УДК 629.58+004.94+004.415.53
DOI: 10.37102/1992-4429_2025_52_02_06
EDN: SXWLAS

# ОПЫТ ПРИМЕНЕНИЯ ЯЗЫКА PYTHON ДЛЯ СОЗДАНИЯ СИСТЕМЫ УПРАВЛЕНИЯ НЕОБИТАЕМОГО ПОДВОДНОГО АППАРАТА

**Г.Д. Елисеенко, А.М. Павин, А.С. Торжков, К.Д. Шилин**

Создание программного обеспечения для систем управления подводными робототехническими комплексами сопряжено как с адаптацией апробированных технических решений, так и с разработкой новых. При этом в процесс могут вовлекаться различные средства разработки программ, спектр которых в настоящее время весьма широк. В статье рассматривается опыт использования языка *Python* при создании «с нуля» систем управления для необитаемых подводных аппаратов (как автономных, так и телеуправляемых) на примере разработки малогабаритного аппарата *Geek*. Особенность данного аппарата заключается в его назначении – обучение студентов и аспирантов основам проектирования подводной необитаемой техники, а также участие студенческих команд в соревнованиях по подводной робототехнике. Рассматриваются различные аспекты использования языка *Python*, включая межпроцессное взаимодействие, обработку данных с датчиков и систем подводного робота, автономное управление, взаимодействие с операторами на поверхности, а также интеграцию с другими технологиями, необходимыми для успешного выполнения аппаратом миссий под водой. Обсуждаются результаты применения подводного робота *Geek* для выполнения различных операций в телеуправляемом и автономном режимах.

**Ключевые слова:** система управления, необитаемый подводный аппарат, программное обеспечение подводных роботов, межпроцессное взаимодействие, язык *Python*, НПА, АНПА, ТНПА.

## Введение

Необитаемые подводные аппараты (НПА) как телеуправляемого, так и автономного класса играют ключевую роль в исследовании и освоении глубин Мирового океана. Создание программного обеспечения (ПО) подводных аппаратов-роботов требует применения современных технологий и программных решений. В свою очередь, системы управления НПА имеют особенности, отличающие их от соответствующих систем наземных и воздушных мобильных роботов. Данные различия обусловлены нестабильной и медленной связью подводного аппарата с оператором или ее полным отсутствием (в случае автономных подводных аппаратов), менее точной и/или редко обновляемой навигацией, ограниченным числом датчиков внешней среды, а также необходимостью оперативного принятия решений в различных нештатных ситуациях, происходящих в процессе выполнения поставленных задач.

Основным назначением системы управления (СУ) НПА является осуществление навигации и управления аппаратом согласно заложенной модели использования. Исходя из назначения системой управления решаются следующие основные задачи:

- определение пространственного и углового местоположения робота в трехмерном пространстве с учетом данных навигационно-пилотажных датчиков (датчики линейных и угловых положений и скоростей, системы акустического позиционирования под водой и др.);
- управление движением робота во всём доступном диапазоне глубин, направлений и скоростей с учетом ограничений энергетической системы и движительно-рулевого комплекса подводного робота;
- обработка данных от бортовых систем сбора информации для решения обзорно-поисковых и обследовательских задач с применением фото- и гидролокационных систем освещения подводной обстановки;
- выполнение манипуляционных операций с применением комплексов манипуляционных

устройств (применительно к телеуправляемым аппаратам, а в последнее время актуально и для автономных роботов).

К системе управления НПА, средствам её разработки и поддержки предъявляется ряд требований. Перечислим наиболее важные из них.

*Минимизация бортовых вычислительных ресурсов*. Определяется необходимостью рассеяния тепла в замкнутом объеме прочных контейнеров, косвенной экономией электроэнергии за счет выбора меньших габаритов прочных контейнеров, а также прямой экономией электроэнергии за счет меньшего потребления системой управления.

*Модифицируемость и наращиваемость*. СУ робота должна легко наращиваться и модифицироваться по мере изменения состава сенсоров и бортовых устройств или появления новых задач. Во многих случаях (например, во время интенсивной работы в море) также актуальна возможность быстрых изменении в логике работы различных алгоритмов управления НПА в условиях ограниченного времени.

*Простота реализации*. СУ должна допускать реализацию с использованием существующих и широко применяемых средств программирования на доступных вычислительных ресурсах.

*Широта использования*. СУ должна быть применима для подводных роботов различного класса и назначения, в частности для автономных (АНПА) и телеуправляемых (ТНПА).

Используемые в настоящее время программные средства создания СУ в различной степени удовлетворяют перечисленным выше требованиям. Целью приведенных ниже исследований является определение места языка *Python* в ряду этих средств, а также оценка характеристик СУ на базе *Python* с точки зрения удобства использования, эффективности и надежности функционирования на этапах разработки и применения НПА.

Статья организована следующим образом. Первый раздел посвящен обзору и анализу применяемых в мировой практике программных средств с использованием приведённых выше критериев. Во втором разделе рассмотрены аппаратные и программные особенности НПА *Geek,* для которого разрабатывается СУ. В третьем разделе внимание уделяется различным вопросам применения языка *Python* для разработки программного обеспечения бортовой системы управления (автопилота) и берегового поста управления НПА *Geek*. Наконец, в четвёртом разделе рассматриваются различные аспекты практического использования разработанной СУ и обсуждаются полученные результаты.

## Средства разработки управляющего ПО необитаемых подводных аппаратов

На сегодняшний день для разработки сложных систем управления используются специализированные среды разработки, известные как робототехнические программные платформы. Данные платформы предоставляют разработчикам инструменты для описания, создания, настройки и выполнения программного обеспечения систем управления. Также они обеспечивают передачу данных и команд между различными компонентами системы. Существует широкий выбор специализированных робототехнических программных платформ [1–3]. Наиболее популярной в области разработки систем управления роботами (не только подводными) является *Robot Operating System (ROS)* [4]. Данная платформа широко используется в разработке промышленных роботов и их прототипов и предлагает разработчикам разнообразные инструменты и библиотеки для создания и управления робототехническими системами. На международных соревнованиях по робототехнике в классе АНПА подавляющее большинство команд для реализации системы управления используют *ROS*. Примерами могут служить российские [5, 17] и зарубежные команды-лидеры в данной области *Duke Robotics Club* [6], *Bumblebee AUV* [7].

Из преимуществ *ROS* можно выделить: хорошую документированность; работу под операционной системой *Linux* и *Mac OS X*; наличие большого числа готовых модулей для поддержки устройств и интерфейсов (джойстики, камеры, протокол *i2c* и др.); возможность использования языка *Python* при создании модулей, что также повышает скорость разработки. Однако, как и у любой другой технологии, у *ROS* есть некоторые ограничения, например:

1. Сложность установки и настройки. Начальная настройка *ROS* может быть сложной для новичков. Большое количество зависимостей, пакетов и компонент системы вызывает затруднения у пользователей, особенно у тех, кто не имеет опыта работы с подобными системами.

2. Избыточность универсальной системы и как следствие – повышенные требования к объему оперативной памяти, частоте процессора и свободного пространства на жестком диске бортового вычислителя.

3. Зависимость от стандартизованных форматов сообщений, на которые подписаны многие утилиты *ROS*. Примером может служить «универсальное» сообщение координат робота, которое состоит из трех поступательных и трех вращательных координат. В

то время как в подводной робототехнике часто используются различные системы координат для горизонтального позиционирования, например, счисленные координаты (измеряемые в метрах), глобальные координаты от гидроакустических и спутниковых навигационных систем (например, в системе *WGS-84*), коплексированные навигационные координаты и др.

