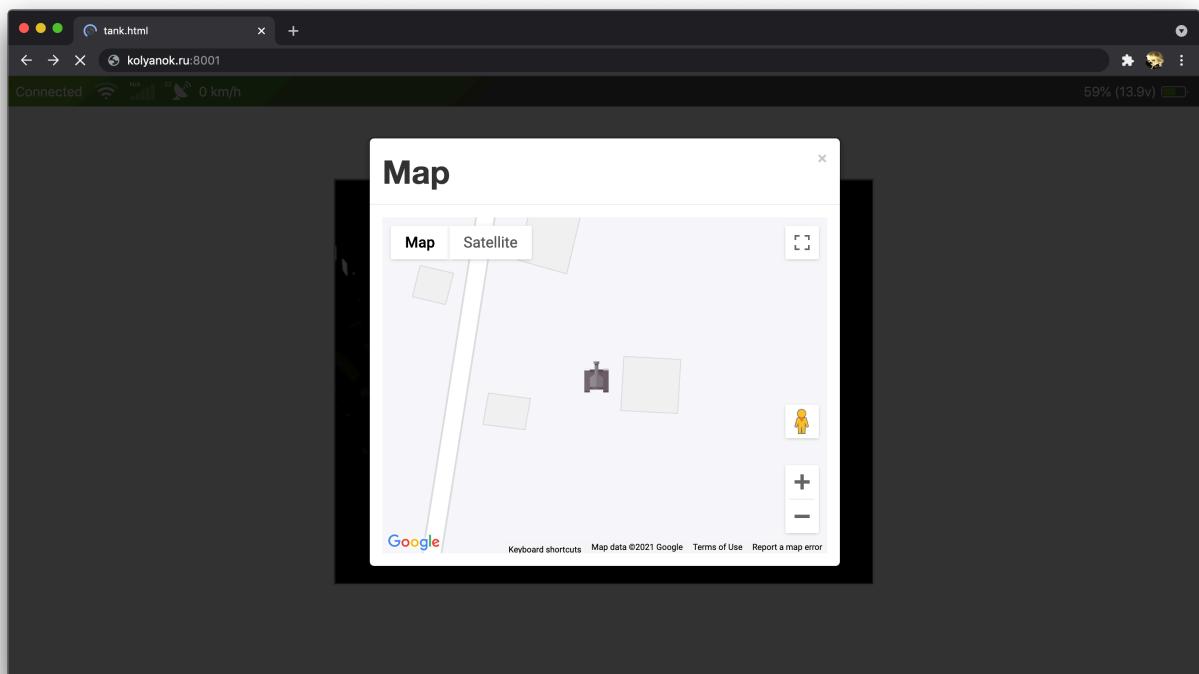
```
1 while True:
2     wifisignal = os.popen("awk 'NR==3 {printf($3*10/7)}' /proc/net/wireless").read()
3     if (wifisignal):
4         socketio.emit('wifi', {'signal': wifisignal+"%"})
5     else:
6         socketio.emit('wifi', {'signal': "0%"})
7     sleep(1)
```

```
1 if message == 'scan':
2     networks = list()
3     [networks.append({'ssid':i.ssid.encode("latin1").decode("unicode-escape").encode("latin1").decode('utf-8'), 'signal':i.signal, 'encrypted':i.encrypted}) for i in list(wsd.Cell.all('wlan0'))]
4     socketio.emit('scan', networks)
```

Передача изображения с камеры

Задача выбора протокола и формата передачи потокового видео с курсовой камеры модели была довольно непростой задачей. Необходимо было найти решение, сочетающее в себе низкое потребление ресурсов, низкую задержку и обеспечение минимальной нагрузки на канал связи. В качестве курсовой камеры тестовой модели используется USB-вебкамера Logitech, которая отдает изображение в формате Motion JPEG. Изначально мне хотелось реализовать транскодинг этого формата в кодек H.264 или H.265, однако в ходе тестирования выяснилось, что транскодинг потребляет слишком много ресурсов и обеспечивает неприемлемое время задержки. Исходя из этого было решено передавать Motion JPEG в браузер напрямую, без транскодинга. Такое решение, на мой взгляд, является оптимальным для передачи потокового камеры, отдающих изображение в формате Motion JPEG, так как этот формат поддерживается большинством браузеров, а отсутствие промежуточных транскодирующих решений позволяет снизить задержки до минимума.

