Разработка интегрированной системы компьютерного зрения и управления походкой для инсектоморфного робота

В. А. Егунов, В. И. Конченков, И. Д. Полухин, В. А. Зыбин

Волгоградский государственный технический университет

Аннотация: Данная работа посвящена описанию разработки и интеграции двух ключевых подсистем инсектоморфного шестиногого робота: модуля управления походкой и системы компьютерного зрения для автономной навигации. Рассматриваются архитектурные решения, алгоритмические основы и практическая реализация компонентов, обеспечивающих стабильное передвижение и интеллектуальное взаимодействие робота с окружающим пространством.

Ключевые слова: инсектоморфный робот, модуль управления походкой, компьютерное зрение, автономная навигация, ROS2, SLAM, RTABMap, NAV2, OctoMap, триподная походка, Raspberry Pi, LiDAR.

Введение

Развитие мобильной робототехники, в частности инсектоморфных (паукообразных) роботов, открывает перспективы для решения сложных задач в неструктурированных и труднодоступных средах. Высокая маневренность и адаптивность таких платформ делают их востребованными в спасательных операциях, исследованиях и мониторинге. Однако реализация их потенциала требует создания комплексных систем управления, объединяющих точное управление движением и развитые возможности восприятия окружающей среды [1].

Модуль управления походкой шестиногого робота

Эффективное передвижение шестиногих роботов в сложных условиях требует точной и скоординированной работы множества исполнительных механизмов. Разработка модуля управления походкой была нацелена на обеспечение плавной, адаптивной и устойчивой локомоции, а также на создание гибкого интерфейса для интеграции с высокоуровневыми системами, такими как система компьютерного зрения.

Архитектура и компоненты системы управления походкой

Аппаратной основой модуля управления походкой служит современный одноплатный компьютер Raspberry Pi 5 под управлением операционной системы Ubuntu 24.04. Данная платформа обладает достаточной вычислительной мощностью для обработки алгоритмов управления движением и взаимодействия с периферийными устройствами. Управление восемнадцатью сервоприводами MG996R, отвечающими за движение суставов ног робота, осуществляется посредством двух драйверов PCA9685. Связь между Raspberry Pi 5 и драйверами сервоприводов реализована через интерфейс I²C. Внешний вид робота изображен на рисунке 1.

