ИТМО

Факультет Систем Управления и Робототехники

Исследование и разработка методов локализации и картирования (SLAM) для мобильной платформы на базе LEGO Mindstorms и ROS 2

Аннотация – Проект охватывает сборку платформы LEGO Mindstorms с ROS 2, разработку одометрии, тестирование ICP и AMCL, внедрение визуального SLAM (ArUco, VO SLAM) и сравнительный анализ результатов.

Ключевые слова: SLAM; ROS 2; LEGO Mindstorms; Одометрия; ICP; AMCL; Visual SLAM; VO SLAM; ArUco-маркеры; rosbag; Картирование; Локализация; LiDAR; Визуальная одометрия

Практику проходили:
студент 3-го курса, 368606
Овчинников Павел Алексеевич
студент 2-го курса, 409502
Сайфуллин Динислам Расилевич

Руководитель:
кандидат технических наук, доцент
Ведяков Алексей Алексеевич

Санкт-Петербург 2025

Содержание

Введение

В этой научно-исследовательской работе рассматривается реализация методов локализации и картирования (SLAM) для мобильной платформы на базе LEGO Mindstorms с использованием ROS 2. Целью работы является создание системы самонавигации робота, способной строить карту окружающей среды и определять свое положение в реальном времени.

Цель исследования: исследовать классические алгоритмы ICP, AMCL, и визуальные методы VO SLAM + ArUco для применения на платформе LEGO Mindstorms, разработать модульную архитектуру ROS 2, обеспечивающую интеграцию одометрии, лидара и камеры Realsense D435.

Задачи исследования:

1. Сборка и настройка мобильной платформы LEGO Mindstorms, реализация базовой одометрии.
2. Сборка датасета с помощью `rosbag`.
3. Реализация и отладка алгоритмов ICP и AMCL.
4. Настройка визуального SLAM на основе ArUco-маркеров, калибровка камеры Realsense.

1. Теория

В рамках научно-исследовательской работы необходимо разобраться в работе трёх алгоритмов: *ICP*, *AMCL* и *VO SLAM*. Разберём, как работает каждый из них с теоретической точки зрения.

1.1. ICP

Алгоритм, использующий результаты сканирования лидара в качестве альтернативы одометрии, т.е. для того, чтобы определить как робот повернулся и сдвинулся.

Входные данные: два соседних измерения лидара — облака точек.

Выходные данные: сдвиг t и поворот R, совмещающие два измерения.

Сам алгоритм на языке псевдокода представлен ниже.

```
Algorithm ICP_Iteration(S_n, S_{n-1}):
    mean_p = mean(S_n)
    mean_q = mean(S_{n-1})
    Cov = 0

    for each point p_i in S_n:
        q_i = argmin over q in S_{n-1} of ||p_i - q||_2
        Cov += (q_i - mean_q)^T * (p_i - mean_p)
    endfor

    [U, Σ, Vh] = SVD(Cov)
    R = Vh^T * U^T
    t = mean_q - R * mean_p

    return t, R
```

Листинг 1: одна итерация алгоритма ICP

1.2. AMCL

Алгоритм, использующий одометрию робота, результаты сканирования лидара и уже построенную карту вида `binary occupancy grid` для локализации робота.

Входные данные: набор частиц на предыдущем шаге, одометрия, измерения лидара, карта окружения

Выходные данные: обновлённый набор частиц на текущем шаге.

```
Algorithm Augmented_MCL(χ_{t-1}, u_t, z_t, m):
    static w_slow, w_fast
    χ_bar_t = χ_t = ∅
    for m = 1 to M:
        x_t^[m] = sample_motion_model(u_t, x_{t-1}^[m])
        w_t^[m] = measurement_model(z_t, x_t^[m], m)
        χ_bar_t = χ_bar_t + <x_t^[m], w_t^[m]>
        w_avg = w_avg + (1/M) * w_t^[m]
    endfor
    w_slow = w_slow + α_slow * (w_avg - w_slow)
    w_fast = w_fast + α_fast * (w_avg - w_fast)
    for m = 1 to M:
        with probability max(0.0, 1.0 - w_fast/w_slow):
            add random pose to χ_t
        else:
            draw i ∈ {1,...,N} with probability ∝ w_t^[i]
            add x_t^[i] to χ_t
        endwith
    endfor
    return χ_t
```

Листинг 2: реализация AMCL

Перед нами адаптивный вариант MCL, который добавляет случайные точки, являющиеся предположением о положении робота. Количество случайных точек определяется путем сравнения краткосрочной и долгосрочной вероятности результатов измерений сенсора.

Как мы видим, для работы алгоритма необходимы `sample_motion_model` и `measurement_model`. Дадим пояснение, за что отвечает каждая из моделей:

- `motion_model` или модель движения — генерирует случайные частицы на известных пустых клетках (помеченные нулями на карте) и использует одометрию робота для того, чтобы соотнести движение частиц с движением робота;
- `measurement_model` или модель измерений — распределяет веса частицам, оценивает их стоимость и занимается их аннигиляцией, основываясь на данных сканирования лидара.

В конечном итоге остаётся некий наиболее весомый набор частиц, близкий к истинному положению робота. Таким образом при движении робота

можно локализовать его положение на уже заранее известной карте, которая была получена, в нашем случае, с помощью *ICP*.

Модель движения описывается так:

```
Algorithm sample_motion_model_velocity(u_t, x_{t-1}):
    v_hat = v + sample(α₁|v| + α₂|ω|)
    ω_hat = ω + sample(α₃|v| + α₄|ω|)
    γ_hat = sample(α₅|v| + α₆|ω|)

    x' = x - (v_hat / ω_hat) * sin(θ) + (v_hat / ω_hat) * sin(θ + ω_hat * Δt)
    y' = y + (v_hat / ω_hat) * cos(θ) - (v_hat / ω_hat) * cos(θ + ω_hat * Δt)
    θ' = θ + ω_hat * Δt + γ_hat * Δt

    return x_i = (x', y', θ')ᵀ
```

Листинг 3: реализация модели движения

Фактически, это алгоритм сэмплинга позиции $x_t = (x', y', \theta')^T$ из предыдущей $x_{t-1} = (x, y, \theta)^T$ и контрольной $u_t = (v, \omega)^T$. Конечная ориентация изменяется с помощью дополнительного случайного члена $\hat{\gamma}$. Переменные от α_1 до α_6 являются параметрами шума движения. Функция `sample(b)` генерирует случайную выборку из распределения с нулевым центром с дисперсией b. Мы используем выборку с нормальным распределением, которая представлена в блоке кода ниже.

```
Algorithm sample_normal_distribution(b):
    return (b / 6) * sum_{i=1 to 12} rand(-1, 1)
```

В качестве модели измерения, которая будет оценивать качество измерения и улучшать алгоритм, будем использовать модель известного соответствия, которая будет использовать ориентиры (landmarks) и уже известную карту и таким образом уточнять положение робота:

```
Algorithm sample_landmark_model_known_correspondence(f_t^i, c_t^i, m):
    j = c_t^i
    γ_hat = rand(0, 2π)
    r_hat = r_t^i + sample(σ_r²)
    φ_hat = φ_t^i + sample(σ_φ²)
    x = m_{j,x} + r_hat * cos(γ_hat)
    y = m_{j,y} + r_hat * sin(γ_hat)
```

```
    θ = γ_hat - π - φ_hat
    return (x, y, θ)ᵀ
```

Листинг 4: реализация модели измерений

1.3. VO SLAM

Алгоритм получает скорректированное изображение с камеры, ее коэффициенты искажения и матрицу K на вход. С помощью пакета `image_proc` выравниваем изображение и отправляем в алгоритм.