Однако, в случае наличия у меня камеры, способной передавать видео сразу с использованием кодека H.264, существовала бы возможность



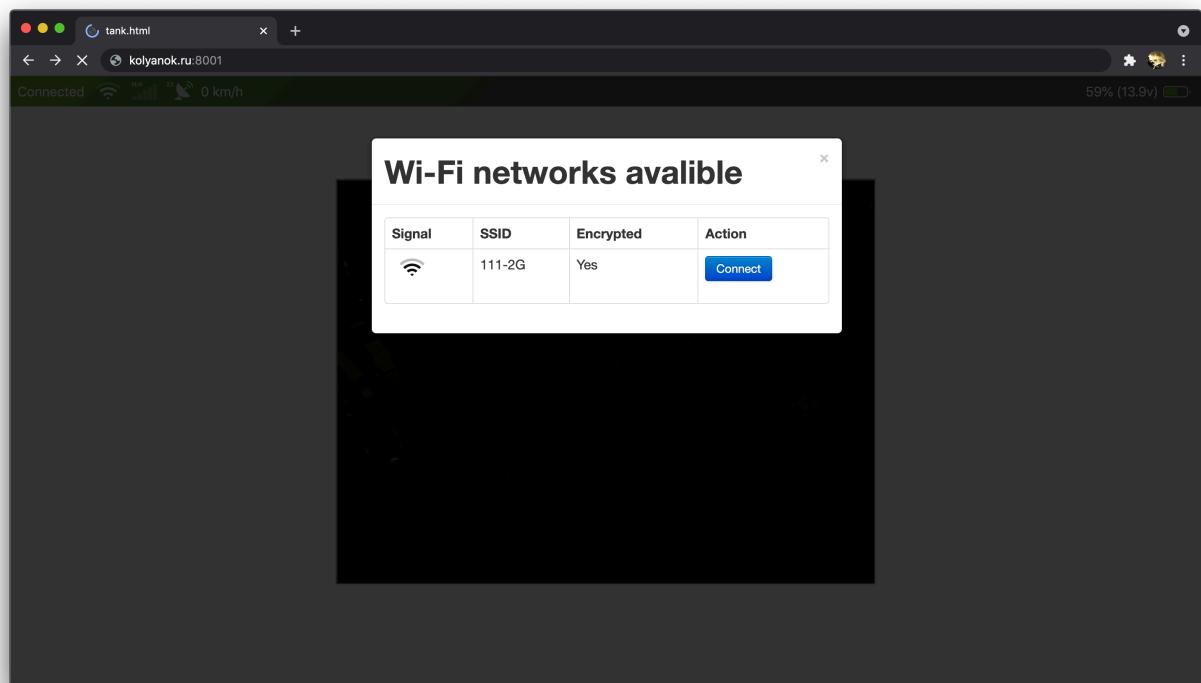
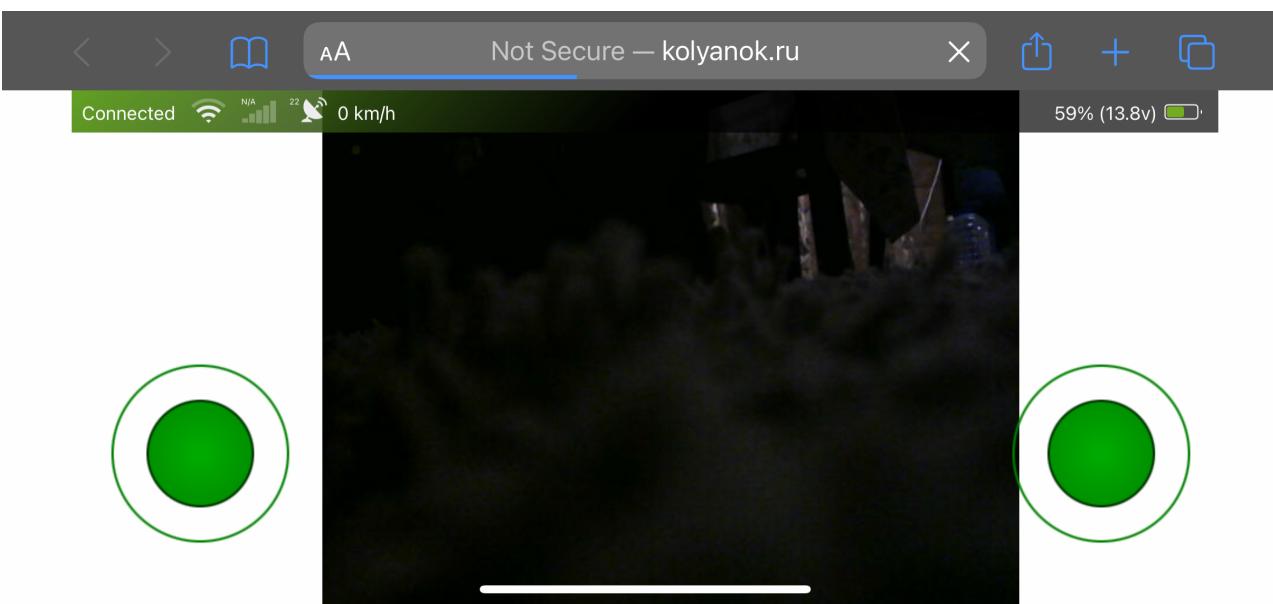
```
1 #!/bin/sh
2
3 DEVICE=/dev/video0
4 RESOLUTION=640x480
5 FRAMERATE=25
6 HTTP_PORT=8001
7 LOGINPASS="just4:fun"
8
9 PLUGINPATH=/usr/local/lib/mjpg-streamer
10
11 mjpg_streamer -i "$PLUGINPATH/input_uvc.so -n -d $DEVICE -r $RESOLUTION
-f $FRAMERATE" -o "$PLUGINPATH/output_http.so -n -p $HTTP_PORT -w
/home/pi/www -c $LOGINPASS"
12
```

использования технологии WebRTC, что, в свою очередь, обеспечило бы гораздо меньшую нагрузку на канал связи, в сравнении с Motion JPEG.