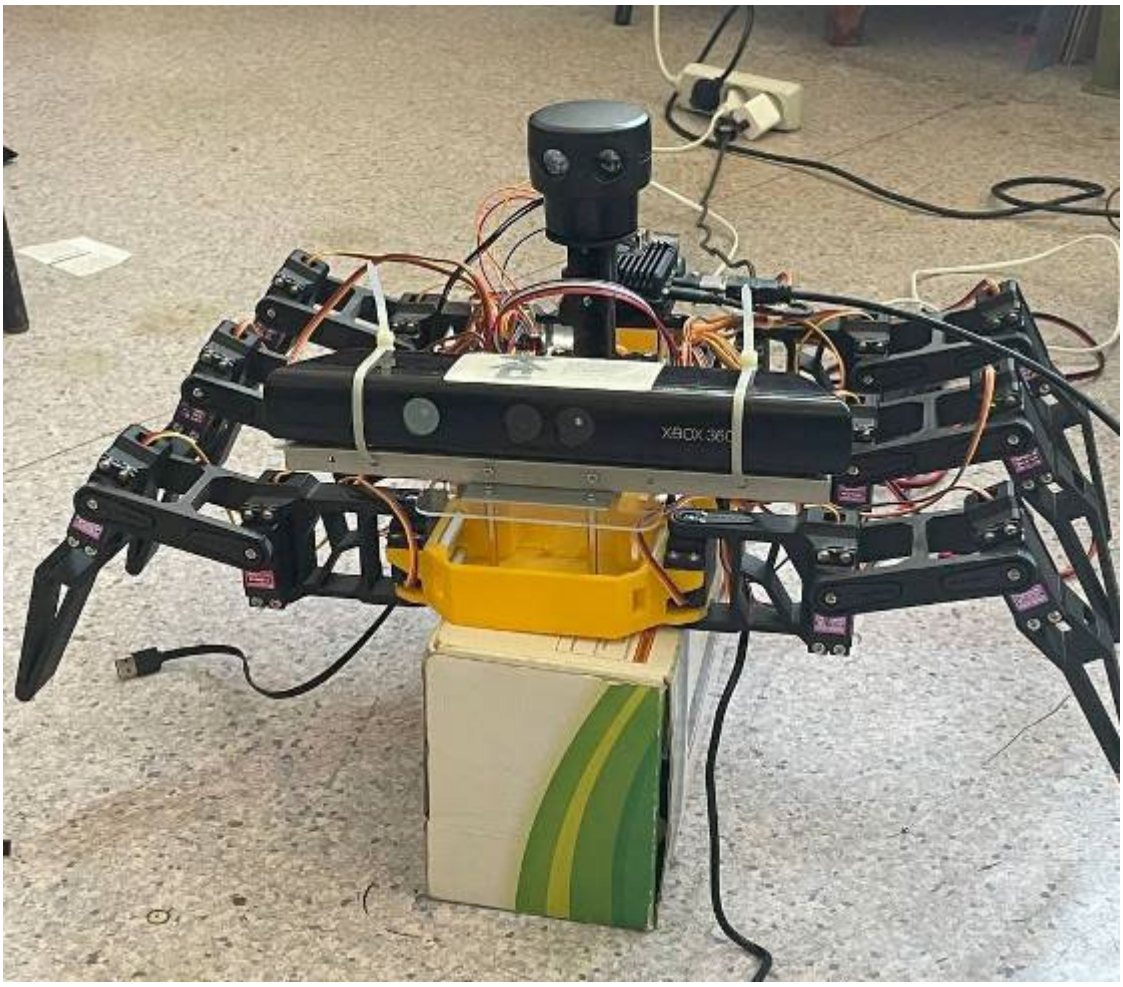

Рис. 1 - Внешний вид робота

Сервоприводы MG996R с металлическими шестернями выбраны как компромисс между стоимостью и достаточной мощностью, несмотря на присущие им невысокую точность (±5°) и заметный люфт при нагрузке, но наличие металлических шестеренок в составе сервоприводов дает высокую надежность при длительных и больших нагрузках. Металл способен выдерживать большие температуры и меньше изнашиваются в отличие от пластмассовых аналогов. В качестве источника данных одометрии для определения положения робота в пространстве может использоваться смартфон, передающий информацию по локальной сети. На рисунке ниже представлен сервопривод MG996R.

Рис. 2. - Сервопривод MG996R

Программная часть модуля разработана на языке Python с использованием платформы ROS2 Jazzy [2]. Ключевым элементом является класс SpiderRobot, инкапсулирующий логику управления роботом. Этот класс включает методы для инициализации, калибровки сервоприводов и реализации различных типов движений. Система также включает сервер для

интеграции с системой компьютерного зрения через WebSocket и сервер одометрии для получения данных со смартфона.

Функциональные возможности модуля управления походкой

Одной из важнейших функций системы является калибровка сервоприводов. Она необходима для компенсации механических погрешностей сборки и индивидуальных особенностей сервоприводов. Реализовано два режима калибровки:

Базовая калибровка: автоматическая установка всех сервоприводов в нейтральное положение (90 градусов).

Интерактивная калибровка: позволяет пользователю тонко настраивать индивидуальные смещения (оффсеты) для каждого сервопривода. Эти смещения сохраняются в JSON-файл и автоматически применяются при последующих запусках системы. Процесс интерактивной калибровки осуществляется через консольный интерфейс, предоставляя визуальный контроль положения конечностей. Ниже представлен пример сохраняемого файла с офсетами.

```
[
    { "coxa": 20, "femur": 0, "tibia": 0 },
    { "coxa": 8, "femur": -13, "tibia": 4 },
    { "coxa": 10, "femur": 6, "tibia": 9 },
    { "coxa": -2, "femur": -1, "tibia": 10 },
    …
]
```

Центральным элементом локомоции является алгоритм управления походкой. Основным реализованным и протестированным алгоритмом является треугольная (триподная) походка. При такой походке шесть ног робота разделяются на две группы по три ноги (триподы). Движение происходит попеременным перемещением этих групп: пока одна группа

находится на земле, обеспечивая устойчивость, другая группа поднимается, перемещается вперед (или назад) и опускается. На рисунке 3 изображены такты такой походки. Этот цикл обеспечивает постоянное наличие трех точек опоры, образующих треугольник, что гарантирует статическую устойчивость робота [3]. Несмотря на простоту реализации и высокую стабильность, треугольная походка менее адаптивна к сильно неровным поверхностям по сравнению с более сложными алгоритмами, такими как волнообразная или адаптивные походки.

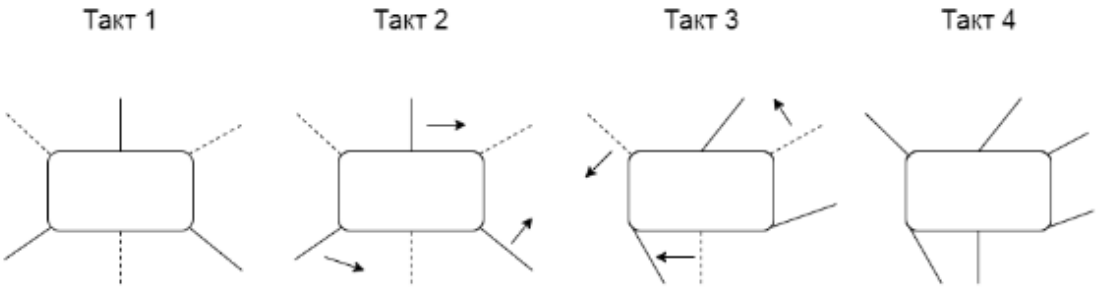

Рис. 3. - Такты треугольной походки инсектоморфного робота

В рамках исследования также рассматривались волнообразная походка, походка "рысью" (trot) и "иноходь" (pace), каждая из которых имеет свои преимущества и недостатки в контексте скорости, стабильности и адаптивности. Так, например, походка "рысью" обеспечивает хороший баланс между скоростью и устойчивостью благодаря диагональному движению ног, но при этом может потребовать корректировки на неровных поверхностях. А походка "иноходь" позволяет более плавное прямолинейное перемещение и является энергоэффективной, однако менее устойчива при поворотах или движении по сложному рельефу из-за синхронного движения ног с одной стороны корпуса.