Входные данные: скорректированное изображение, коэффициенты искажения, матрицу K, карта окружения

Выходные данные: трансформации ArUco-маркеров.

```
Algorithm Visual_SLAM(frame, K, distCoeffs, m₀):
    gray = to_grayscale(frame)

    corners_list, ids = detectMarkers(gray, marker_size, camera_matrix,
dist_coeffs)
    rvecs, tvecs = estimatePose(corners_list, marker_size, camera_matrix,
dist_coeffs)
    detected = {}
    for i = 0 to len(ids)-1:
        id = ids[i]
        R = Rodrigues(rvec[i])
        T_cam_m = eye(4)
        T_cam_m[0:3, 0:3] = R
        T_cam_m[0:3, 3] = tvec[i]
        detected[id] = T_cam_m
    endfor

    updateMap(detected)
```

Листинг 5: реализация алгоритма визуального SLAM

Для ускорения и упрощения детектирования маркеров исходное цветное изображение конвертируется в градации серого. На полученном «сером» кадре вызывается `detectMarkers`, которая возвращает списки углов и соответствующих им уникальных ID маркеров. По найденным углам и известному физическому размеру маркеров с помощью `estimatePose` вычисляются векторы вращения `rvecs` и сдвига `tvecs` относительно камеры. Для каждого маркера собирается однородная матрица `T_cam_m`,

которая однозначно задаёт положение и ориентацию маркера в системе координат камеры. Собранный словарь `detected` (где ключ — ID маркера, значение — его матрица трансформации) передаётся функции `updateMap`, которая интегрирует новые или скорректированные положения маркеров в глобальную карту.

```
Algorithm updateMap(detected):
    if map is None:
        first_id = any key from detected
        map[first_id] = detected[first_id]
    else:
        known_ids = intersection(keys(detected), keys(map))
        if known_ids is not None:
            ref = any element of known_ids
            T_cam_ref = detected[ref]
            T_world_ref = map[ref]
            for (id, T_cam_m) in detected.items():
                if id not in map:
                    T_ref_m = inverse(T_cam_ref) @ T_cam_m
                    map[id] = T_world_ref @ T_ref_m
                end if
            end for
        end if
    end if
```

Листинг 6: функция обновления карты с маркерами

Результатом работы будет карта с положением ArUco-маркеров в пространстве.

2. Практика

2.1. День 1 (16.06)

В течение первого дня необходимо настроить окружение и изучить существующие решения и литературу касательно ICP и AMCL. Также необходимо собрать данные, которые в дальнейшем будут использоваться для повторного воспроизведения сообщений из топиков.

На ноутбук была поставлена KDE Neon 6.3, основанная на Ubuntu 24.04 LTS, в которой установлен ROS 2 Jazzy. Также поставлены пакеты и ноды ROS, необходимые для дальнейшей работы:

- самописный `lego_driver` для сокет-взаимодействия с Mindstorms EV3;
- `teleop_twist_keyboard` для ручного управления роботом;
- `urg_node2` для работы с лидаром Hokuyo;
- `tf_transformations` для пространственных преобразований в одометрии

Кроме того был инициализирован репозиторий в GitHub и установлен Obsidian для ведения логов практики.

Собрали робота на платформе Lego Mindstorms, сверху робота установили лидар (Рис. 1):

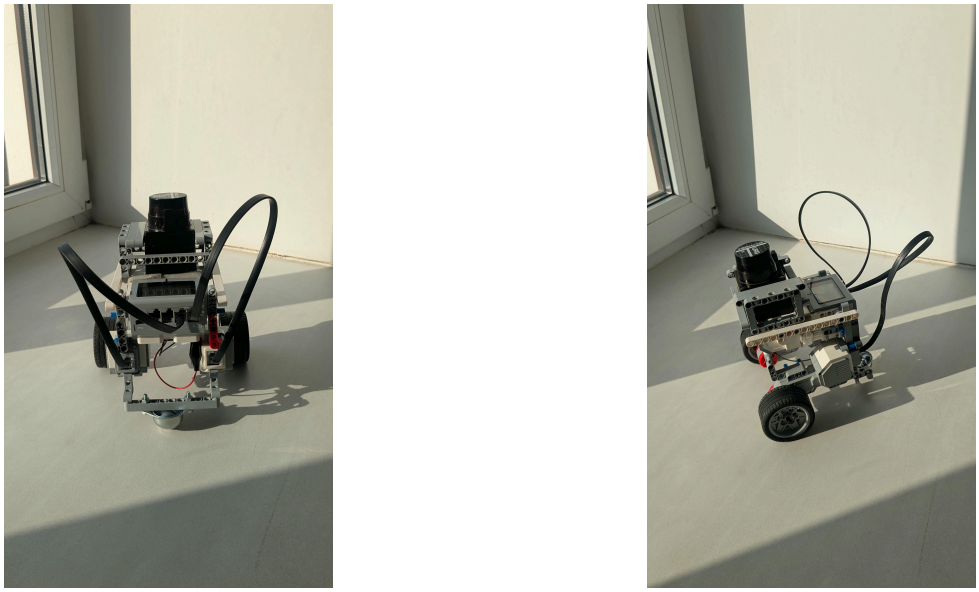

Рис. 1. Собранный робот

Датасет для повторного воспроизведения топиков был собран при помощи `rosbag`. Все собранные «бэги» весят много, поэтому для удобства они были выгружены в отдельную папку в Google Диск.

2.2. День 2 (17.06)

Разработали `ICPNode`, в рамках которой происходит вычисление ICP для облака точек, которые были записаны через `rosbag` в течение первого дня. Алгоритм возвращает требуемые нам смещения dx, dy и поворот yaw одного облака к другому. Алгоритм работал неуверенно, поэтому, чтобы отточить его работу, мы вынесли работу ICP в отдельный Python-модуль и там реализовали сохранение gif-анимации и анимацию итераций в `matplotlib`. Результат представлен ниже.

