4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Аппаратная часть

4.2 Программная часть

Программное средство для управления подвижной системой-комплексом представляет из себя приложение, разработанное на языке Python. Данный язык был выбран в качестве основного, так как представляется несложным в понимании и предоставляет большое количество возможностей для гибкой разработки приложений любых форматов. Данный язык программирования поддерживает несколько парадигм программирования, в том числе объектно-ориентированное, функциональное, императивное и аспекто-ориентированное.

Также одним из аспектов в выборе Python в качестве основного языка для программного средства послужила динамическая типизация, автоматическое управление памятью и удобный механизм обработки исключений.

Пространство имен и модульность повлияли на условие расширяемости программного средства. С этими двумя факторами Python отлично справляется. В ходе разработки проекта это очень помогло.

Начинается работа программы с того, что пользователь должен ввести IP-адрес, к которому должна подключится программа, чтобы наладить контакт с устройством. Получение IP-адреса, к которому нужно подключаться, описано в предыдущем пункте.

Ниже приведен код для ввода и проверки IP-адреса:

REGULAR\_IP = '^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$'

regex = re.compile(REGULAR\_IP)

while (is\_valid\_address == False):

addr = input("\nInput host addres: ")

if (regex.match(addr)):

is\_valid\_address = True

HOST = addr

Как можно заметить REGULAR\_IP является регулярным выражением для проверки валидности введенного IP-адреса.

Программа будет предлагать ввести IP-адрес до тех пор, пока он не пройдет основную валидацию.

Далее приведена настройка клиентского сокета:

*wifi\_module = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)*

*wifi\_module.connect((HOST, PORT))*

После подключения к wi-fi модулю пользователю предлагается ввести команду. За данную функциональность отвечает данный блок кода:

while True:

try:

cmd = input("Type command: ")

if validate\_command(cmd):

execute(cmd)

В данном блоке происходит сначала проверка валидности команды. Проверка представляет собой соответствие введенной команды на адекватность и наличие данной команды в списке команд.

После прохождения проверки введенная команда готова к исполнению.

В методе execute происходит выполнение, а именно:

def execute(cmd):

if is\_menu\_cmd(cmd):

menu.run(cmd)

else:

communication.send\_data(cmd)

communication.print\_data()

Сначала нужно определить является ли введенная команда командой для меню программы. В случае, если команда ориентирована на взаимодействие с wi-fi модулем, то она отправляется в блок else, в котором команда с помощью модуля Communication отправляется модулю обработки данных, а далее беспроводному модулю.

Далее ожидается ответ. Ответ может быть одним из 3-х вариантов:

– Null – в случае отсутствие данных;

– Unknown command – в случае, когда была введена неизвестная команда для контроллера;

– Любые, адекватные данные, пришедшие со стороны контроллера.

Рассмотрим форматирование и конвертацию данных при получении и отправке. Для этого рассмотрим методы модуля Communication:

– get\_data;

– send\_data.

def get\_data():

return wifi\_module.recv(BUFFER\_SIZE).decode('utf-8')

BUFFER\_SIZE по умолчанию равен 1024.

В данном методе происходит прослушивание сокета и ожидание, когда будут посланы данные с другой стороны – со стороны wi-fi модуля.

Конвертация происходит из utf-8 в обычную кодировку.

def send\_data(data):

wifi\_module.send(str(data+STOP\_SYMBOL).encode('utf-8'))

В данном методе происходит отправка данных wi-fi модулю. Данные кодируются в utf-8 – для более компактного хранения и передачи. Как можно заметить к данным конкатенируется дополнительный стоп-символ. Это сделано для того, чтобы контроллер мог распознать закончена команда или нет. В качестве стоп-символа можно использовать любой символ, только следует заранее указать его в контроллере.

Использование обработчика ошибок и исключений является обязательным в разработке приложений с межсетевым взаимодействием.

В модуле Error предусмотрены обработки таких ошибок как:

– ошибка сети, сокета;

– прерывание клавиатуры.

except KeyboardInterrupt:

print("KeyboardInterrupt was handled")

wifi\_module.close()

os.\_exit(1)

except socket.error as e:

print("WiFi session was stoped")

wifi\_module.close()

os.\_exit(1)