Реализация триподной походки в классе SpiderRobot включает следующие фазы движения для каждой группы ног:

Подъем трипода: Сервоприводы, отвечающие за вертикальное перемещение ног (femur), изменяют свое положение, поднимая ноги на заданную высоту (step_height).

```
def lift_tripod(self, leg_nums, lift_height=20):
    # Поднимаем ноги на заданную высоту
    for leg in leg_nums:
        self._move_servo(leg, 'femur', target_angle) # target_angle рассчитывается исходя из lift_height
```

Перемещение трипода: Сервоприводы тазобедренных суставов (coxa) изменяют углы для перемещения поднятых ног вперед или назад на длину шага (step_length).

```
def move_tripod_forward(self, leg_nums, step_length=30):
    # Плавное изменение угла coxa
    self._smooth_move_legs(leg_nums, 'coxa', start_angles, target_angles) # target_angles для шага вперед
```

Опускание трипода: Ноги возвращаются в контакт с поверхностью.

```
def lower_tripod(self, leg_nums):
    # Возвращаем ноги в исходное положение
    self._smooth_move_legs(leg_nums, 'femur', current_angles, default_angles)
```

Плавность движений достигается за счет метода _smooth_move_legs, который интерполирует углы сервоприводов между начальным и конечным положениями с заданным количеством шагов и задержкой (movement_delay), которая задается как параметр в программе позволяя менять ее только в одном месте программы, но применяемая во многих местах. Кинематика движения для одного шага может быть упрощенно описана следующим образом, где θ_{coxa} – угол поворота в тазобедренном суставе, θ_{femur} – угол в

коленном суставе, T – период шага, A – амплитуда вертикального движения ноги, а $\Delta\theta$ – изменение угла для горизонтального перемещения:

$$\theta_{coxa}(t) = \theta_{coxa_0} + \Delta\theta \cdot sin(\frac{2\pi t}{T})$$

$$\theta_{femur}(t) = \theta_{femur_0} + A \cdot cos(\frac{2\pi t}{T})$$

Система поддерживает базовые команды движения: вперед, назад, поворот влево, поворот вправо и сброс в исходное (нейтральное) положение.

Интеграция с внешними системами является ключевым аспектом. Для взаимодействия с системой компьютерного зрения реализован WebSocket-сервер в классе MotionExecutor. Этот сервер принимает JSON-сообщения с командами управления (например, 'F' - вперед, 'B' - назад, 'L' - влево, 'R' - вправо, 'S' - стоп/сброс).

```
async def handle_command(self, websocket):
    async for message in websocket:
        try:
            data = json.loads(message)
            cmd = data.get('cmd')
            if cmd in self.command_map:
                self.command_map[cmd]() # Выполнение команды роботом
        except Exception as e:
            print(f"Ошибка обработки команды: {e}")
```

В результате была получена следующая последовательная UML диаграмма взаимодействия компонентов походки внутри самого модуля.

Такая архитектура позволяет модулю управления походкой получать команды от высокоуровневой системы навигации и выполнять соответствующие маневры.