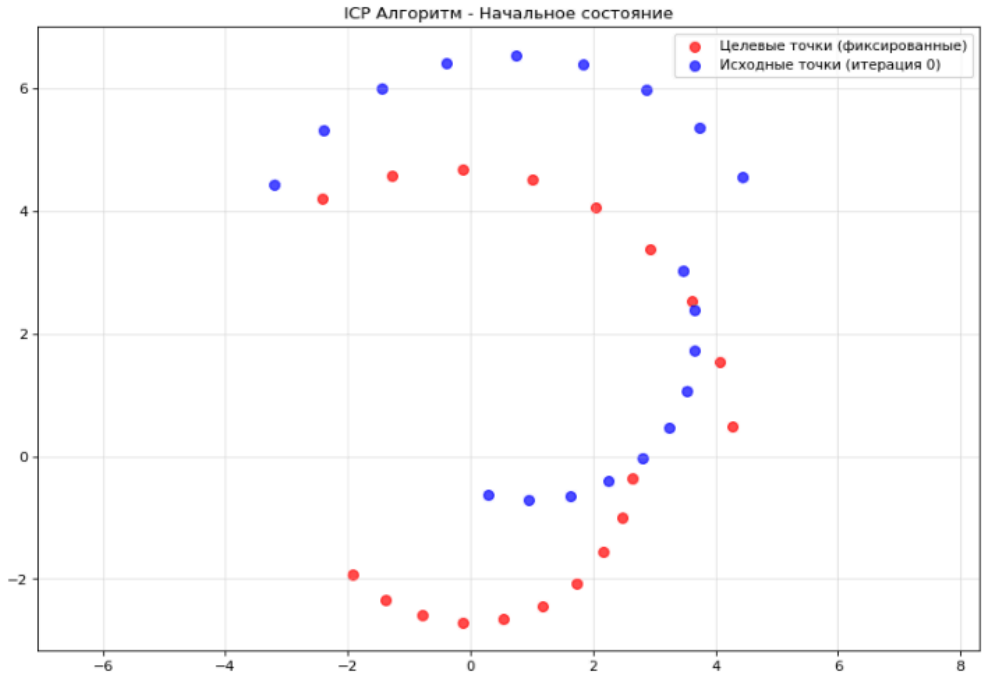

Рис. 2. Это GIF-анимация — посмотреть можно здесь.

Алгоритм хорошо сходился на некотором наборе точек `dst` — для симулирования случайного преобразования и получения `src` был применён сдвиг по $dx = 1$, $dy = 2$ и поворот на $30°$ против часовой стрелки.

2.3. День 3 (18.06)

Как выяснилось, алгоритм хорошо работал в отдельной визуализации, но всё ещё некорректно отрабатывал на данных с лидара в `rviz2`. Это было связано с размерностями и неверным пониманием, что есть `src`, а что `dst`.

- `dst` — облако точек с предыдущего сканирования лидаром;
- `src` — новое облако точек, которое необходимо подвинуть к `dst`.

В случае с алгоритмом, описанным в ICP, `src` является S_n, а `dst` является S_{n-1}. Таким образом, в течение дня отлаживался алгоритм, чтобы он верно работал:

- Перебор всех ближайших точек был заменён на оптимальный `KDTree` из `sklearn.neighbors`;
- Проверили, что `SVD` в `numpy` работает так, как мы ожидаем, и возвращает правильные матрицы.

Начали записи сканов с лидара в `pkl`-файлы, чтобы воспроизводить сканы в `matplotlib` и отлаживать алгоритм там.

2.4. День 4 (19.06)

Параллельно с ICP приступаем к реализации алгоритма MCL. Создадим визуализацию, чтобы в `matplotlib` можно было увидеть ход работы. Первые неудачные результаты можно увидеть ниже.

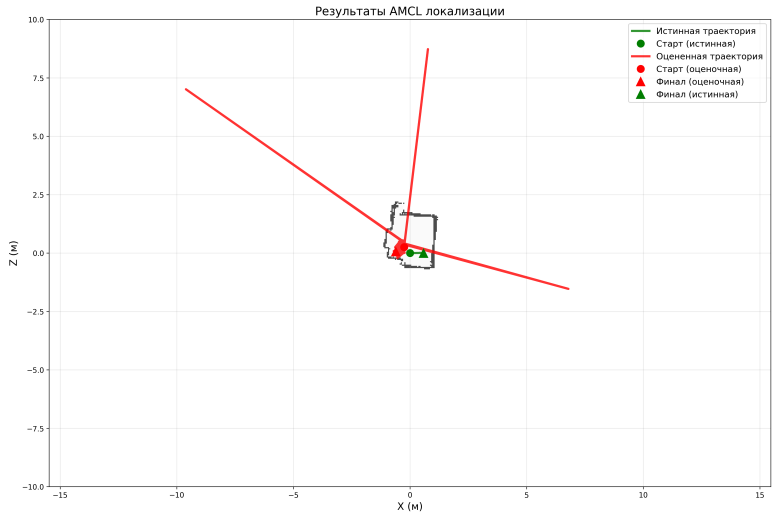

Рис. 3. Результат работы первой версии алгоритма AMCL.

Очевидно, алгоритм требует доработки. Основная проблема заключалась в выставлении ориентиров и скачках ошибки локализации при повороте робота на месте.

Принято решение согласовывать алгоритм ICP совместно с преподавателем. Подбирались различные методы вычисления на определённых шагах — в частности методы вычисления ближайших соседей и фильтрации связей один ко многим, чтобы достичь связей один к одному. Это было необходимо для улучшения работы алгоритма.

2.5. День 5 (20.06)

Пятый день ушёл на настройку камеры Intel RealSense D435 Depth Camera и виртуального окружения для него, в частности установка Python 3.11. Был установлен `pyrealsense2` и была проверена работа камеры с помощью кода из GitHub.

Начали процесс интеграции в ROS2 и столкнулись с проблемой — ROS пытается использовать установленную системно версию Python 3.12. Использование версий Python и виртуальных окружений через `uv`, а также попытки запустить ROS2 на версии 3.11 не увенчались успехом.

Было решено использовать встроенную в ROS реализацию realsense — `realsense2_camera`. Сначала устанавливаем необходимые зависимости для работы:

```
sudo apt-get install ros-jazzy-realsense2-camera ros-jazzy-realsense2-
description
```

И запускаем ноду, работающую с камерой:

```
ros2 launch realsense2_camera rs_launch.py pointcloud.enable:=true
```

Затем запускаем узел с включённой генерацией облака точек. Калибровку камеры проверяли с помощью примеров из репозитория: обнаружили, что без привязки к TF-фреймам маркерные трансформации «гуляли» — поэтому необходимо настроить статический трансформ из /camera_link в /base_link, чтобы последующие подсчёты позиций маркеров в глобальной системе шли корректно.

