



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, внедрение визуального SLAM (ArUco, VO SLAM) и сравнительный анализ результатов.

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

**Практику проходил:**

студент 2-го курса, 409502

Сайфуллин Динислам Расилевич

**Руководитель:**

кандидат технических наук, доцент

Ведяков Алексей Алексеевич

# Содержание

Введение .....	3
1. Теория .....	4
1.1. ICP .....	4
1.2. VO SLAM .....	4
2. Практика .....	7
2.1. День 1 (16.06) .....	7
2.2. День 2 (17.06) .....	8
2.3. День 3 (18.06) .....	8
2.4. День 4 (19.06) .....	9
2.5. День 5 (20.06) .....	9
2.6. День 6 (23.06) .....	10
2.7. День 7 (24.06) .....	12
2.8. День 8 (25.06) .....	14
2.9. День 9 (26.06) .....	14
Заключение .....	17
Список используемой литературы .....	19

# 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 классический алгоритм ICP, и визуальный метод 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 `rosvbag`.
3. Реализация и отладка алгоритма ICP.
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* 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  $S_n$ :  
         $q_i = \operatorname{argmin}_{q \in S_{\{n-1\}}} ||p_i - q||_2$   
        Cov += ( $q_i - \operatorname{mean}_q$ )T * ( $p_i - \operatorname{mean}_p$ )  
    endfor  
  
    [ $U, \Sigma, V_h$ ] = SVD(Cov)  
     $R = V_h^T * U^T$   
     $t = \operatorname{mean}_q - R * \operatorname{mean}_p$   
  