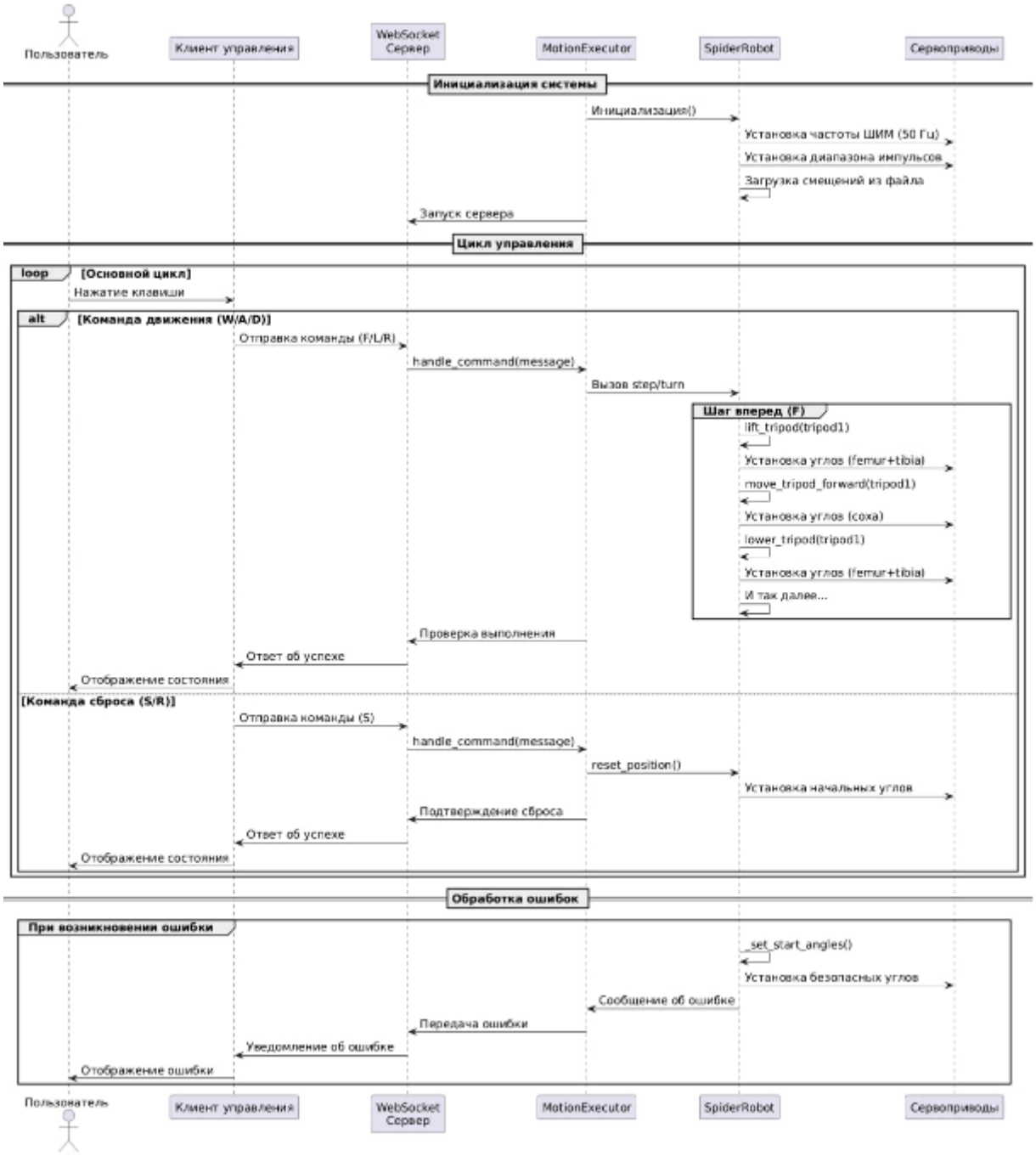

Рис. 4. - Последовательная UML диаграмма походки робота

Система компьютерного зрения и навигации

Автономная работа инсектоморфного робота в сложных и динамичных средах невозможна без развитой системы восприятия и навигации [4]. Разработанная система компьютерного зрения и навигации предназначена для создания точных карт окружающей местности, локализации робота на

этих картах и планирования безопасных траекторий движения. Система построена на базе Robot Operating System 2 (ROS2), что обеспечивает модульность и широкие возможности интеграции различных компонентов.

Архитектура и компоненты системы зрения и навигации

Аппаратной основой сенсорной подсистемы являются 2D LiDAR (RP Lidar) и RGB-D камера (Kinect 360). RP Lidar предоставляет данные для 2D-сканирования, используемые в построении карт стоимости (costmaps) и для одометрии. Kinect 360 обеспечивает получение RGB изображений и карт глубины, которые являются ключевыми для алгоритмов 3D SLAM (Simultaneous Localization and Mapping) и построения трехмерных карт (OctoMap). Опционально может использоваться инерциальный измерительный модуль (IMU) для коррекции ориентации робота. Внешний вид робота представлен на рисунке 1.

Программный комплекс включает несколько ключевых компонентов ROS2. RTABMap (Real-Time Appearance-Based Mapping): Используется в качестве основного SLAM-алгоритма. Его выбор обусловлен не только способностью обрабатывать данные от различных сенсоров (LiDAR, RGB-D камеры, IMU), но и тесной интеграцией с экосистемой ROS2, что позволяет легко подключать его к стандартным навигационным стекам, таким как Nav2. RTABMap публикует построенные 3D-карты (часто в формате OctoMap для эффективного представления занятого и свободного пространства), текущую оценку положения робота (одометрию и локализацию на карте) и другую служебную информацию через стандартизированные топики ROS2. Это позволяет другим узлам, ответственным за планирование маршрута (глобальный и локальный планировщики) и управление движением, использовать актуальные и точные данные об окружении и положении робота. Механизм обнаружения замыкания циклов (loop closure) в RTABMap, корректирующий накопленную

ошибку одометрии, играет жизненно важную роль в обеспечении долгосрочной точности карты и, следовательно, надежности всей навигационной системы. Пример визуализации данных, обрабатываемых и генерируемых RTABMap, можно наблюдать на рисунке 5, что подчеркивает его эффективность в создании детальных 3D-моделей окружения.

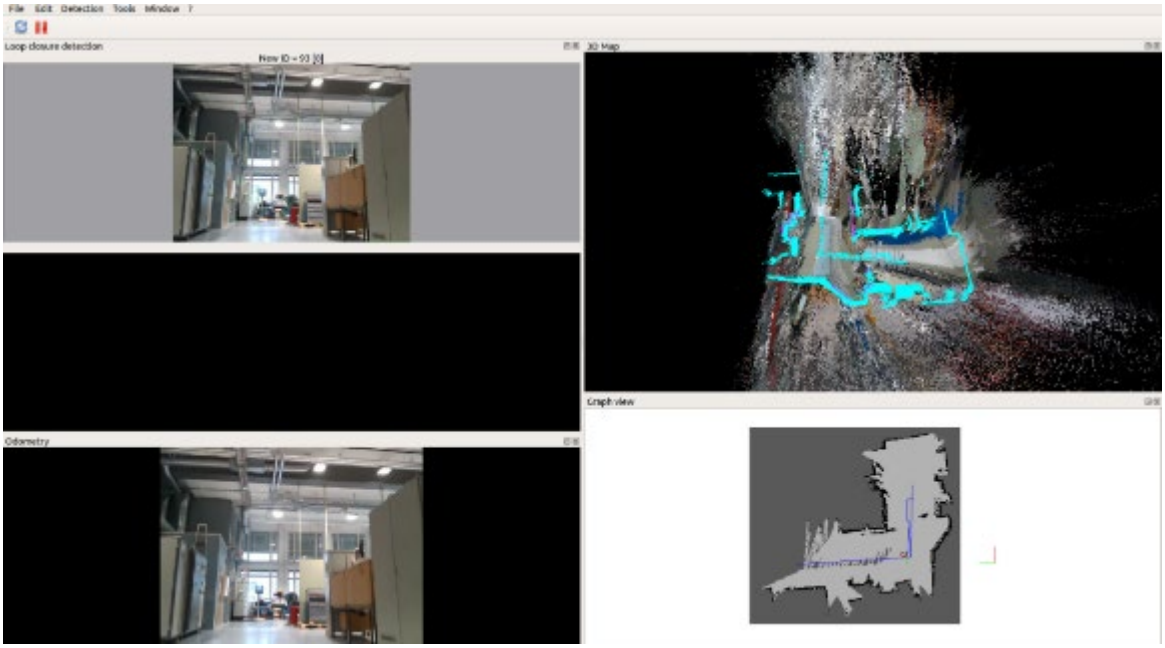

Рис. 5. - Визуализация RTABMap

OctoMap: Применяется для генерации трехмерных воксельных карт на основе облаков точек, полученных от Kinect (через RTABMap). Эти карты используются для навигации и избегания трехмерных препятствий [6]. Визуализация OctoMap для тестового окружения в Gazebo изображен на рисунке ниже.