2.6. День 6 (23.06)

Шестой день ушёл на доработку ноду с алгоритмом ICP. Основной задачей было конвертировать облако точек `PointCloud` от лидара в `OccupancyGrid` для публикации карты. Сгенерировали сетку 20 × 20 м с разрешением 5 см/ячейка: неизвестные ячейки получили -1, свободные – 0, препятствия – 100.

При реализации столкнулись с проблемами — при каждом новом скане карта составлялась целиком, и свободные области постепенно исчезали. Решением стал переход на накопительную стратегию: храним массив

предыдущих значений и обновлем только те ячейки, где действительно появляются новые препятствия или освобождаются участки. Для заполнения свободного пространства была реализована трассировка лучей алгоритмом Брезенхема, а не просто пороговое преобразование глубины: это убрало «пробелы» и дрожание светлых зон при визуализации в RViz.

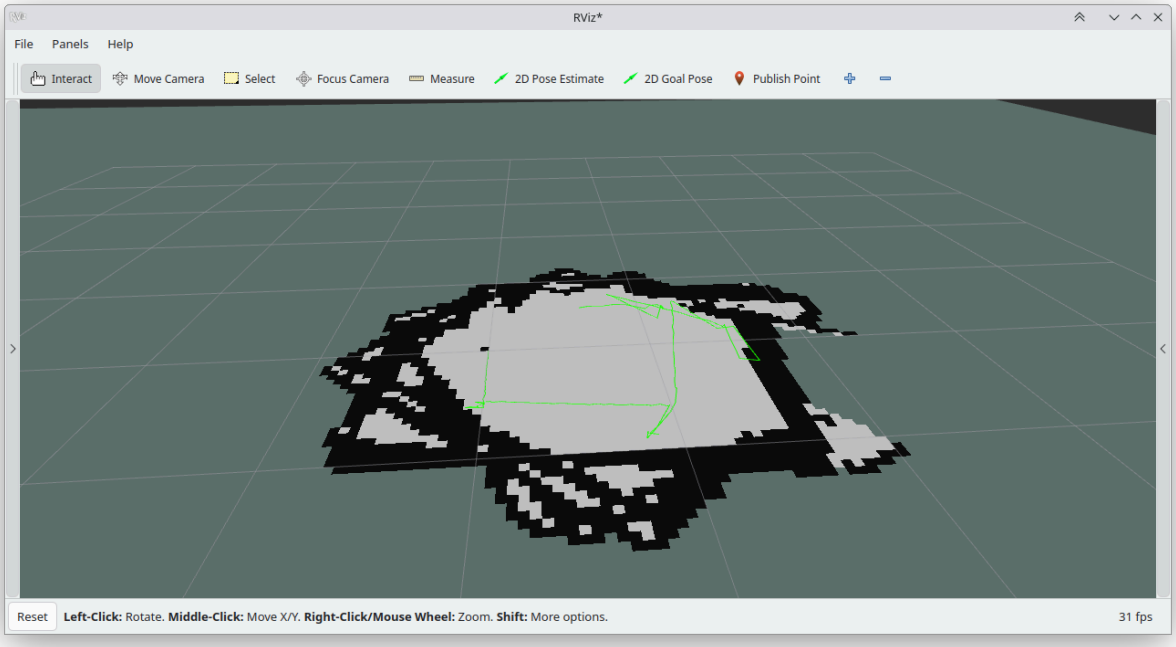

Рис. 4. Результат работы первой версии алгоритма ICP.

Далее были подобраны оптимальные параметры алгоритма для построения карты и отлажен код. Итоговый результат работы построения карты с помощью лидара:

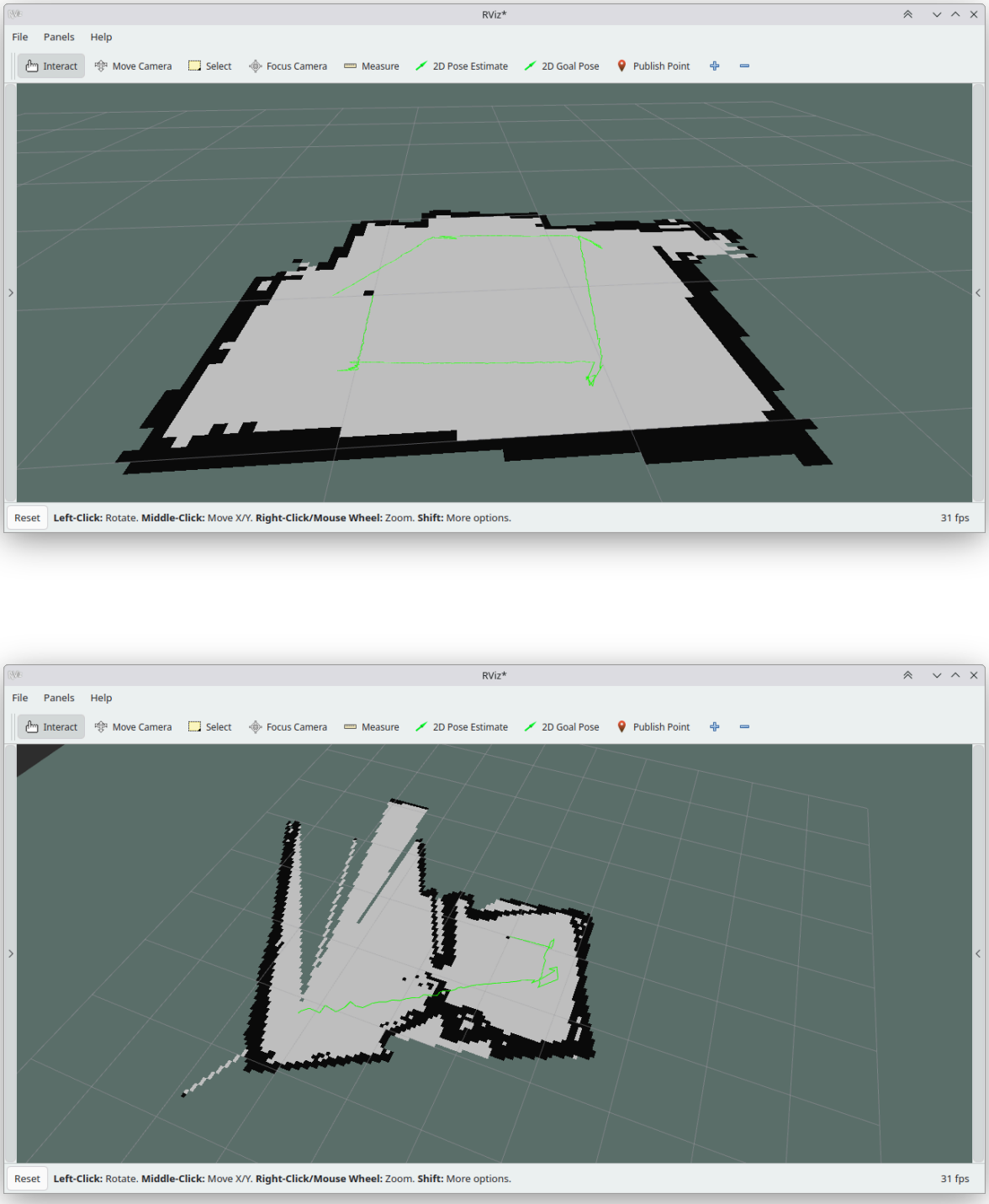

Рис. 5. Результат построения карты.