4. Сложность проведения сертификации на недокументированные возможности из-за чрезмерного объема исходного кода и отсутствие гарантий и официальной поддержки производителя.

5. Отсутствие поддержки операционной системы *Windows* и некоторых других.

Из наиболее свежих разработок (2019 год) набирает популярность фреймворк *Isaac SDK,* представленный компанией *NVIDIA* для разработки систем управления роботами, ориентированный на машинное обучение [8]. Фреймворк был разработан специально для встраиваемой платформы *Jetson* и не может быть использован на других платформах. *Isaac SDK* является аналогом *ROS*, однако обладает существенными отличиями в реализации и архитектуре. Архитектура системы управления в *Isaac SDK* представляет собой граф узлов (*nodes*) и их соединений (*channels*), как и в *ROS*. Однако узлы не являются отдельными независимыми процессами, вместо этого весь граф собирается в одно приложение. Данный подход увеличивает скорость обмена между узлами, однако требует дополнительных усилий разработчиков для того, чтобы избежать «падения» всей системы при возникновении ошибок в одном из узлов. *SDK* включает в себя поддержку технологий *CUDA, TensorRT, OpenCV* и др. В дополнение к *Isaac SDK* поставляется ПО *IsaacSim*, которое позволяет использовать симуляторы для роботов. Поддерживаются движки *Unity* и *NVIDIA – Omniverse IsaacSim*. Примеры реальных разработок представлены только в виде наземных экземпляров.

Кроме широко известных платформ существуют и специализированные программные платформы для морских роботизированных комплексов, например программная платформа *IPC* [12–14], разработанная в Институте проблем морских технологий им. академика М.Д. Агеева Дальневосточного отделения Российской академии наук (ИПМТ ДВО РАН). На основе этой платформы функционируют большинство АНПА, разработанных в ИПМТ. Платформа имеет следующие преимущества: а) децентрализованная распределенная архитектура, в которой отсутствуют выделенные модули для обмена сообщениями; б) единый механизм информационного взаимодействия между программными компонентами одного компьютера или нескольких компьютеров НПА на основе *UDP*-сообщений (датаграмм); в) в информационных сообщениях программной платформы отсутствует адресат–потребитель данных, что обеспечивает максимальную гибкость создаваемого ПО; г) поддержка операционных систем *Linux* (в том числе *Astra*, *Debian*, *Ubuntu*), *QNX*, *MacOS*, *Windows* и *Android*. К минусам платформы можно отнести: язык реализации программной платформы только *C++*, что ограничивает использование готовых решений на других языках, и отсутствие на настоящий момент полноценной документации.

Студенческие соревнования по подводной робототехнике являются удобной и стимулирующей средой для обкатки новых решений и алгоритмов управления перед их возможным последующим внедрением в промышленные НПА. Пять лет назад в ИПМТ ДВО РАН совместно с Дальневосточным федеральным университетом (ДВФУ) был создан АНПА *Pandora* [5] (предшественник НПА *Geek*), специально предназначенный для участия в соревнованиях по подводной робототехнике и обучению студентов. В роботе применялась программная платформа *IPC* в качестве базы для реализации системы управления. Однако отсутствие полноценной документации, большого открытого сообщества данной платформы и невозможность использования других языков программирования кроме *C++* значительно усложняют разработку программного обеспечения студентами.

В последнее десятилетие высокоуровневый язык программирования *Python* приобрел большую популярность в сфере разработки прототипов систем управления для подводных робототехнических комплексов [15–16]. Его привлекательность заключается в ряде фундаментальных характеристик, которые делают его подходящим инструментом для создания роботов и автоматизированных устройств: а) *Python* обладает простым и понятным синтаксисом, что делает его доступным для начинающих разработчиков; б) поддержка разнообразных структур данных и парадигм программирования; в) создание рабочих прототипов систем и тестирование их концепции до перехода к полноценной разработке; г) *Python* располагает множеством библиотек и фреймворков, которые подходят для робототехнических проектов (например, *TensorFlow* и *PyTorch* для работы с алгоритмами машинного обучения). Среди недостатков языка *Python* можно выделить: а) отсутствие статической проверки типов на этапе компиляции увеличивает вероятность ошибок; б) сложности в обнаружении ошибок (логические ошибки в ветках кода выявляют-

ся только во время выполнения, что делает их выявление особенно трудоемким); в) неявное поведение, возникающее из-за неправильного использования возможностей динамического переопределения переменных во время выполнения программы; г) медленное выполнение циклов, особенно вложенных, что делает трудоемким процесс создания алгоритмов реального времени без применения специальных оптимизирующих процедур (таких как векторизация).

## Необитаемый подводный аппарат *Geek*

Созданный несколько лет назад необитаемый подводный аппарат *Geek* предназначен для обучения студентов основам конструирования и программирования автономных роботов, а также для участия в международных и региональных соревнованиях. Разработка этого аппарата велась совместными усилиями студентов ДВФУ различных направлений (конструкторы, электронщики и программисты) под руководством наставников из ИПМТ ДВО РАН. Внешний вид робота представлен на рис. 1.

*Рис. 1.* Внешний вид необитаемого подводного аппарата *Geek*

В состав штатных систем НПА *Geek* входят: система бортового управления (автопилот), система энергообеспечения, движительно-рулевой комплекс и система технического зрения. К особенностям конструктивного исполнения аппарата можно отнести (основные характеристики НПА *Geek* приведены в таблице):

- малое лобовое сопротивление для достижения высоких показателей скорости в продольном направлении;
- высокая маневренность по всем шести степеням свободы (движительно-рулевой комплекс аппарата позволяет выполнять фигуры высшего пилотажа, такие как «бочка» и «мертвая петля»);
- гибридная схема энергообеспечения и связи, позволяющая выполнять задачи как автономно, так и в режиме телеуправления;
- относительно малые габариты и вес аппарата в сравнении с роботами других команд, выступающих на международных соревнованиях в классе АНПА;
- относительно простая реализация и поддержание рабочего состояния за счет применения доступных на рынке компонент и материалов.

**Основные характеристики НПА *Geek***

| № | Параметр | Значение | Единица измерения |
|---|---|---|---|
| 1 | Максимальная рабочая глубина погружения | 30 | м |
| 2 | Масса в воздухе | 12 | кг |
| 3 | Габариты | 0.5×0.4×0.25 | м |
| 4 | Скорость движения относительно воды в продольном направлении | до 1.5 | м/с |
| 5 | Время автономной работы | до 3 | час |
| 6 | Время заряда аккумуляторной батареи | не более 1 | час |

В состав системы энергообеспечения подводного робота входит батарейный модуль, состоящий из литий-ионных аккумуляторов 18650, способных работать с высокими токами для обеспечения кратковременных режимов быстрого перемещения. Движительно-рулевой комплекс подводного аппарата состоит из четырех малых подруливающих устройств (*Underwater Brushless DC Motor Thruster* производства компании *RovMaker*) и двух основных маршевых движителей (*T200 Thruster* производства *BlueRobotics*). Управление двигателями осуществляется с использованием платы *ESC Holybro Kotleta* 20 с интерфейсом *CAN*. Система технического зрения (СТЗ) состоит из двух стереокамер производства компании *StereoLab*, направленных вперед и вниз аппарата. Кроме того, в состав СТЗ входит гидроакустический приемник, позволяющий определять дистанцию и направление до источника акустического сигнала в диапазоне частот от 10 до 50 кГц. Сенсорные и исполнительные устройства аппарата представлены следующими комплектующими: датчик глубины *MS5837* производства компании *RovMaker* с интерфейсом *i2c*; датчик угловых скоростей ВГ-103ПТ производства российской компании

«Физоптика» с интерфейсом *UART*; плата силовых ключей, управляемая с помощью интерфейса *GPIO*, для коммутирования забортной полезной нагрузки (устройства сброса шаров и др.). В качестве основного вычислителя на борту используется одноплатный компьютер *Nvidia Jetson TX2*, который обладает следующими характеристиками: четырехядерный процессор *Arm Cortex-A57 MPCore*; оперативная память 8 ГБ *LPDDR4*; графический процессор с 256 ядрами *NVIDIA CUDA*.