NAV2: Стандартный навигационный стек ROS2, который используется для планирования пути и управления движением. NAV2 включает глобальный планировщик (например, A* или Dijkstra) для построения маршрута на статической карте и локальный планировщик (например, Timed Elastic Band (TEB) или Dynamic Window Approach (DWA)) для динамического обхода препятствий и следования по глобальному пути.

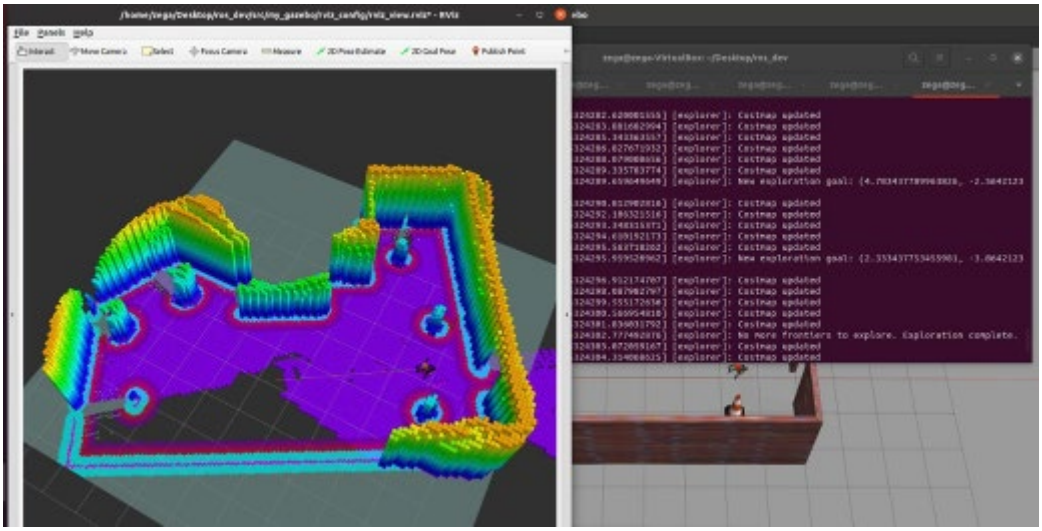

Рис. 6. - Визуализация симуляционного окружения с помощью OctoMap

Драйверы сенсоров и обработка данных: Специализированные ROS2-ноды для чтения данных с LiDAR (/scan) и Kinect (/depth/image, /rgb/image), а также для их предварительной обработки (например, фильтрация облаков точек с помощью pcl_ros) и синхронизации. Важным элементом является корректная настройка TF-трансформаций (transform frames) для связи положений различных сенсоров с основной системой координат робота (base_link).

В результате разработки была получена следующая схема компонентов модуля компьютерного зрения, изображенная на рисунке 7.

Симуляционная среда Gazebo: Используется для тестирования и отладки алгоритмов в виртуальной среде перед развертыванием на реальном роботе. Gazebo позволяет моделировать робота (с использованием URDF-описания), его сенсоры и различные окружения. Созданное окружение видно на рисунке 8.

Визуализация RVIZ2: Инструмент для отображения сенсорных данных, карт, траекторий планировщика и других состояний системы в реальном времени.