Под конец дня обсудили с преподавателем работу AMCL с визуализацией в `matplotlib`. Для дальнейшей работы из результата работы ICP были выгружены необходимые данные: карта из топика `/map`, сканы с лидара `/scan` и одометрия `/odom` с корректировкой алгоритма ICP. Карта будет использоваться в `measurement_model`, а одометрия — в `motion_model`.

2.7. День 7 (24.06)

Приступаем к разработке ноды `slam_node` для реализации алгоритма VO SLAM. Необходимо считать изображение из топика `/camera/camera/color/image_rect_color` и матрицы преобразования камеры realsense из `/camera/camera/color/camera_info`, чтобы скорректировать искажение камеры, и с помощью `opencv` определять Aruco-маркеры и находить направления их осей. Результат поиска маркеров в `RViz`.

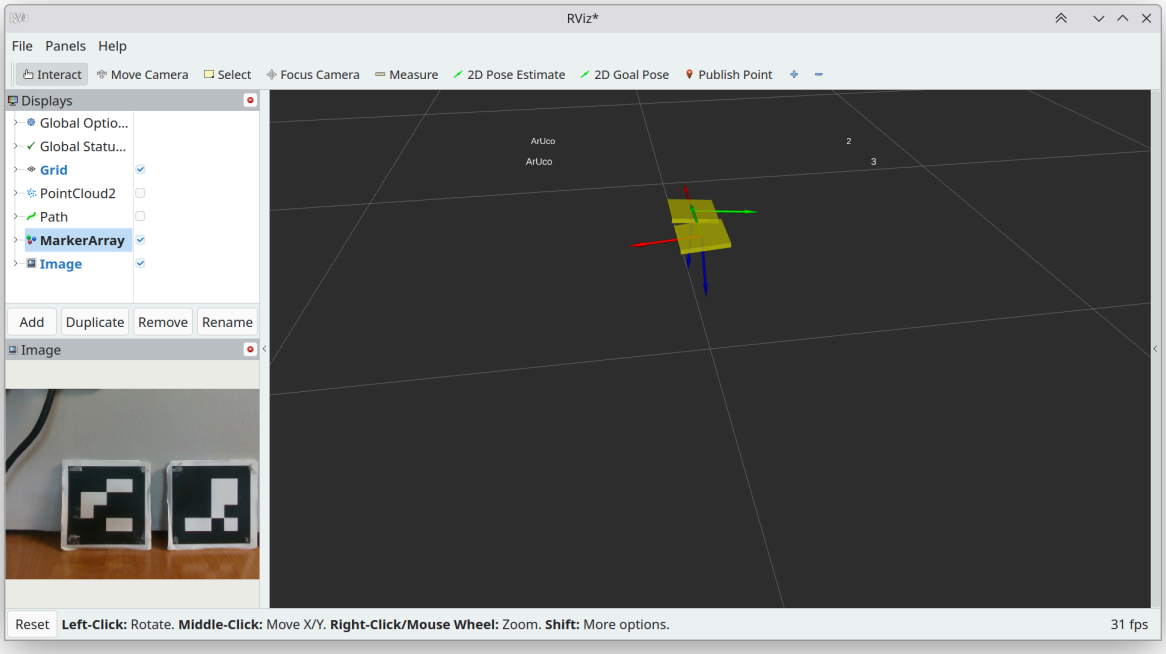

Рис. 6. Визуализация отслеживания Aruco-маркеров в `RViz`.

Также в этот день оттачивалась работа алгоритма AMCL и изучалось, как встроить данные с лидара и одометрию робота в `measurement_model` и `motion_model` соответственно. Приходим к выводу, что существующий алгоритм недостаточно хорош для этого, поэтому появилась необходимость в переписании и использовании нового решения.

Руководитель практики отправил новый документ, содержащий описание работы ресемплинга и взаимодействия с картой, одометрией и сканами с лидара. В том же репозитории с документом имеется шаблон MCL, который был использован для своей реализации.

2.8. День 8 (25.06)

Реализован AMCL на реальных данных — с одометрией, сканами с лидара и картой, сформированной через ICP по тем же сканам с лидарам.