    return  $t, R$ 
```

Листинг 1: одна итерация алгоритма ICP

## 1.2. 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 2: 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 3: 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. 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):

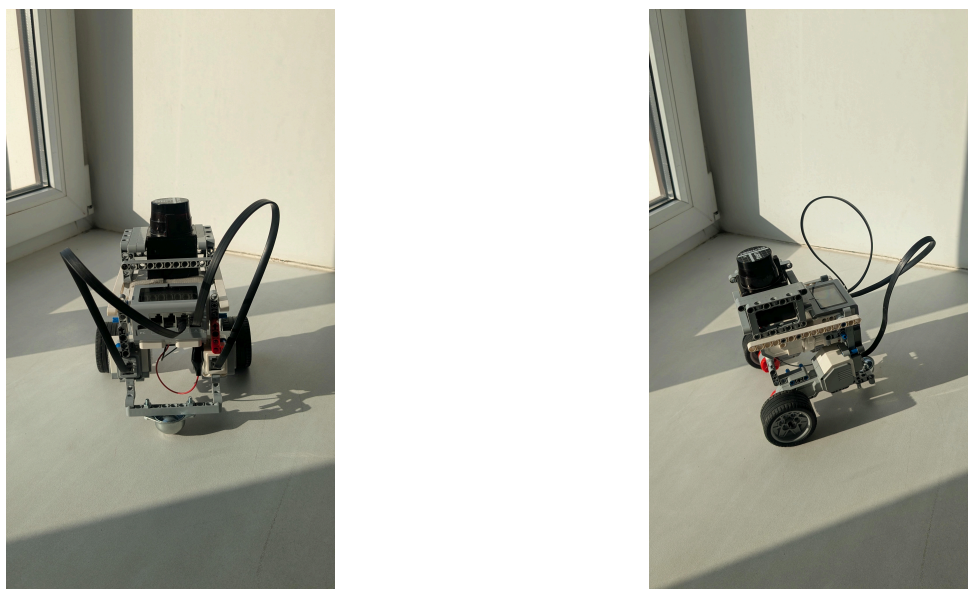


Рис. 1. Собранный робот

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)

Разработана `ICPNode`, в рамках которой происходит вычисление ICP для облака точек, которые были записаны через `rosvbag` 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, работа ICP была вынесена в отдельный Python-модуль и там реализовано сохранение gif-анимации и анимации итераций в `matplotlib`. Результат представлен ниже.

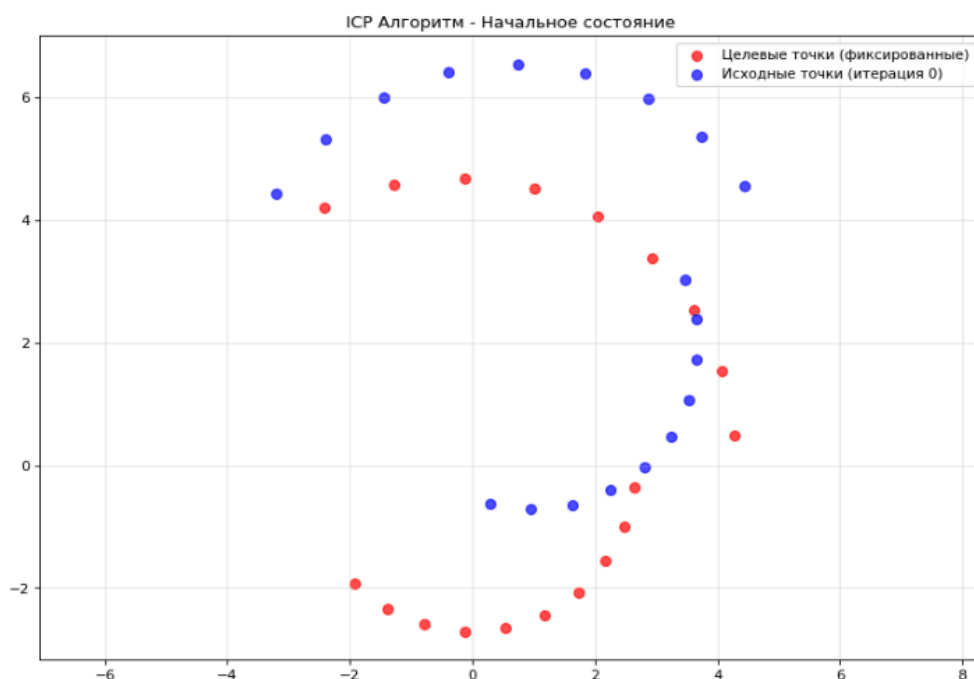


Рис. 2. Это GIF-анимация — посмотреть можно [здесь](#).

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