Состав программной части системы управления представлен на рис. 2. Данная система программного управления представлена в классической трехуровневой архитектуре, принятой в ИПМТ ДВО РАН, и содержит следующие уровни: а) «исполнительный» уровень (желтые модули на схеме) объединяет драйвера сенсорных и исполнительных устройств, обеспечивающие рефлекторные функции аппарата; б) «тактический» уровень (зеленые модули на схеме) содержит программные модули, организующие выполнение очередного элемента миссии, модуль навигации и модули СТЗ; в) «стратегический» уровень (фиолетовые модули на схеме) состоит из программного модуля, обеспечивающего логику выполнения миссии в целом, и набора модулей, отвечающих за решение конкретной задачи миссии.

Реализация системы управления подводным роботом состоит из отдельных программных модулей, отвечающих за решение следующих основных задач:

- низкоуровневая обработка данных сенсорных систем (модули: *zed2*, *acoustic1*, *bar100* и *vg103* – в скобках указаны параметры запуска, если модуль повторно используется для решения нескольких задач);
- высокоуровневая обработка данных систем технического зрения (модули: *bearing*, *aruco*, *neural* и другие, в зависимости от решаемых аппаратом задач);
- слияние навигационных данных и данных СТЗ (модули: *scene*, *navigation* и *odometry*);
- управление подводным роботом (модули: *mission*, *tasks*, *regulator* и *spreader*);
- выработка управляющих воздействий исполнительным механизмам (модули: *gpio4* и *kotleta1*).

Взаимодействие перечисленных модулей отражено на рис. 2 синими стрелками, которые по своей сути являются сообщениями системы управления. Сообщения в разработанной СУ передаются между модулями системы децентрализованным образом. Каждый модуль системы при этом является отдельным процессом, что в сочетании с децентрализованным механизмом передачи сообщений гарантирует отсутствие «падения» всех модулей при «падении» или некорректной работе одного из них.

Еще одной особенностью разработанной системы управления является соглашение о наименовании и идентификации передаваемых между модулями сообщений. Наименования стрелок на рис. 2 являются одновременно и идентификатором сообщений в системе и именем класса. Такой подход позволяет разработчикам относительно независимо создавать отдельные модули СУ, имея только один общий файл, описывающий структуры передаваемых данных. При этом, например, количество и наименования самих модулей в системе может достаточно часто и независимо друг от друга меняться, что позволяет масштабировать СУ в достаточно широких пределах под решаемые задачи.

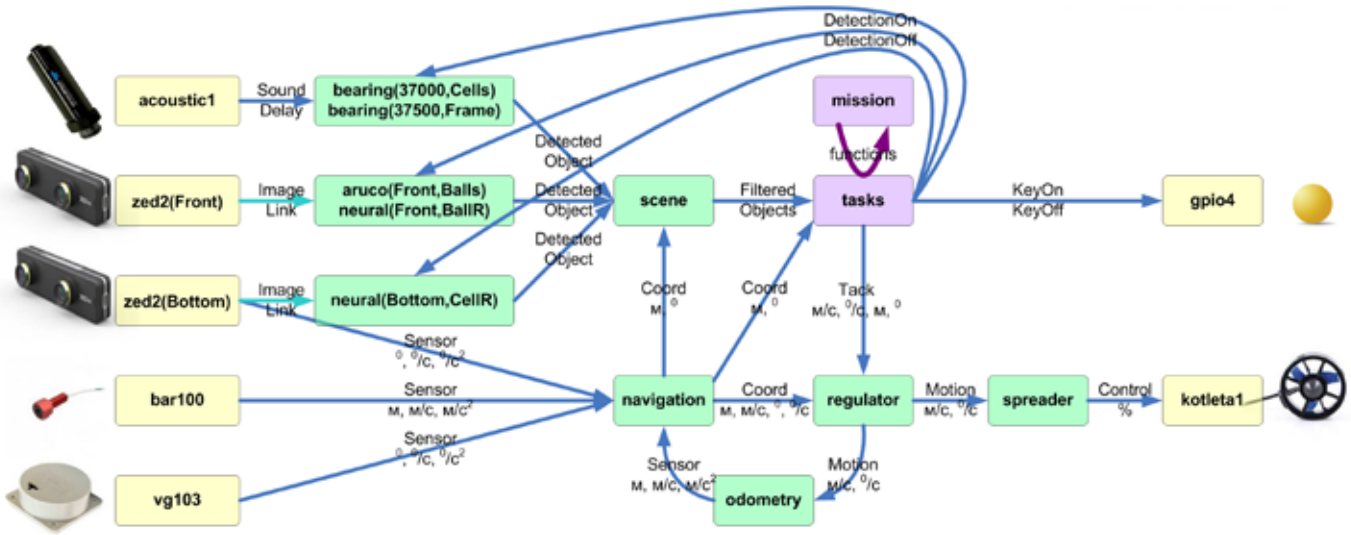


*Рис. 2.* Схема взаимодействия программных модулей системы управления НПА

В разработанной архитектуре СУ изначально был заложен функционал, позволяющий проводить симуляцию работы большей части ПО без использования внешнего оборудования. В режиме симуляции используются все программные модули, не имеющие внешних связей с периферийными устройствам (на рис. 2 обозначены зеленым и фиолетовым цветом). При этом сами модули не обладают информацией о том, производится в данный момент режим симуляции или реальной работы. В свою очередь, модули, непосредственно связанные с оборудованием (рис. 2 – желтые прямоугольники), заменяются их симуляционными аналогами. Назначение данных симуляционных модулей – замкнуть внешний (разомкнутый) контур потока данных и «подменить» все сообщения от сенсорных устройств.

## Использование языка *Python* для создания системы управления

Как указывалось выше, одной из целей разработки программного обеспечения системы управления малогабаритного НПА *Geek* было обучение студентов и аспирантов основам подводной робототехники. С учетом опыта разработки СУ аппаратов *Junior* [17] и *Pandora* [5] (в основе которых были программные платформы *ROS* и *IPC* соответственно) в новом аппарате было отдано предпочтение простоте реализации программного обеспечения. В связи с этим основой для реализации ПО стал язык программирования *Python*, который имеет большую популярность в студенческой среде.

Одним из ключевых элементов при проектировании систем управления НПА является механизм межпроцессного взаимодействия программных модулей. В рамках работ по созданию СУ аппарата *Geek* был разработан программный интерфейс (*API*), предоставляющий функции для отправки и получения модулями специализированных структур данных (сообщений). Основу механизма составляет класс *Message*, являющийся родительским классом для всех структур данных, участвующих в обмене. Данный класс содержит служебные функции и переменные, отвечающие за отправку, прием и идентификацию полученных сообщений. Таким образом, создание нового сообщения, участвующего в межпроцессном взаимодействии модулей СУ, сводится только к наследованию от класса *Message* без применения дополнительных операций, связанных с настройкой каких-либо конфигурационных файлов.