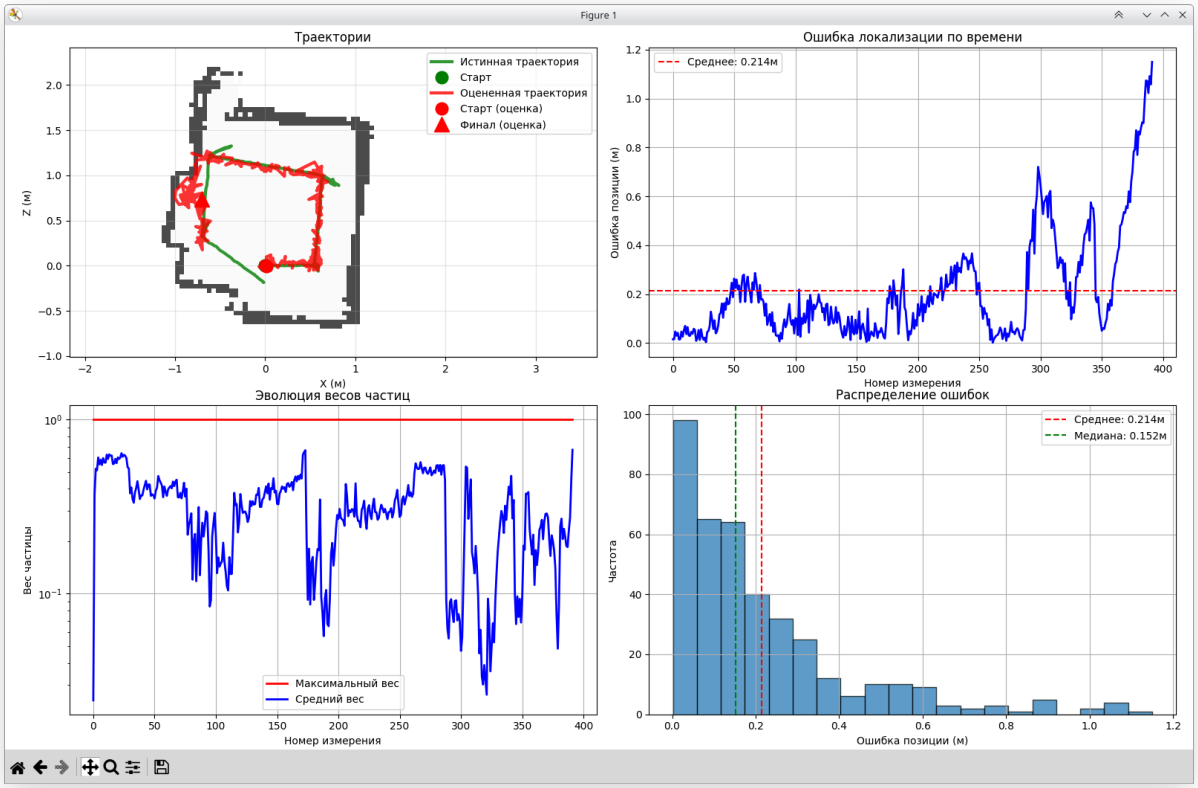

Рис. 7. Визуализация работы AMCL на реальных данных и ошибок вычислений.

Алгоритм работает в целом неплохо до предпоследнего поворота — исправить эту ошибку, увы, так и не удалось. На Рис. 7, что после 280-й позиции робота ошибка локализации растёт, а в эволюции весов наблюдаются скачки.

Также в этот день начинаем тестировать алгоритм визуального SLAM, который использует ArUco-маркеры в качестве ориентиров для построения карты окружающего пространства и локализации камеры в этой карте. Для этого вычисляем положение маркеров в системе координат камеры и переводим их в глобальную координатную систему. В процессе обнаружились две главные трудности:

- смещение карты из-за накопления погрешностей между сменами маркеров;
- локализация камеры работает нестабильно.

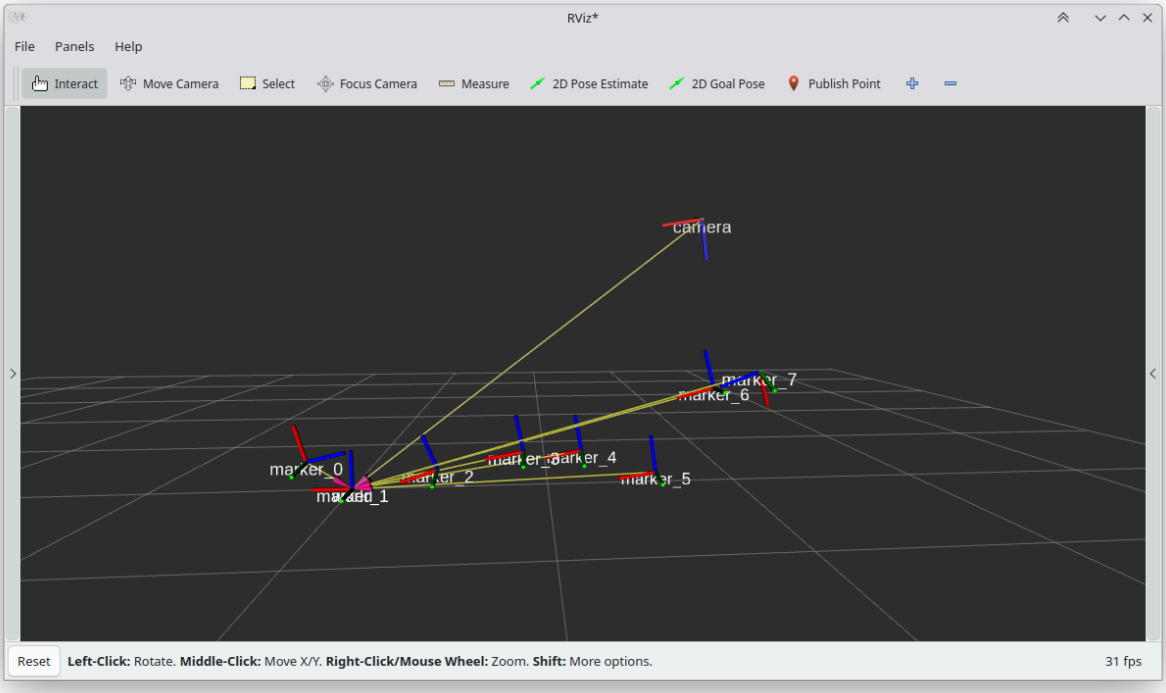

Рис. 8. Расположение ArUco-маркеров в глобальных координатах.

Чтобы частично сгладить эти эффекты, стали хранить позы последних маркеров относительно первого распознанного маркера.

2.9. День 9 (26.06)

Отладили алгоритм: добавили возможность локализоваться камере относительно ArUco-маркеров на полигоне — положение определяется корректно. Далее был подготовлен полигон для съемки - расклеили по периметру комнаты уникальные ArUco-маркеры примерно на одной высоте. В процессе съемок столкнулись с проблемой, что угол обзора камеры слишком мал из-за чего в кадр попадают только 2 маркера и карта строится неверно:

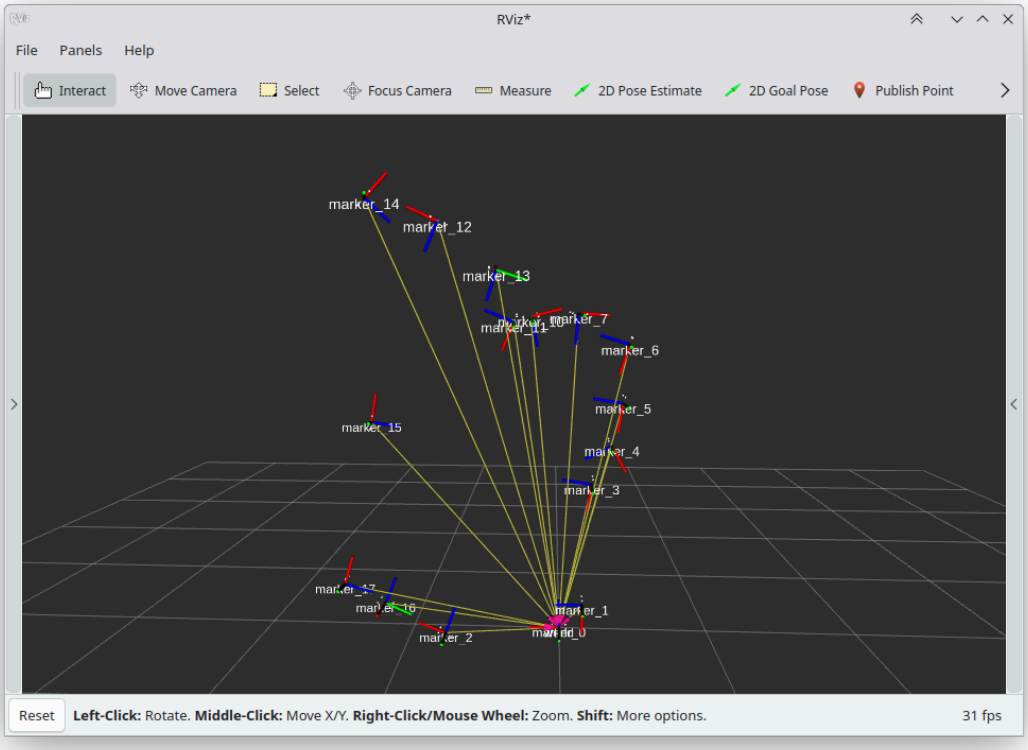

Рис. 9. Первая попытка построения карты алгоритмом VO SLAM.

Пришли к выводу, что необходимо наклеить больше маркеров. Примерное расстояние между ними стало 20-30 см и комната выглядит так:

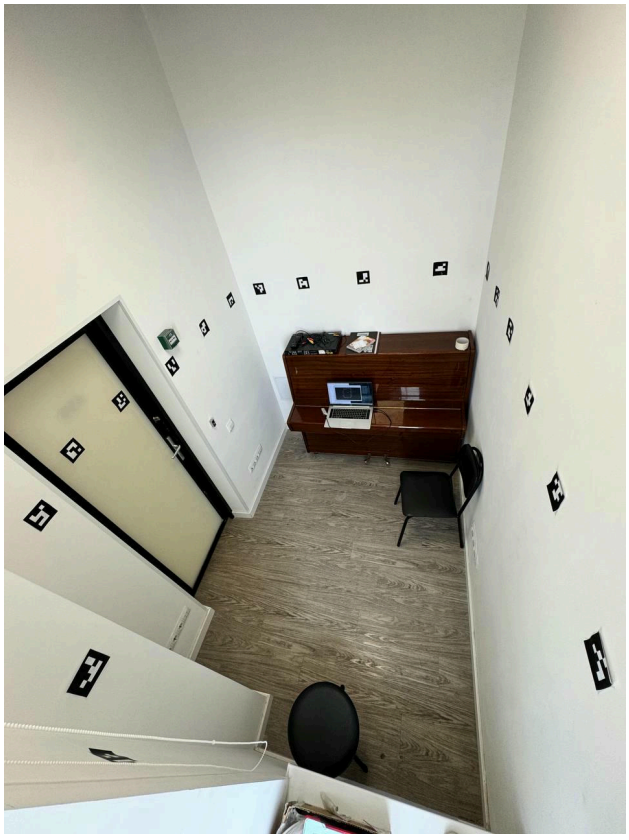

Рис. 10. Комната с ArUco-маркерами.

Запустили алгоритм и разными способами снимали карту. Карта строилась «по спирали» и камера неправильно локализовалась на стыке двух маркеров. Самый оптимальный вариант съёмки: вращение камеры на 360° в центре комнаты — тогда все маркеры оказываются в одной плоскости.

Результат работы алгоритма:

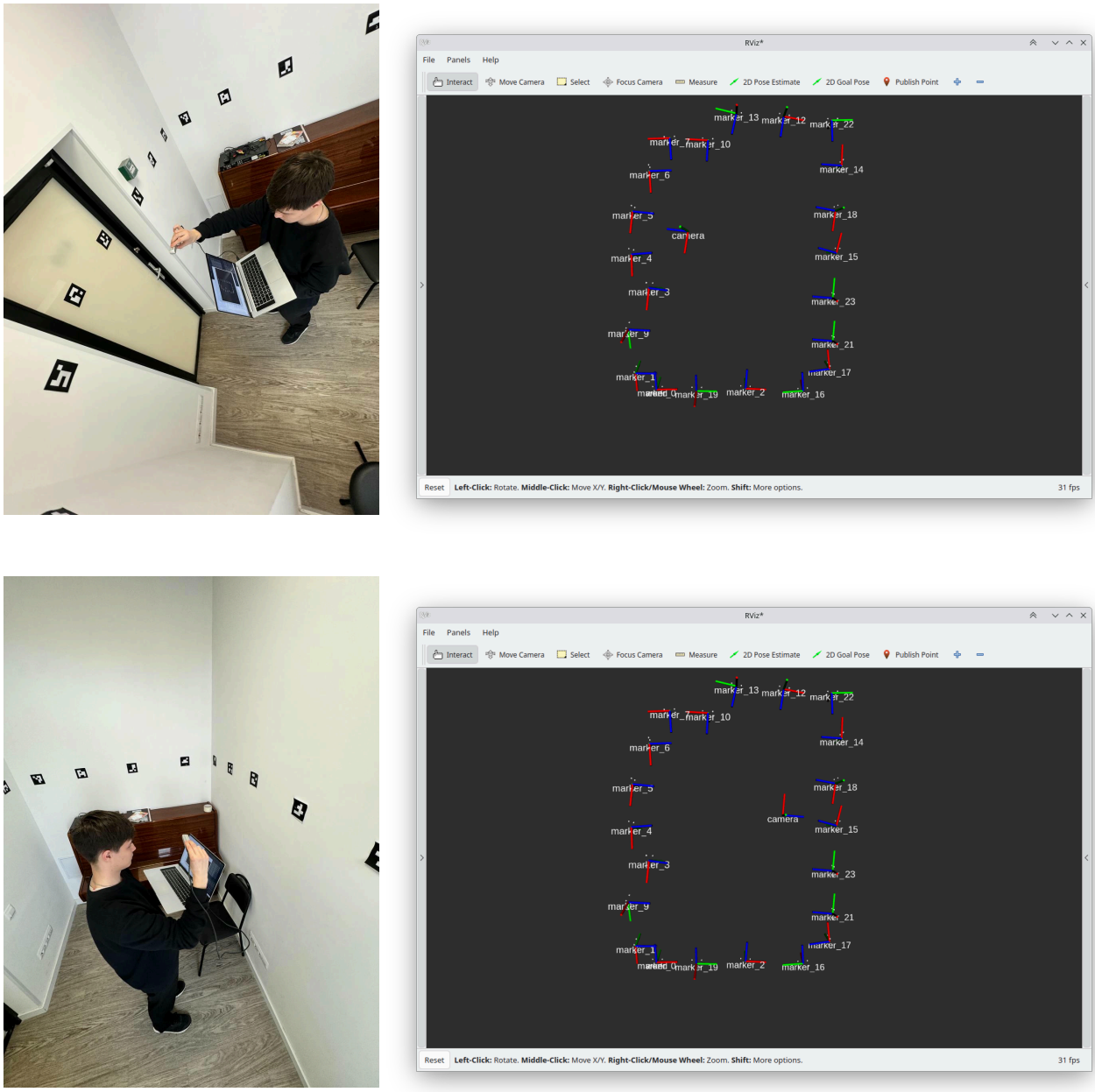

Рис. 11. Результат локализации робота

В результате получилось собрать координаты и направление каждого маркера. Камера отлично локализуется в комнате. Это работоспособный каркас для построения карты окружения и локализации, однако требуется дальнейшая доработка фильтрации соответствий, интеграция IMU для