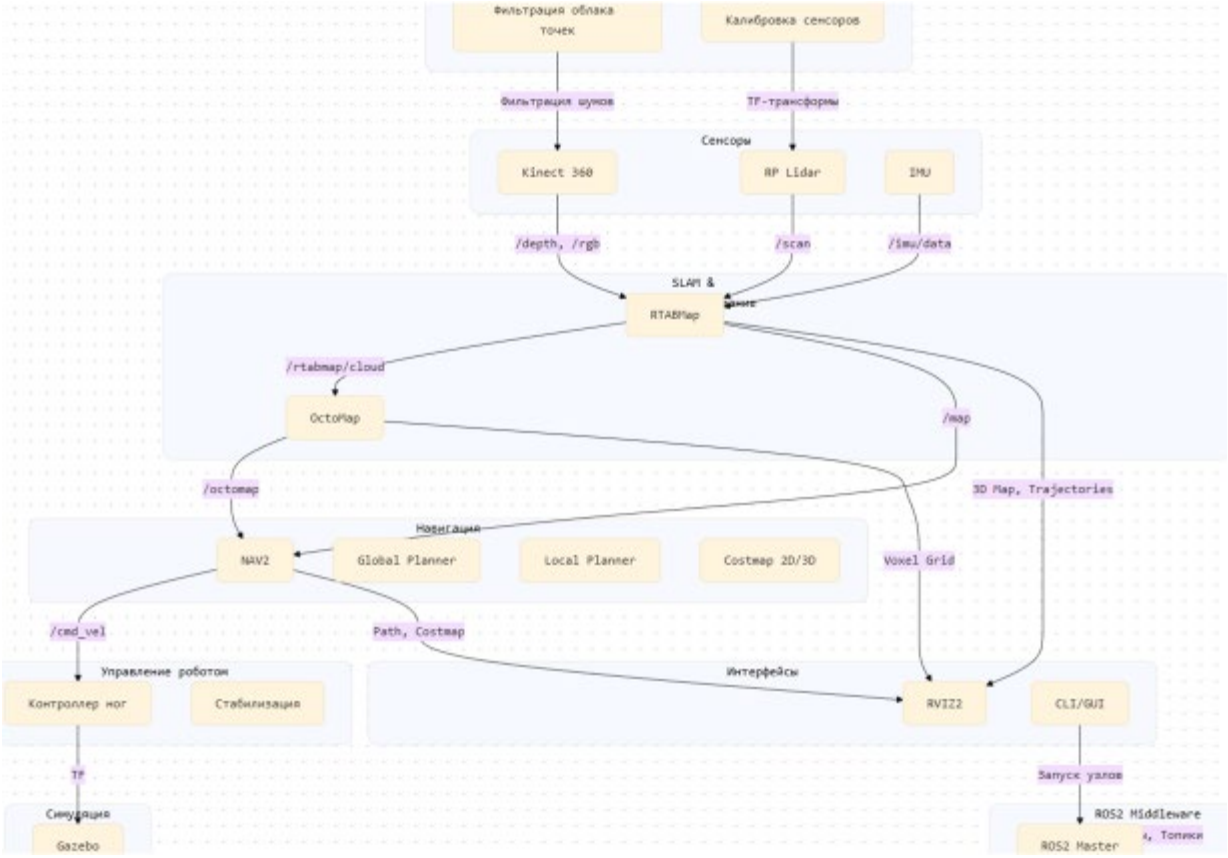

Рис. 7. - Архитектура модуля компьютерного зрения на базе ROS2

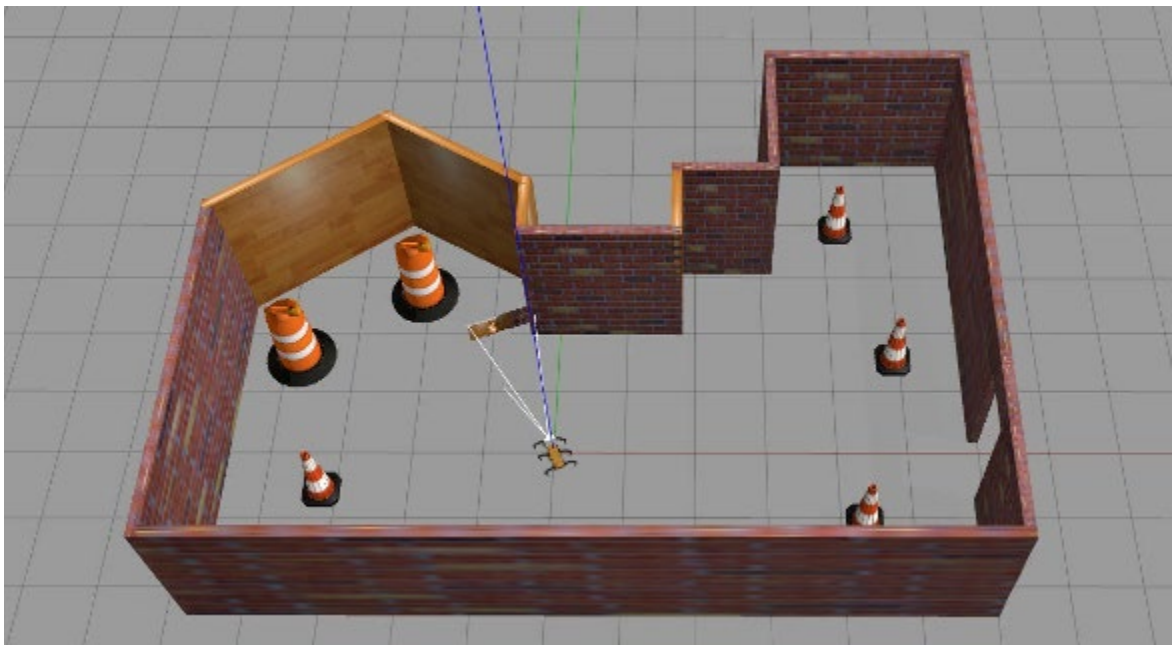

Рис. 8. - Симуляционная среда Gazebo

Функциональные возможности системы зрения и навигации

Процесс работы системы можно разделить на несколько этапов:

1. Сбор и обработка сенсорных данных: Данные с LiDAR и Kinect поступают в соответствующие ROS2-топики. Для Kinect реализован компонент KinectRosComponent (на C++), который взаимодействует с датчиком через библиотеку Freenect, публикует RGB-изображения и изображения глубины, а также информацию о калибровке камеры (camera_info).

```
// Фрагмент KinectRosComponent::timer_callback()
auto header = std_msgs::msg::Header();
header.frame_id = "kinect_depth"; // Имя фрейма для Kinect
header.stamp = now();

if (_depth_flag) { // _depth_flag используется для синхронизации
  auto msg = cv_bridge::CvImage(header, "16UC1", _depth_image).toImageMsg();
  depth_pub_.publish(*msg, depth_info_); // Публикация изображения глубины и информации о камере
  _depth_flag = false;
}
```