Алгоритм обмена данными между модулями СУ включает следующие основные этапы:

- создание передаваемого потока байт из *Python*-объекта внутри модуля-отправителя;
- передача потока байт всем модулям-потребителям, участвующим в межпроцессном взаимодействии;
- идентификация принятого сообщения модулями-получателями;
- восстановление структуры данных из потока байт каждым модулем-получателем, который заинтересован в чтении данного сообщения.

Создание байтового потока для передачи и восстановление структуры данных после приема производится с использованием популярной в *Python* библиотеки *Pickle*. Данная библиотека предоставляет возможность сериализовать и десериализовать объекты *Python*. Сериализация – это процесс преобразования объекта в поток байтов, который затем может быть сохранен в файл, передан другому модулю на данном компьютере или к другому узлу вычислительной сети НПА. Десериализация – есть обратный процесс, при котором поток байт преобразуется в объект, обрабатываемый модулем-получателем.

В реализованном механизме межпроцессного взаимодействия сериализованный поток байт передается всем модулям-потребителям по сети с использованием протокола *UDP*. Его использование для передачи данных позволяет избежать создания и поддержания соединений между модулями (в отличие, например, от *TCP*), а также обеспечивает максимально быструю (хоть и не гарантированную) доставку сообщений. В разработанной системе обмена сообщениями *UDP*-пакеты отправляются по широковещательному каналу (*broadcast*) на единый для всех модулей-потребителей порт. Подобный подход позволяет получать сообщения всем модулям системы управления как внутри одного компьютера, так и по локальной сети.

Можно выделить следующие особенности реализованного на языке *Python* межпроцессного взаимодействия компонент СУ:

*1. Простота и лаконичность реализации*, в частности, реализация *API* взаимодействия компонент СУ заняла менее 200 строк исходного кода и имеет минимальное количество внешних зависимостей: библиотека *copy* (глубокое копирование для создания нового объекта); библиотека *pickle* (для сериализации и десериализации объектов); библиотека *socket* (для работы с *UDP*-пакетами); библиотека *time* (для доступа к системным часам) и библиотека *json* (для сериализации данных в *json*-формат с целью ведения отладочной печати).

*2. Сознательное упрощение механизма обмена*, которое повлекло ограничение в количестве одновременно взаимодействующих компонент СУ. Поскольку механизм обмена данными между модулями (в общем случае) представляет собой полносвязный граф «каждый-с-каждым» (т.е. опубликованное сообщение доставляется всем модулям), то и рост числа модулей системы приводит к существенной нагрузке на центральный процессор. Однако практика показала, что в рамках системы управления малогабаритного НПА (предназначенного преимущественно для обучения студентов и прототипирования) данное ограничение не является существенным. В частности, при наличии в системе около 15 модулей и частоте навигационно-управляющего контура порядка 20 Гц для реализованной системы управления накладными расходами центрального процессора на организацию обмена сообщениями можно пренебречь.

Для реализации драйверов устройств на языке *Python* в большинстве случаев использовались готовые библиотеки, часто поставляемые разработчиками самих устройств или открытым сообществом. Например, для взаимодействия драйвера фотосистемы со стерекамерами *zed-2* (камера нижнего обзора) и *zed-mini* (камера переднего обзора) применялся официальный *SDK* от компании *Stereolabs*. Датчики глубины, блоки управления двигателями, датчики угловых скоростей, ключи управления полезной нагрузкой и многие другие периферийные устройства НПА *Geek* подключены через стандартные интерфейсы *i2c*, *CAN* и *UART* и другие. Для доступа к таким интерфейсам существует множество встроенного функционала в языке *Python* или имеются сторонние библиотеки. В большинстве случаев написание драйверов устройств разработанной СУ не вызывало трудностей у студентов технических специальностей.

Из особенностей реализации СТЗ на языке *Python* можно отметить:

1. Применение популярных библиотек *OpenCV*, *Pillow*, *NumPy*, *TensorFlow* и др. для решения задач компьютерного зрения часто дает выигрыш во времени реализации ПО при использовании языка *Python* в сравнении с аналогичными реализациями на *C++*.

2. Разработка, анализ и сопровождение исходного кода различными разработчиками одних и тех же компонент СТЗ также проще на *Python* в сравнении с *C++* ввиду лаконичности и краткости *Python*.

3. К недостаткам *Python* следует отнести необходимость осуществления оптимизирующих процедур при реализации некоторых компонент СТЗ. Примером может служить векторизация вычислений с целью отказа от двойных (и более) вложенных циклов при обработке фотоизображений в режиме реального времени.

Реализация большинства модулей «тактического» и «стратегического» уровней (зеленые и фиолетовые модули на рис. 2) на языке *Python* в целом не имеет кардинальных отличий от реализации на языке *C++*. Исключение составляет модуль *mission*, который используется для автономного (беспилотного) управления подводным аппаратом. Автономная миссия подводного аппарата представляет собой последовательность высокоуровневых команд роботу, таких как: следовать указанным курсом (или в указанную точку) на заданной глубине с заданной скоростью, стабилизироваться (стоять в точке) над определенным объектом, привести в действие (сжать/отпустить) манипуляторы и др. Набор команд миссии зависит от выполняемых аппаратом задач, а в случае с НПА *Geek* – еще и от регламента проводимых соревнований.

Существует множество языков описания миссии [18], применение которых обусловлено моделью использования подводного робота. Ранее используемый на АНПА *Pandora* язык миссии представлял собой линейную последовательность действий аппарата, составленную в формате *JSON*. К преимуществам такого подхода следует отнести возможность автоматической верификации текста миссии (до запуска робота) с помощью *JSON-Schema* [19] и даже возможность написания синтаксически верного (безошибочного) текста миссии с помощью различных компонент, основанных на *JSON-Schema*, например [20]. Недостатком данного подхода является меньшая гибкость при составлении миссии, которая заключается в отсутствии (сложности формализации в формате *JSON*) логических ветвлений при описании последовательности выполнения задач. Как показал опыт, данный недостаток является весьма существенным во время выполнения соревновательных задач в категории АНПА [21–23]. В этой связи при разработке системы управления НПА *Geek* на языке *Python* команда сознательно отказалась от «надежного» *JSON*-описания миссии в пользу «гибкого» *Python*-кода.

Для сравнения производительности разработанного механизма взаимодействия программных модулей с предыдущими разработками был проведен ряд экспериментов. Суть экспериментов заключалась в последовательном наращивании количества модулей (участвующих в обмене сообщениями) и измерении временной задержки в прохождении сообщений (от первого модуля до последнего в цепочке), а также измерении нагрузки центрального процессора. Измерения проводились для двух режимов:

- режим «точка-точка», где каждый модуль взаимодействовал только с соседними модулями в цепочке;
- режим «каждый-с-каждым», где каждый модуль получал сообщение от всех модулей системы, т.е. имитировал полносвязанный граф – наихудший с точки зрения производительности вариант конфигурации взаимодействия модулей, но очень простой с точки зрения реализации.

Для проведения экспериментов использовался одноплатный компьютер *Orange Pi5* [24] (который планируется к установке на НПА *Geek* взамен устаревшего *Nvidia Jetson TX2*) со следующими характеристиками:

- восьмиядерный процессор с архитектурой *ARM* (*4xCortex-A76* и *4xCortex-A55*);
- оперативная память 8 ГБ *LPDDR4/4X*.