компенсации дрейфа и более гибкая стратегия ресемплинга частиц в AMCL при слабой плотности препятствий.

Заключение

В результате выполнения работы были реализованы несколько методов SLAM для мобильного робота на базе LEGO Mindstorms и ROS 2.

Основные достижения:

- Успешно собраны и настроены аппаратные и программные компоненты платформы: робот, ПО и ноды;
- Разработали пайплайн для работы с роботом:
  1. Соединение с роботом по SSH
  2. Запуск `lego_driver` локально
  3. Запуск `sockClient` на роботе
  4. Загрузка `urg_node2` для работы лидара
  5. Запуск `teleop_twist_keyboard` для управления роботом с ноутбука
  6. Запуск `rviz2` для визуализации
- Собраны датасеты для повторного воспроизведения сообщений из топиков;
- Реализованы алгоритмы ICP и AMCL, разработаны соответствующие ноды;
- Настроен визуальный SLAM с использованием ArUco-маркеров;
- Проведены эксперименты, предложены направления доработки.

Данная работа создаёт прочную основу для дальнейшего развития гибридных SLAM-систем на мобильных платформах. В перспективе возможно внедрение алгоритмов на платформу, использование нейросетевых методов для извлечения признаков и повышение устойчивости в динамических условиях.

Список используемой литературы

[1] Википедия, «Итеративный алгоритм ближайших точек [Электронный ресурс]». [Онлайн]. Доступно на: https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%B1%D0%BB%D0%B8%D0%B6%D0%B0%D0%B9%D1%88%D0%B8%D1%85_%D1%82%D0%BE%D1%87%D0%B5%D0%BA

[2] LearnOpenCV, «Understanding Iterative Closest Point (ICP) Algorithm with Code [Электронный ресурс]». [Онлайн]. Доступно на: https://learnopencv.com/iterative-closest-point-icp-explained/

[3] Wikipedia, «Monte Carlo localization [Электронный ресурс]». [Онлайн]. Доступно на: https://en.wikipedia.org/wiki/Monte_Carlo_localization

[4] L. Ronghua и H. Bingrong, «CEAMCL [Электронный ресурс]». [Онлайн]. Доступно на: https://arxiv.org/pdf/cs/0411021

[5] S. Das, «Robot localization in a mapped environment using Adaptive Monte Carlo algorithm [Электронный ресурс]». [Онлайн]. Доступно на: https://www.overleaf.com/articles/robot-localization-in-a-mapped-environment-using-adaptive-monte-carlo-algorithm/dxyhwhsyjfbt.pdf

[6] T. Lauttia, «Adaptive Monte Carlo Localization in ROS [Электронный ресурс]». [Онлайн]. Доступно на: https://trepo.tuni.fi/bitstream/handle/10024/134867/TuomasLauttia.pdf

[7] S. He, T. Song, P. Wang, C. Ding, и X. Wu, «An Enhanced Adaptive Monte Carlo Localization for Service Robots in Dynamic and Featureless Environments», *Journal of Intelligent & Robotic Systems*, т. 104, вып. 2, сс. 1–20, 2023, doi: 10.1007/s10846-023-01858-7.

[8] M. V. babu и A. V. Ramprasad, «Adaptive self-localized DQMCL scheme for WSN based on antithetic Markov process», *International Journal of Engineering Trends and Technology*, т. 14, вып. 6, сс. 274–280, 2014, [Онлайн]. Доступно на: https://enggjournals.com/ijet/docs/IJET14-06-02-053.pdf

[9] L. Zhang, R. Zapata, и P. Lépinay, «Self-adaptive Monte Carlo Localization for Mobile Robots Using Range Finders [Электронный ресурс]». [Онлайн]. Доступно на: https://hal-lirmm.ccsd.cnrs.fr/lirmm-00806955/document

[10] S. Thrun, W. Burgard, и D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005. [Онлайн]. Доступно на: https://docs.ufpr.br/~danielsantos/ProbabilisticRobotics.pdf

[11] ROS Wiki, «icp (ROS Fuerte) [Электронный ресурс]». [Онлайн]. Доступно на: https://wiki.ros.org/icp

[12] PRBonn, «KISS-ICP [Электронный ресурс]». [Онлайн]. Доступно на: https://github.com/PRBonn/kiss-icp

[13] Ishfaz, «ROS-Point-Cloud-ICP [Электронный ресурс]». [Онлайн]. Доступно на: https://github.com/Ishfaz/ROS-Point-Cloud-ICP

[14] ROS Wiki, «amcl (ROS Melodic/Noetic) [Электронный ресурс]». [Онлайн]. Доступно на: https://wiki.ros.org/amcl

[15] laygond, «Adaptive Monte Carlo Localization [Электронный ресурс]». [Онлайн]. Доступно на: https://github.com/laygond/Adaptive-Monte-Carlo-Localization

[16] PyPI, «pyrealsense2 [Электронный ресурс]». [Онлайн]. Доступно на: https://pypi.org/project/pyrealsense2/

[17] GitLab, «MCL [Электронный ресурс]». [Онлайн]. Доступно на: https://gitlab.u-angers.fr/cours/mobile_robotic_student/-/blob/master/documents/MCL/MCL.pdf

[18] A. Zhou, T. Funkhouser, M. Kazhdan, J. Chen, H. Dong, и J. Savarese, «Zipper: Fast approximate nearest neighbor search [Электронный ресурс]». [Онлайн]. Доступно на: https://graphics.stanford.edu/papers/zipper/zipper.pdf

[19] G. Cignoni, F. Ganovelli, и R. Scopigno, «RANSAC-ICP: RANSAC Meets ICP [Электронный ресурс]». [Онлайн]. Доступно на: https://vcg.isti.cnr.it/~cignoni/GMP2223/PDF/GP2123_12_RansacICP.pdf