Изображения глубины затем преобразуются в облака точек (sensor_msgs/PointCloud2) с помощью стандартного ROS2-компонента depth_image_proc. Этот компонент использует данные калибровки для точного преобразования. Пример параметров калибровки (calibration_depth.yaml):

```
image_width: 640
image_height: 480
camera_name: kinect_depth
```

distortion_model: plumb_bob

D: [0.1, -0.2, 0.001, -0.003, 0.0] # Коэффициенты дисторсии

K: [525.0, 0.0, 320.0, 0.0, 525.0, 240.0, 0.0, 0.0, 1.0] # Матрица камеры

R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0] # Матрица ректификации

P: [525.0, 0.0, 320.0, 0.0, 0.0, 525.0, 240.0, 0.0, 0.0, 0.0, 1.0, 0.0] #

2. SLAM и картографирование: RTABMap использует обработанные данные (сканы LiDAR, RGB-D данные, одометрию от /odom) для построения 3D-карты и одновременной локализации робота. Математически, графовая оптимизация в RTABMap минимизирует ошибку между предсказанными и реальными измерениями сенсоров, представляя позы робота как вершины графа, а ограничения между позами (измерения) как ребра. Целевая функция выглядит как: $\underset{x}{\operatorname{argmin}} \sum_i e_i^T \Omega_i e_i$ – вектор ошибки измерения для *i*-го ограничения, а Ω_i – информационная матрица. Для регистрации облаков точек используется алгоритм ICP (Iterative Closest Point) [7]. Построенная RTABMap карта может выводиться как 2D сетка занятости (/map), так и 3D облако точек или сетка (/rtabmap/grid_map). На основе облаков точек от Kinect (например, топик /rtabmap/cloud или напрямую /kinect/points) генерируется OctoMap (/octomap), представляющая собой воксельную 3D-карту, эффективную для обнаружения препятствий в трехмерном пространстве.

3. Локализация: Для уточнения положения робота на уже построенной карте используется модуль AMCL (Adaptive Monte Carlo Localization). AMCL – это вероятностный алгоритм локализации, использующий фильтр частиц [8]. Параметры AMCL, такие как количество частиц (max_particles: 2000) и модель лазерного дальномера (laser_model_type: "likelihood_field"), настраиваются для достижения баланса между точностью и вычислительной нагрузкой.

4. Навигация (NAV2): NAV2 использует построенную 2D-карту (/map от RTABMap) и данные OctoMap для планирования пути [9]. Система карт стоимости (Costmaps) играет центральную роль.

Глобальная карта стоимости (global_costmap): Строится на основе статической карты от SLAM и используется глобальным планировщиком.

Локальная карта стоимости (local_costmap): Обновляется с высокой частотой (например, 10 Гц) вокруг робота, учитывая данные с сенсоров для обнаружения динамических препятствий. Важными слоями являются voxel_layer (для 3D препятствий от Kinect) и inflation_layer (расширяет границы препятствий для безопасного обхода, inflation_radius: 0.3 м).

Глобальный планировщик (например, NavFn, использующий Dijkstra или A*) прокладывает маршрут от текущего положения робота до цели. Алгоритм A* использует эвристическую функцию $h(n)$ и стоимость пути от начальной точки $g(n)$ для выбора оптимального пути: $f(n) = g(n) + h(n)$.

Локальный планировщик (например, DWB - Dynamic Window Approach, или TEB - Timed Elastic Band) генерирует управляющие команды (/cmd_vel), стремясь следовать глобальному пути и избегая при этом локальных препятствий. DWB, например, оценивает множество возможных траекторий в пределах динамических ограничений робота, используя набор критериев (critics), таких как RotateToGoal, PathAlign (следование пути, вес PathAlign.scale: 32.0), Obstacle (избегание препятствий, вес Obstacle.scale: 0.02). Пример параметров для DWB:

FollowPath:
  plugin: "dwb_core::DWBLocalPlanner"
  max_vel_x: 0.26   # м/с
  acc_lim_theta: 3.2   # рад/с²

NAV2 также включает механизмы восстановления (recovery behaviors), такие как вращение на месте (spin) или откат назад (backup), если робот застрял или планирование пути не удалось.

5. Интеграция с управлением движением: Управляющие команды скорости (geometry_msgs/Twist в топике /cmd_vel), генерируемые локальным планировщиком NAV2, передаются на модуль управления походкой. Модуль управления походкой преобразует эти команды в конкретные действия сервоприводов, реализуя движение вперед, назад или повороты. Например, угловая скорость msg.angular.z может быть интерпретирована как команда поворота, а линейная скорость msg.linear.x – как команда движения вперед.

Потоки данных в системе организованы следующим образом:

LiDAR → RTABMap → /map → NAV2 (для 2D-навигации).

Kinect → depth_image_proc → /points → RTABMap / OctoMap → NAV2 Costmap (для учета 3D-препятствий).

NAV2 → /cmd_vel → Контроллер походки робота.

Такая архитектура позволяет роботу автономно исследовать неизвестные пространства, строить их карты и перемещаться к заданным целям, избегая препятствий. Точность позиционирования системы, согласно тестам, может достигать ±1.8−2 см, с поддержкой обработки до 10 динамических объектов в секунду.

Заключение и перспективы развития

Представленная интегрированная система управления походкой и компьютерного зрения для шестиногого инсектоморфного робота демонстрирует комплексный подход к созданию автономных мобильных платформ. Модуль управления походкой обеспечивает стабильное и адаптивное перемещение робота благодаря точной калибровке сервоприводов и реализации триподного алгоритма движения, а также предусматривает гибкую интеграцию с системой навигации через

WebSocket-интерфейс. Система компьютерного зрения и навигации, построенная на базе ROS2, RTABMap, NAV2 и OctoMap, позволяет роботу осуществлять 3D-картографирование, точную локализацию и автономное планирование пути в сложных, в том числе динамических, средах с использованием данных от LiDAR и RGB-D камеры.