Результаты проведенных экспериментов представлены на рис. 3. Можно видеть, что разработанная система на языке *Python* (рис. 3 – зеленые графики) достойно конкурирует с ее предшественницей (системой *IPC* – реализованной на языке *C++*, рис. 3 – синие графики), показывая даже лучший результат в некоторых случаях. В частности, в режиме «точка-точка» (сплошные линии) обе реализации дают стабильные показатели временной задержки (менее 1,5 мс, см. рис. 3, *а*) и низкую загрузку центрального процессора (менее 20%, см. рис. 3, *б*), даже при большом количестве модулей в системе (вплоть до 100 программных модулей). В режиме «каждый-с-каждым» (пунктирные линии) при запуске до 20 программных модулей обе системы также показывают стабильную работу, небольшие временные задержки в передаче данных (менее 1,5 мс) и приемлемое линейное возрастание нагрузки процессора (примерно до 20%). Данный факт позволяет сделать выводы о возможности применения простой системы обмена сообщениями, построенной по принципу взаимодействия «каждый-с-каждым», для небольших систем, обладающих общим числом взаимодействующих модулей до 20 шт. При этом язык реализации (*Python* или *C++*) фактически не имеет существенного значения.

При увеличении количества программных модулей в режиме «каждый-с-каждым» обе системы обмена сообщениями имеют существенный рост нагрузки на центральный процессор (рис. 3, *б* – пунктирные линии). При этом система обмена на *C++* перестает справляться с передачей данных (резкий рост графиков на рис. 3, *а*) примерно на 40 взаимодействующих модулях, а система на *Python* – на 70. Для обеих реализаций характерна существенная нагрузка при увеличении количества взаимодействующих модулей, что является естественным ограничением полносвязанной модели «каждый-с-каждым». Некоторые различия в поведении графиков (как временной задержки, так и нагрузки процессора) объясняются не только отличиями в языках реализации, но также и накладными расходами двух решений. Например, имеющаяся система связи на языке *C++*, помимо непосредственного обмена сообщениями, имеет встроенный функционал накопления передаваемых данных и взаимодействия с *WEB*-браузером и не предназначена для использования в режиме обмена «каждый-с-каждым». Этим фактом объясняются несколько худшие ее показатели в части нагрузки ЦПУ в данном режиме.

В результате реализации новой системы управления для НПА *Geek* общее количество строк кода на языке *Python* составило 896, из которых 95 строк занимают комментарии. Для сравнения, предыдущая

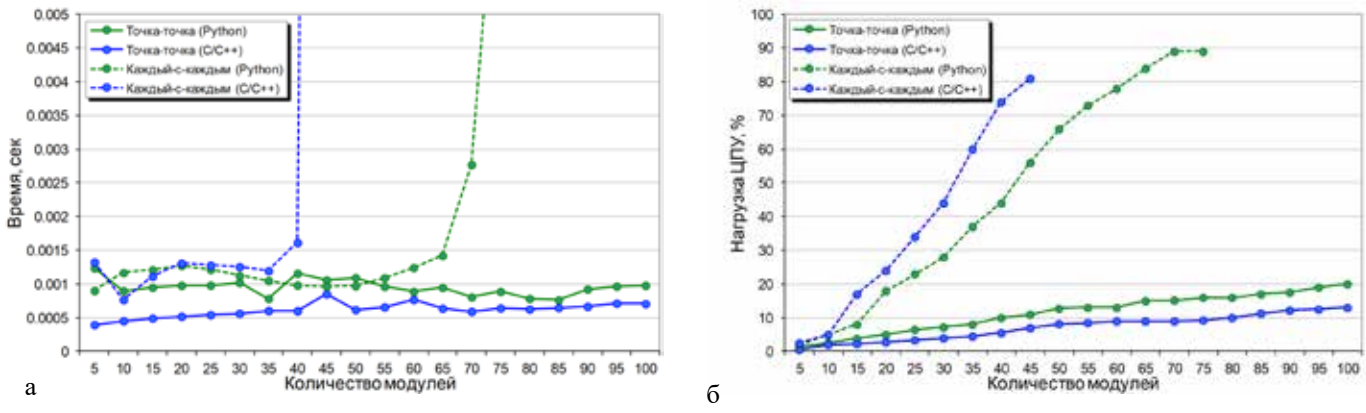


*Рис. 3.* Сравнение систем обмена сообщениями, реализованными с применением языков *C++* и *Python*: а – зависимость времени прохождения сообщений от количества модулей; б – рост нагрузки центрального процессора от количества модулей

версия СУ на языке *C++* для АНПА *Pandora* насчитывала 8914 строк кода и 4911 строк комментариев. Стоит отметить, что реализация СУ на языке *Python* является более «молодой» и «сознательно упрощенной», с целью быстрого приобщения обучающихся к специфике разработки ПО в области подводной робототехники.

## Особенности применения НПА *Geek* в качестве автономного и телеуправляемого подводного робота

Для использования подводного робота в соревнованиях по подводной робототехнике в классе телеуправляемых аппаратов, а также для отладки автономных функций в режиме телеуправления для НПА *Geek* был разработан специализированный пост управления роботом. Пост включает в себя следующее оборудование (рис. 4):

- ноутбук с подключенным дополнительным монитором для отображения видеопотока с двух камер;
- ноутбук для отображения параметров движения (курс, скорость, и т.д.), траектории движения робота и распознанных им объектов;
- джойстик (*Sony PlayStation Dualshock 4*) для управления роботом в супервизорном режиме.

Особенностью реализации ПО поста управления на языке *Python* является способ организации передачи видеопотока с НПА *Geek* на ноутбуки оператора. Видеопоток представлен в виде передаваемой серии фотоизображений в формат *JPEG* с разрешением 480×270 пикселей. Сформированные в драйвере камеры фотоизображения отправляются в сеть по протоколу *UDP* на пост управления наравне с остальными сообщениями системы управления. Отличие от обычных сообщений СУ заключается только в номере порта, который был выбран иным, с целью доставки большого потока данных только одному адресату. На приемной стороне полученные данные интерпретируются как изображение и выводятся на экран монитора с использованием библиотек *tkinter* и *ImageTk*. Опыт эксплуатации такой схемы показал, что при использовании проводной сети *Ethernet 100 Mbit* с длиной кабеля 50 м достигается частота обновления кадров более 10 кадров/с с двух стереокамер. Такие показатели вполне достаточны для управления аппаратом пилотом-оператором, а сама реализация решения на языке *Python* заняла менее 100 строк кода.

В результате эксплуатации НПА *Geek* в соревнованиях в классе АНПА и ТНПА можно сделать ряд выводов:

1. Программное обеспечение системы управления без учета нагрузки СТЗ задействует не более 30% ресурсов всех ядер центрального процессора *NVIDIA Jetson TX2*. Нагрузка на процессор системы технического зрения очень сильно зависит от решаемых задач. В режиме ТНПА, когда отключены модули распознавания объектов на фотоизображениях, нагрузка составляет около 35%.