Для непосредственно трансляции видеоизображения в браузер мною была использована программа mjpg-streamer, также содержащая в себе веб-сервер, на котором я разместил HTML, CSS и JS файлы фронтенда. Программа также имеет возможность защиты доступа к веб-сервера паролем, которая также была мною активирована. Для запуска программы с соответствующими параметрами я написал bash-скрипт.

Фронтенд

Дизайн фронтенда был сделан мною с нуля. Для этого я использовал возможности HTML5 и CSS3. Логика была написана на чистом JavaScript без использования фреймворков. Простые иконки, в частности иконка уровня сигнала сети, иконка уровня заряда аккумулятора и иконка уровня сигнала Wi-Fi, были нарисованы с помощью CSS, без использования изображений. Для реализации виртуальных джойстиков, обеспечивающих управление с сенсорных экранов, был использован скрипт joy.min.js разработчика [bobboteck](#). Джойстики появляются только при условии наличия в устройстве сенсорного



экрана. Реализация поддержки геймпадов реализована на встроенном Gamepad API языка JavaScript и активируется при подключении совместимого с операционной системой устройства геймпада любым доступным методом (с использованием беспроводного, либо проводного подключения). Работа с WebSocket реализована средствами библиотеки Socket.IO.

Дизайн является полностью адаптивным, из-за чего работа с системой возможна как с использованием компьютера, так и с использованием смартфона и планшета.

Тестирование

Тестирование работы проводилось на созданной мной при помощи моих друзей тестовой модели колёсно-гусеничного танка. Для тестирования использовалась операционная система Raspberry Pi OS, установленная на одноплатный компьютер Raspberry Pi 3.

Работа фронтенда была проверена на смартфонах под управлением iOS и Android, компьютерах под управлением macOS и Windows, браузерах Google Chrome и Safari. Работа управления с геймпада была протестирована на геймпаде Sony DualShock 4.

Было выполнено тестирование на предмет задержки видеосигнала и сигнала управления: модель, находясь в Нижнем Новгороде на Большой Покровской улице и будучи подключенной к интернету через LTE-сеть оператора Мегафон, управлялась людьми из США и Узбекистана. При этом уровень задержки и качество связи позволяли участникам тестирования управлять моделью без каких-либо проблем.

Из-за отсутствия у операторов сотовой связи услуг по предоставлению частным абонентам статического IP-адреса, мною было использовано OpenVPN подключение для возможности доступа к устройству через интернет, но даже в случае использования OpenVPN задержка оставалась на приемлемом уровне.

Также я протестировал показания модуля GPS и Глонасс, отображаемые уровни сигналов беспроводных подключений и заряда аккумулятора — данные соответствовали действительности.

При этом ПО не давало большой нагрузки на процессор, что обеспечивало низкое энергопотребление и запас хода тестовой модели на одном заряде аккумуляторов около 1,5 км.

Заключение

Я остался доволен своей работой и в полной мере реализовал все, что задумал. Показатели скорости и стабильности работы реализованного мною программного обеспечения меня полностью устраивают.

В процессе работы я получил серьезный опыт взаимодействия с радиоэлектроникой, освоил технологию WebSocket, хорошо попрактиковался в дизайне интерфейсов и администрировании операционной системы Linux. Я применил знания, полученные на таких дисциплинах как «Операционные системы» и «Проектирование интерфейсов (UI/UX)».

В будущем я планирую добавить совместимость с большим количеством датчиков, другими камерами, а также реализовать, в случае отсутствия возможности подключиться к интернету, создание устройством Ad-Hoc Wi-Fi сети для уменьшения зависимости устройства от покрытия Wi-Fi, LTE и 3G сетей. Также я рассматриваю возможность портирования данного ПО не только на модели колёсно-гусеничных танков, но и на другие устройства, в том числе роверы для доставки и летающие беспилотники.

GitHub

<https://github.com/SharagaFun/tank>

Источники

<https://flask-socketio.readthedocs.io/en/latest/> — документация к Flask SocketIO (по состоянию на 13 июня).

<https://www.raspberrypi.org/documentation/usage/gpio/> — документация к Raspberry Pi GPIO (по состоянию на 13 июня).

<https://ph0en1x.net/94-adc-dac-pcf8591-raspberry-pi-tutorial-i2c-bus-python.html> — Знакомство с шиной I2C в Raspberry Pi, работаем с ADC-DAC PCF8591 на Python (по состоянию на 13 июня).

<https://pypi.org/project/huawei-lte-api/> — документация к huawei-lte-api (по состоянию на 13 июня).

<https://gpsd.gitlab.io/gpsd/#documentation> — документация к gpsd (по состоянию на 13 июня).

<https://wifi.readthedocs.org/en/latest/> — документация к wifi (по состоянию на 13 июня).