Ключевыми особенностями разработанного решения являются модульная архитектура, обеспечивающая гибкость и расширяемость, использование современных алгоритмов SLAM и навигации, а также тесная интеграция подсистем восприятия и движения. Практическая значимость подтверждается успешной реализацией запланированных функций и стабильной работой системы как в симулированных условиях (Gazebo), так и на реальной робототехнической платформе с достижением высокой точности позиционирования.

Перспективы дальнейшего развития системы включают интеграцию более продвинутых методов восприятия, таких как семантический SLAM (например, Kimera) для улучшенного понимания роботом окружающей среды и идентификации объектов [10]. Также возможна реализация адаптивных алгоритмов походки, которые бы динамически изменяли параметры движения в зависимости от характеристик поверхности, определяемых системой компьютерного зрения. Оптимизация энергопотребления за счет адаптивного управления активностью сенсоров и вычислительных модулей, а также разработка алгоритмов для группового взаимодействия роботов (swarm intelligence) представляют собой другие важные направления для будущих исследований.

Исследование выполнено за счет гранта Российского научного фонда № 25-21-20073 (rscf.ru/project/25-21-20073/) и гранта администрации Волгоградской области.

Литература

1. Егунов В.А., Королева И.Ю., Типаев Д.В. Алгоритмы движения мобильного робота с построением карты местности в реальном времени // Инженерный вестник Дона. – 2022. – № 4. – URL: ivdon.ru/ru/magazine/archive/n4y2022/7570.

2. Macenski S., Foote T., Gerkey B., Lalonde C., Woodall W. Robot Operating System 2: Design, architecture, and uses in the wild // Science Robotics. – 2022. – Vol. 7, No. 66. – eabm6074. DOI: 10.1126/scirobotics.abm6074.

3. Егунов В.А., Петросян М.К. Разработка инерциальной системы навигации шагающего робота с возможностью визуализации положения робота в пространстве // Известия Волгоградского государственного технического университета. – 2018. – № 5 (215). – С. 106–109.

4. Егунов В.А., Жуков А.П., Потапов М.И. Об управлении манипуляционным механизмом мобильного робота // Известия Волгоградского государственного технического университета. – 2011. – № 11 (84). – С. 49–51.

5. Labbé M., Michaud F. RTAB-Map as an open-source Lidar and visual SLAM library for large-scale and long-term online operation // Journal of Field Robotics. – 2019. – Vol. 36, No. 2. – pp. 416–446. DOI: 10.1002/rob.21831.

6. Hornung A., Wurm K. M., Bennewitz M., Stachniss C., Burgard W. OctoMap: an efficient probabilistic 3D mapping framework based on octrees // Autonomous Robots. – 2013. – Vol. 34, No. 3. – pp. 189–206. DOI: 10.1007/s10514-012-9321-0.

7. Besl P. J., McKay N. D. A method for registration of 3-D shapes // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 1992. – Vol. 14, No. 2. – pp. 239–256. DOI: 10.1109/34.121791.

8. Буданов А.С., Егунов В.А. Использование углов Эйлера в инерциальных навигационных системах // Инженерный вестник Дона. – 2021. – № 7. – URL: ivdon.ru/ru/magazine/archive/n7y2021/7072.

9. Macenski S. et al. The Marathon 2: A Navigation System // 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). – Las Vegas, NV, USA, 2020. – pp. 4413–4420. DOI: 10.1109/IROS45743.2020.9341326.

10. Rosinol A., Abate M., Chang Y., Carlone L. Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping // 2020 IEEE International Conference on Robotics and Automation (ICRA). – Paris, France, 2020. – pp. 1689–1696. DOI: 10.1109/ICRA40945.2020.9197066.

References

1. Egunov V.A., Koroleva I.Yu., Tipaev D.V. Inzhenernyj vestnik Dona 2022. No. 4. URL: ivdon.ru/ru/magazine/archive/n4y2022/7570.

2. Macenski S., Foote T., Gerkey B., Lalonde C., Woodall W. Science Robotics. 2022. Vol. 7, No. 66. eabm6074. DOI: 10.1126/scirobotics.abm6074.

3. Egunov V.A., Petrosyan M.K. Izvestiya Volgogradskogo gosudarstvennogo tekhnicheskogo universiteta. 2018. No. 5 (215). pp. 106–109.

4. Egunov V.A., Zhukov A.P., Potapov M.I. Izvestiya Volgogradskogo gosudarstvennogo tekhnicheskogo universiteta. 2011. No. 11 (84). pp. 49–51.

5. Labbé M., Michaud F. Journal of Field Robotics. 2019. Vol. 36, No. 2. pp. 416–446. DOI: 10.1002/rob.21831.

6. Hornung A., Wurm K. M., Bennewitz M., Stachniss C., Burgard W. Autonomous Robots. 2013. Vol. 34, No. 3. pp. 189–206. DOI: 10.1007/s10514-012-9321-0.

7. Besl P. J., McKay N. D. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1992. Vol. 14, No. 2. pp. 239–256. DOI: 10.1109/34.121791.

8. Budanov A.S., Egunov V.A. Inzhenernyj vestnik Dona. 2021. No. 7. URL: ivdon.ru/ru/magazine/archive/n7y2021/7072.

9. Macenski S. et al. The Marathon 2: A Navigation System. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Las Vegas, NV, USA, 2020. pp. 4413–4420. DOI: 10.1109/IROS45743.2020.9341326.

10. Rosinol A., Abate M., Chang Y., Carlone L. 2020 IEEE International Conference on Robotics and Automation (ICRA). Paris, France, 2020. pp. 1689–1696. DOI: 10.1109/ICRA40945.2020.9197066.

Дата поступления: 29.05.2025

Дата публикации: 25.07.2025