2. Цикл регулирования движения подводного робота осуществлялся на частоте 20 Гц. При этом удер-

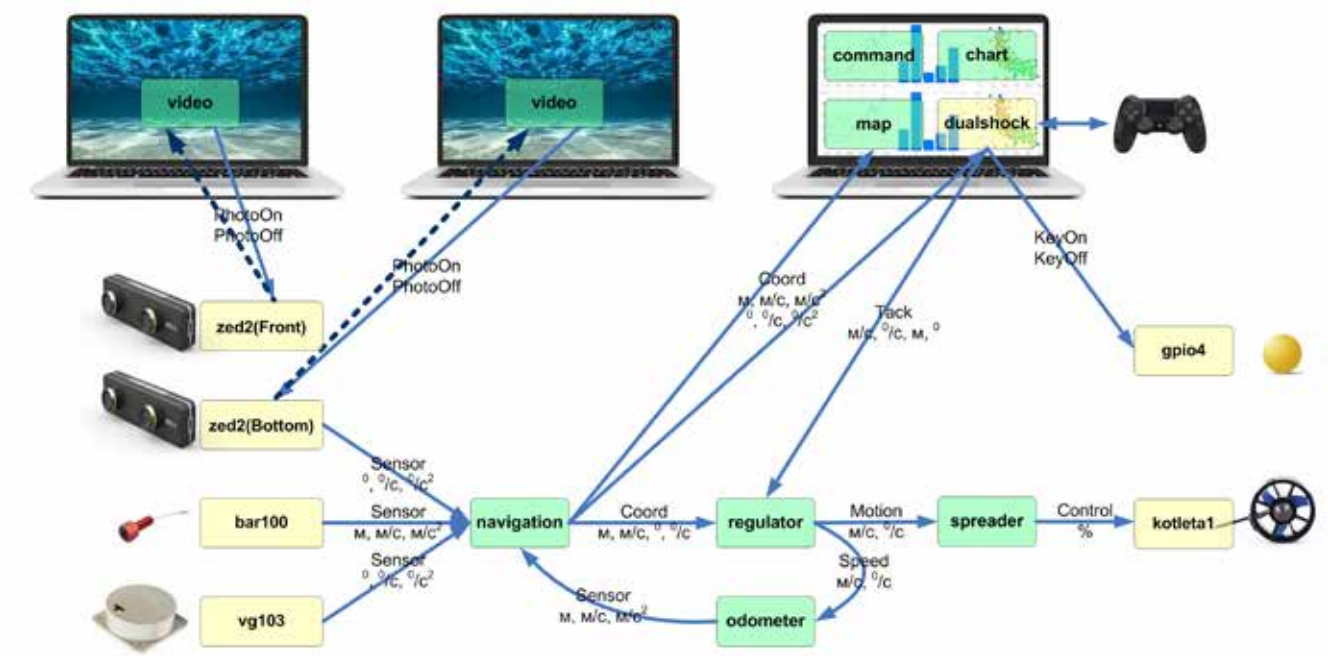


*Рис. 4.* Схема взаимодействия поста управления с автопилотом НПА *Geek*

жание угловых величин (курса, крена и дифферента) аппарата находится в пределах 0.1 градуса, а погрешность удержания глубины не превышает нескольких сантиметров.

3. Частота обработки фотокадров системой технического зрения устанавливается в зависимости от решаемых задач и составляет: от 4 кадров/с (при работе в режиме АНПА и включенном распознавании объектов на фотоизображениях) до 10–15 кадров/с (в режиме ТНПА). С учетом независимого контура регулирования движения с частотой 20 Гц приведенных показателей достаточно для высокоточного позиционирования аппарата в обоих режимах.

Реализация системы управления НПА *Geek* «с нуля» на языке *Python* заняла около трех месяцев при работе командой из трех студентов, одного аспиранта и двух наставников команды приблизительно по 1 часу в день (подсчитать точное время разработки не представляется возможным). Благодаря простоте и лаконичности как самого языка, так и реализуемых подходов в разработке смогли принять участие студенты 1-го и 2-го курсов. Активное использование языка *Python* выявило следующие положительные стороны:

1. Язык позволяет быстро и эффективно провести этап прототипирования и без особых усилий поддерживать исходный код в рабочем состоянии даже силами малочисленного студенческого коллектива.

2. Исходный код *Python* в большинстве случае получается лаконичным, минималистичным и легко воспринимаемым программистами любого уровня.

3. Наличие интерпретатора и отсутствие этапа компиляции зачастую ускоряет процесс отладки ПО и повышает скорость его доработки.

4. Модульность и мультипарадигменность языка позволяет разработчикам повторно использовать наработки других членов команды, а большое количество готовых библиотек и решений ускоряет решение поставленных задач.

Однако применение языка *Python* при разработке системы управления подводного робота выявило и ряд недостатков:

1. Небольшая скорость вычислений и сложность оптимизации. Многие алгоритмические реализации в *Python* выполняются медленнее в сравнении с *C++,* особенно связанные с вложенной циклической обработкой данных. При сравнении времени выполнения большинства стандартных операторов *C++* и *Python* будет получена часто десятикратная разница. Для устранения данного недостатка существует необходимость прибегать к оптимизирующим процедурам с использованием различных библиотек, что требует от разработчиков глубоких знаний в специфичной области и времени.

2. Повышенные требования к объему оперативной памяти. Как правило, код на языке Python более требователен к оперативной памяти, в сравнении с аналогичным кодом на языке *C++*. В контексте работы на современных вычислительных ресурсах подобный минус не является существенным, но не является таковым для реализации долгосрочных решений на борту подводных роботов с относительно маломощными энергоэффективными вычислителями.

3. Отсутствие строгой типизации и проверок на этапе компиляции. *Python* не требует от разработчика строгого указания типов, а предлагает латентную типизацию с возвратом ошибки в процессе выполнения кода. Сложность поиска ошибок во время исполнения подобного кода на борту АНПА – очевидна.

## Заключение

Реализованная система управления НПА *Geek* на языке *Python* иллюстрирует возможность использования скриптовых языков программирования при создании прототипов систем управления для подводных роботов. Применение *Python* зачастую позволяет значительно ускорить разработку различных модулей системы управления за счет отсутствия стадии компиляции и наличия множества готовых библиотек и решений. Одновременно с этим невысокая скорость вычислений, сложности с оптимизацией, повышенные требования к вычислительным ресурсам и ненадежность системы в целом из-за отсутствия стадии компиляции весьма ограничивают применение данного языка для построения полномасштабной СУ промышленного автономного подводного робота. Дальнейшие исследования в направлении использования языка *Python* для НПА видятся в реализации отдельных частей СУ и СТЗ на *Python* в комбинации с уже проверенными решениями на *C++*. Также интересным видится направление, связанное с компиляцией *Python*-скриптов в исполняемые файлы с применением таких библиотек, как *PyInstaller*, *auto-py-to-exe* и др.

Авторы выражают признательность всему коллективу студенческой команды ИПМТ–ДВФУ за многолетний совместный плодотворный труд по созданию НПА *Geek*, Платону Пряженникову и Владу Гой – за проработку и реализацию конструктивного облика НПА *Geek*, а также Инзарцеву Александру Вячеславовичу – за многочисленную и терпеливую корректуру материалов данной публикации.

## СПИСОК ИСТОЧНИКОВ


1. Инзарцев А.В., Боровик А.И., Баль Н.В. Разработка программного обеспечения системы управления АНПА на базе модифицированной платформы Player // Материалы Шестой Всероссийской научно-практической конференции «Перспективные системы и задачи управления» и Третьей молодежной школы-семинара «Управление и обработка информации в технических системах». Таганрог: ТТИ ЮФУ, 2011. С. 380–391.