## 2.3. День 3 (18.06)

Как выяснилось, алгоритм хорошо работал в отдельной визуализации, но всё ещё некорректно обрабатывал 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.

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

## 2.4. День 4 (19.06)

Принято решение согласовывать алгоритм ICP совместно с руководителем практики. 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](#).

В процессе интеграции в ROS2, столкнулся 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 была проверена с помощью примеров из репозитория: оказывается 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. Генерируем сетку  $20 \times 20$  м с разрешением 5 см/ячейка: неизвестные ячейки записываются как -1, свободные — 0, препятствия — 100.

При реализации возникла проблема — при каждом новом скане карта составлялась целиком, и свободные области постепенно исчезали. Решением стал переход на накопительную стратегию: храним массив предыдущих значений и обновляем 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.

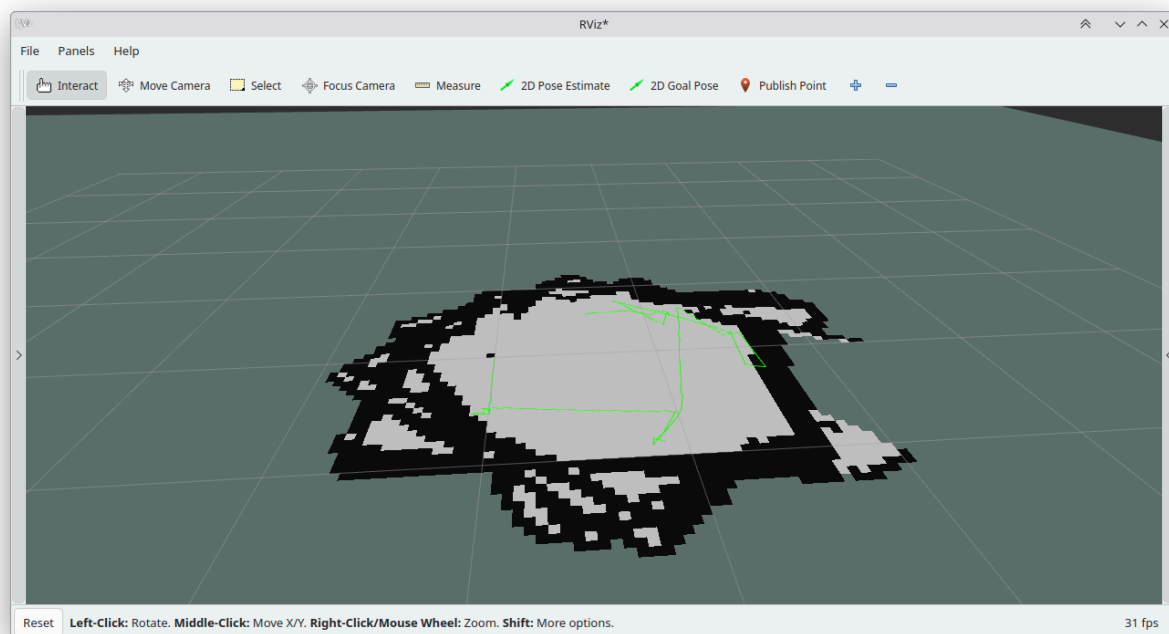