2. Наумов Л.А., Боровик А.И., Баль Н.В. Программная платформа для системы управления АНПА // Четвертая всероссийская научно-техническая конференция «Технические проблемы освоения Мирового океана»: материалы конференции. Владивосток: ИПМТ, 2011. С. 352–356.

3. Боровик А.И., Наумов Л.А. Проект системы управления АНПА на базе компонентно-ориентированной программной платформы RCE // Пятая всероссийская научно-техническая конференция «Технические проблемы освоения Мирового океана»: материалы конференции. Владивосток: ИПМТ, 2013. С. 435–439.

4. Robot Operating System / ROS. URL: https://www.ros.org/ (дата обращения: 13.12.2024).

5. Студенческий проект робота «Pandora». URL: https://robonation.org/app/uploads/sites/4/2021/07/RoboSub_2021_Far-Eastern-Federal-U_TDR.pdf (дата обращения: 13.12.2024).

6. Duke University. URL: https://duke-robotics.com/robosub-2021/ (дата обращения: 13.12.2024).

7. Singapore Nation University Bumblebee AUV. URL: https://bumblebee.sg/ (дата обращения: 13.12.2024).

8. Isaac SDK. URL: https://developer.nvidia.com/isaac (дата обращения: 13.12.2024).

9. Платформа NVIDIA Carter. URL: https://docs.nvidia.com/isaac/archive/2021.1/doc/tutorials/carter_hardware.html (дата обращения: 13.12.2024).

10. Проект BMW STR на базе Isaac SDK. URL: https://www.greencarcongress.com/2020/05/20200515-nvidia.html (дата обращения: 13.12.2024).

11. Проект Робопёс на Isaac SDK. URL: https://omega-industrial.ru/article/nvidia-isaac-sdk (дата обращения: 13.12.2024).

12. Инзарцев А.В., Павин А.М., Елисеенко Г.Д, Родькин Д.Н., Сидоренко А.В., Лебедко О.А., Панин М.А. Реконфигурируемая кроссплатформенная среда моделирования поведения необитаемого подводного аппарата // Подводные исследования и робототехника. 2015. № 2(20). С. 28–34.

13. Елисеенко Г.Д., Инзарцев А.В., Павин А.М. Программная платформа для построения распределенных систем управления морских роботизированных комплексов // Подводные исследования и робототехника. 2019. № 30. С. 13–20.

14. Свидетельство о государственной регистрации программ для ЭВМ. Программная платформа для информационного взаимодействия функциональных компонентов в робототехнических системах: № 2019610890, опубл. 18.01.2019 / Елисеенко Г.Д., Павин А.М., Инзарцев А.В., Сидоренко А.В. Заявитель и патентообладатель ИПМТ ДВО РАН.

15. Саммерфилд М. Программирование на Python 3. Подробное руководство. СПб.: Символ Плюс, 2009. 608 с.

16. Джозеф Л. Изучение робототехники с помощью Python. М.: ДМК Пресс, 2019. 250 с.

17. Студенческий проект робота «Junior». URL: https://robonation.org/app/uploads/sites/4/2019/10/FEFU_RS15_Paper.pdf. (дата обращения: 13.12.2024).

18. Pavin A., Inzartsev A. A GeoJSON-based Mission Planning Language for AUV (AUVGeoJSON Language) // OCEANS 2018 MTS/IEEE. Charleston, 2018. P. 1–5.

19. Документация по JSON-схеме. URL:https://json-schema.org/ (дата обращения: 13.12.2024).

20. Реализация компоненты на основе JSON-схемы для языка программирования JavaScript. URL: https://www.jeremydorn.com/json-editor (дата обращения: 13.12.2024).

21. Международные соревнования WUURC–MURC. URL: https://w2urc.org/en (дата обращения: 13.12.2024).

22. Многонациональные соревнования по подводной робототехнике MURC-WUURC. URL: https://murc.pro (дата обращения: 13.12.2024).

23. Открытое первенство Санкт-Петербурга по морской робототехнике. URL: https://www.smtu.ru/ru/viewnews/833 (дата обращения: 13.12.2024).

24. Официальный сайт производителя одноплатных компьютеров OrangePi. URL: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-5.html (дата обращения: 13.12.2024).


## Справка об авторах:


**ЕЛИСЕЕНКО Григорий Дмитриевич,** научный сотрудник, лаб. Систем управления
Институт проблем морских технологий им. академика М.Д. Агеева Дальневосточного отделения Российской академии наук
**Адрес места работы:** Россия, 690091, г. Владивосток, ул. Суханова, д.5а
**Область научных интересов:** программирование, система управления, моделирование
**E-mail:** eliseenko@marine.febras.ru
**ORCID:** 0000-0003-0386-0068

**ПАВИН Александр Михайлович,** к.т.н., в.н.с. лаб. Систем технического зрения
Федеральное государственное бюджетное учреждение науки Институт проблем морских технологий им. академика М.Д. Агеева Дальневосточного отделения Российской академии наук
**Адрес места работы:** Россия, 690091, г. Владивосток, ул. Суханова, д.5а
**Область научных интересов:** системы технического зрения, распознавание образов, управление подвижными объектами, НПА, АНПА, ТНПА
**E-mail:** pavin@bk.ru
**ORCID:** 0000-0001-8878-7888
**WoS ResearcherID:** AAO-2252-2020
**Scopus AuthorID:** 24765383300
**SPIN-код:** 8565-7211
**РИНЦ AuthorID:** 508298

**ТОРЖКОВ Александр Сергеевич,** техник, лаб. Систем управления
Институт проблем морских технологий им. академика М.Д. Агеева Дальневосточного отделения Российской академии наук
**Адрес места работы:** Россия, 690091, г. Владивосток, ул. Суханова, д.5а
**Область научных интересов:** программирование, системное программирование, подводная робототехника
**E-mail:** torzhkov.as@dvfu.ru
**ORCID:** 0009-0000-3950-4028

**ШИЛИН Константин Дмитриевич**, младший научный сотрудник лаборатории систем технического зрения
Институт проблем морских технологий им. академика М.Д. Агеева Дальневосточного отделения Российской академии наук
**Адрес места работы**: Россия, 690091, г. Владивосток, ул. Суханова, д.5а
**Область научных интересов:** техническое зрение, компьютерное зрение, нейронные сети, подводная робототехника
**E-mail:** konshilin@gmail.com
**ORCID:** 0000-0003-3322-0184

DOI: 10.37102/1992-4429_2025_52_02_06

# EXPERIENCE OF PYTHON LANGUAGE USAGE FOR AUV CONTROL SYSTEM DEVELOPMENT

**G.D. Eliseenko, A.M. Pavin, A.S. Torzhkov, K.D. Shilin**

Developing software for underwater vehicle control systems requires both the adaptation of existing solutions and the development of new ones. Various programming tools can be involved in the process, the range of which is currently quite wide. This article describes the experience of using Python language to develop control systems «from scratch» for unmanned underwater vehicles (autonomous underwater vehicle (AUV) and remote operated vehicle (ROV)), using the example of developing a small-sized Geek vehicle. The feature of this vehicle lies in its purpose — teaching students and postgraduates the basics of designing AUV, as well as involving student teams in underwater robotics competitions. Various aspects of using the Python are considered, including inter-process communication, data processing from sensors and systems of AUV, autonomous management, and interaction with operators, as well as integration with other technologies necessary for the successful completion of missions by the vehicle underwater. The results of using the underwater robot Geek in the modes of AUV and ROV are discussed.

**Keywords:** control system, unmanned underwater vehicle, software, inter-process communication, Python language, UUV, AUV, ROV.

## References

1. Inzarcev A.V., Borovik A.I., Bal' N.V. Razrabotka programmnogo obespecheniya sistemy upravleniya ANPA na baze modificirovannoj platformy Player // Materialy Shestoj Vserossijskoj nauchno-prakticheskoj konferencii «Perspektivnye sistemy i zadachi upravleniya» i Tret'ej molodezhnoj shkoly-seminara «Upravlenie i obrabotka informacii v tekhnicheskih sistemah». Taganrog: TTI YuFU, 2011. P. 380–391. [In Russ.]

2. Naumov L.A., Borovik A.I., Bal' N.V. Programmnaya platforma dlya sistemy upravleniya ANPA // Chetvertaya vserossijskaya nauchno-tekhnicheskaya konferenciya «Tekhnicheskie problemy osvoeniya Mirovogo okeana»: materialy konferencii. Vladivostok: IPMT, 2011. P. 352–356. [In Russ.]

3. Borovik A.I., Naumov L.A. Proekt sistemy upravleniya ANPA na baze komponentno-orientirovannoj programmnoj platformy RCE // Pyataya vserossijskaya nauchno-tekhnicheskaya konferenciya «Tekhnicheskie problemy osvoeniya Mirovogo okeana»: materialy konferencii. Vladivostok: IPMT, 2013. P. 435–439. [In Russ.]

4. Robot Operating System / ROS. URL: https://www.ros.org/ (Access date: 13.12.2024).

5. Studencheskij proekt robota «Pandora». URL: https://robonation.org/app/uploads/sites/4/2021/07/RoboSub_2021_Far-Eastern-Federal-U_TDR.pdf (data obrashcheniya: 13.12.2024). [In Russ.]

6. Duke University. URL: https://duke-robotics.com/robosub-2021/ (дата обращения: 13.12.2024).

7. Singapore Nation University Bumblebee AUV. URL: https://bumblebee.sg/ (дата обращения: 13.12.2024).

8. Isaac SDK. URL: https://developer.nvidia.com/isaac (дата обращения: 13.12.2024).

9. Platforma NVIDIA Carter. URL: https://docs.nvidia.com/isaac/archive/2021.1/doc/tutorials/carter_hardware.html (data obrashhenija: 13.12.2024). [In Russ]

10. Proekt BMW STR na baze Isaac SDK. URL: https://www.greencarcongress.com/2020/05/20200515-nvidia.html (data obrashhenija: 13.12.2024). [In Russ]

11. Proekt Robopjos na Isaac SDK. URL: https://omega-industrial.ru/article/nvidia-isaac-sdk (data obrashhenija: 13.12.2024). [In Russ]

12. Inzarcev A.V., Pavin A.M., Eliseenko G.D, Rod'kin D.N., Sidorenko A.V., Lebedko O.A., Panin M.A. Rekonfiguriruemaya krossplatformennaya sreda modelirovaniya povedeniya neobitaemogo podvodnogo apparata // Podvodnye issledovaniya i robototekhnika. 2015. № 2(20). P. 28–34. [In Russ.]

13. Eliseenko G.D., Inzarcev A.V., Pavin A.M. Programmnaya platforma dlya postroeniya raspredelennyh sistem upravleniya morskih robotizirovannyh kompleksov // Podvodnye issledovaniya i robototekhnika. 2019. № 30. P. 13–20. [In Russ.]

14. Svidetel'stvo o gosudarstvennoj registracii programm dlja JeVM. Programmnaja platforma dlja informacionnogo vzaimodejstvija funkcional'nyh komponentov v robototehnicheskih sistemah: № 2019610890, opubl. 18.01.2019 / Eliseenko G.D., Pavin A.M., Inzarcev A.V., Sidorenko A.V. Zajavitel' i patentoobladatel' IPMT DVO RAN. [In Russ]

15. Sammerfild M. Programmirovanie na Python 3. Podrobnoe rukovodstvo. SPb.: Simvol Plyus, 2009. 608 p. [In Russ.]

16. Dzhozef L. Izuchenie robototekhniki s pomoshch'yu Python. M.: DMK Press, 2019. 250 p. [In Russ.]

17. Studencheskij proekt robota «Junior». URL: https://robonation.org/app/uploads/sites/4/2019/10/FEFU_RS15_Paper.pdf. (data obrashhenija: 13.12.2024). [In Russ]

18. Pavin A., Inzartsev A, 'A GeoJSON-based Mission Planning Language for AUV (AUVGeoJSON Language)' // OCEANS 2018 MTS/IEEE. Charleston, 2018. P. 1–5.

19. Dokumentacija po JSON-sheme. URL:https://json-schema.org/ (data obrashhenija: 13.12.2024). [In Russ]

20. Realizacija komponenty na osnove JSON-shemy dlja jazyka programmirovanija JavaScript. URL: https://www.jeremydorn.com/json-editor (data obrashhenija: 13.12.2024). [In Russ]

21. Mezhdunarodnye sorevnovanija WUURC–MURC. URL: https://w2urc.org/en (data obrashhenija: 13.12.2024). [In Russ]

22. Mnogonacional'nye sorevnovanija po podvodnoj robototehnike MURC-WUURC. URL: https://murc.pro (data obrashhenija: 13.12.2024). [In Russ]

23. Otkrytoe pervenstvo Sankt-Peterburga po morskoj robototehnike. URL: https://www.smtu.ru/ru/viewnews/833 (data obrashhenija: 13.12.2024). [In Russ]

24. Oficial'nyj sajt proizvoditelja odnoplatnyh komp'juterov OrangePi. URL: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-5.html (data obrashhenija: 13.12.2024). [In Russ]

## Information about the authors

**ELISEENKO Grigorii D.**, Research scientist of the laboratory of control systems
Institute of Marine Technology Problems, Far Eastern Branch of the Russian Academy of Sciences
**Address:** 5-a, Sukhanov street, Vladivostok, 690950, Russia
**Research Interests:** programming, control system, modeling
**E-mail:** eliseenko@marine.febras.ru
**ORCID:** 0000-0003-0386-0068

**PAVIN Alexander M.** , Leading researcher, Head of Computer vision Lab.
Institute of Marine Technology Problems, Far Eastern Branch of the Russian Academy of Sciences
**Address:** 5-a, Sukhanov street, Vladivostok, 690950, Russia
**Research Interests:** computer vision systems, pattern recognition, control systems, UUV, AUV, ROV
**E-mail:** pavin@bk.ru
**ORCID:** 0000-0001-8878-7888
**WoS ResearcherID:** AAO-2252-2020
**Scopus AuthorID:** 24765383300
**SPIN-code:** 8565-7211

**TORZHKOV Alexander S.,** Technician of the laboratory of control systems
Institute of Marine Technology Problems, Far Eastern Branch of the Russian Academy of Sciences
**Address:** 5-a, Sukhanov street, Vladivostok, 690950, Russia
**Research Interests:** computer science, system programming, underwater robotics
**E-mail:** torzhkov.as@dvfu.ru
**ORCID:** 0009-0000-3950-4028

**SHILIN Konstantin D.**, Researcher assistant of Computer vision Lab.
Institute of Marine Technology Problems, Far Eastern Branch of the Russian Academy of Sciences
**Work address:** 5-a, Sukhanov street, Vladivostok, 690950, Russia
**Research Interests:** technical vision, computer vision, neural networks, underwater robotics
**E-mail:** konshilin@gmail.com
**ORCID:** 0000-0003-3322-0184