Рис. 3. Результат работы первой версии алгоритма ICP.

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

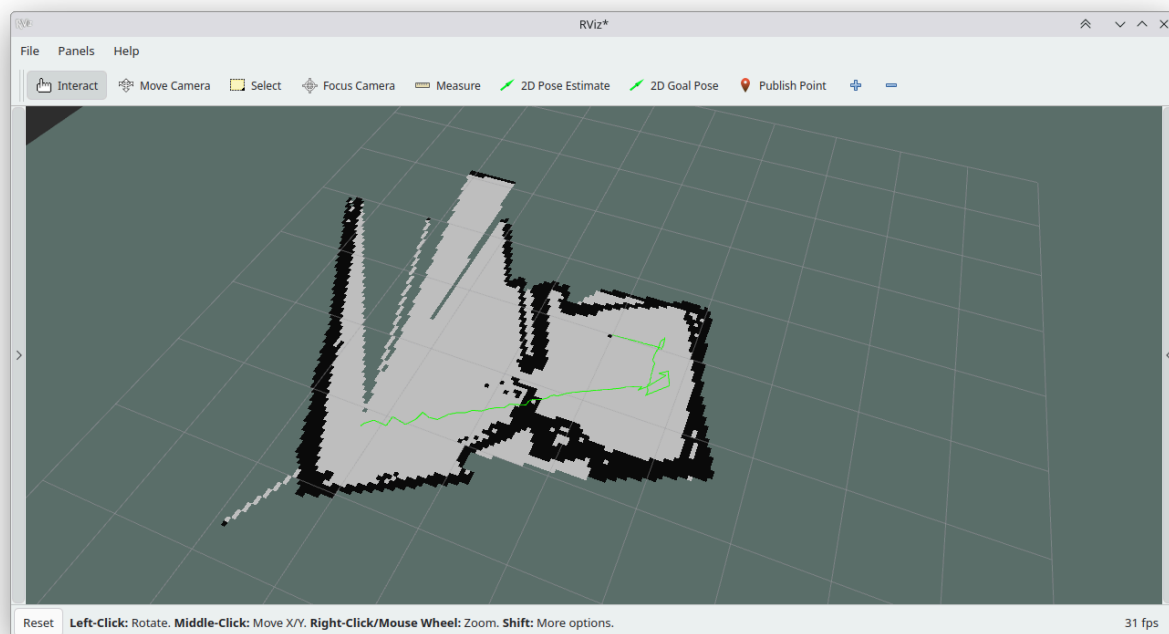
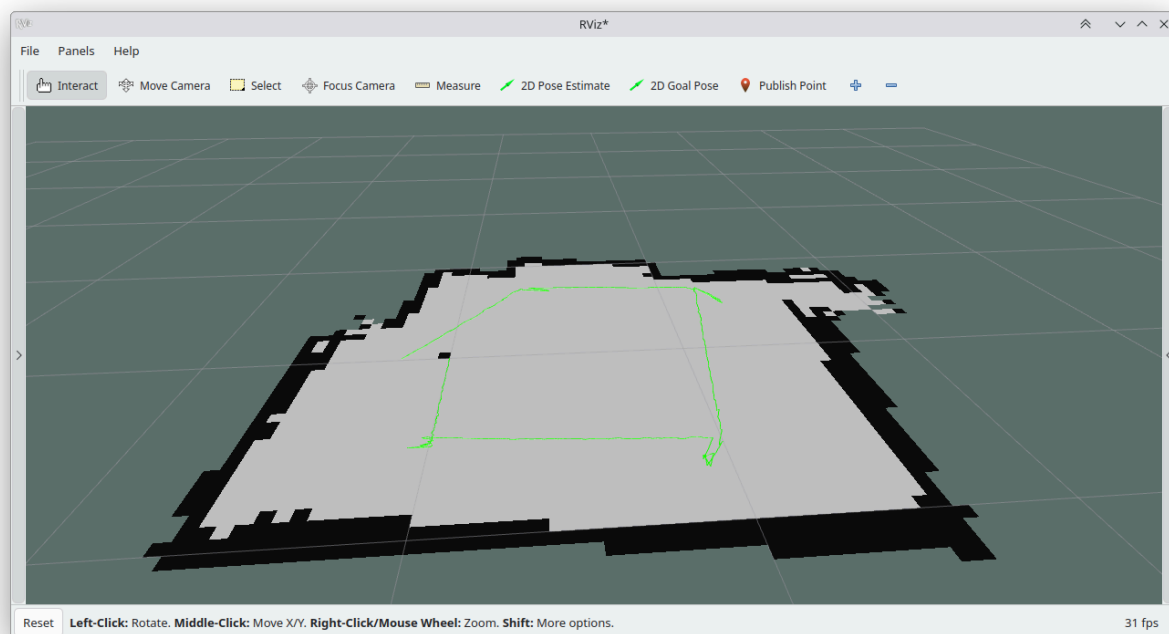


Рис. 4. Результат построения карты.

## 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 скорректи-

ровать искажение камеры, и с помощью `opencv` определять Aruco-маркеры и находить направления их осей. Результат поиска маркеров в `RViz`.

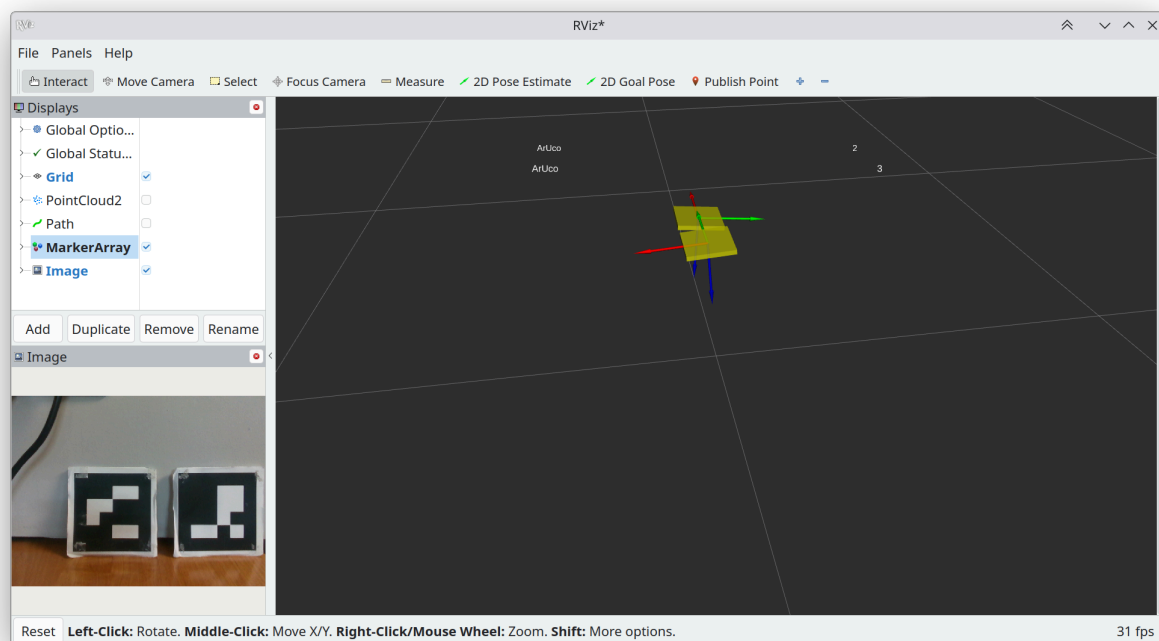


Рис. 5. Визуализация отслеживания Aruco-маркеров в `RViz`.

## 2.8. День 8 (25.06)

В этот день начал тестировать алгоритм визуального SLAM, который использует ArUco-маркеры в качестве ориентиров для построения карты окружающего пространства и локализации камеры в этой карте. Для этого вычисляется положение маркеров в системе координат камеры, после этого они переводятся в глобальную координатную систему. В процессе обнаружались две главные трудности:

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

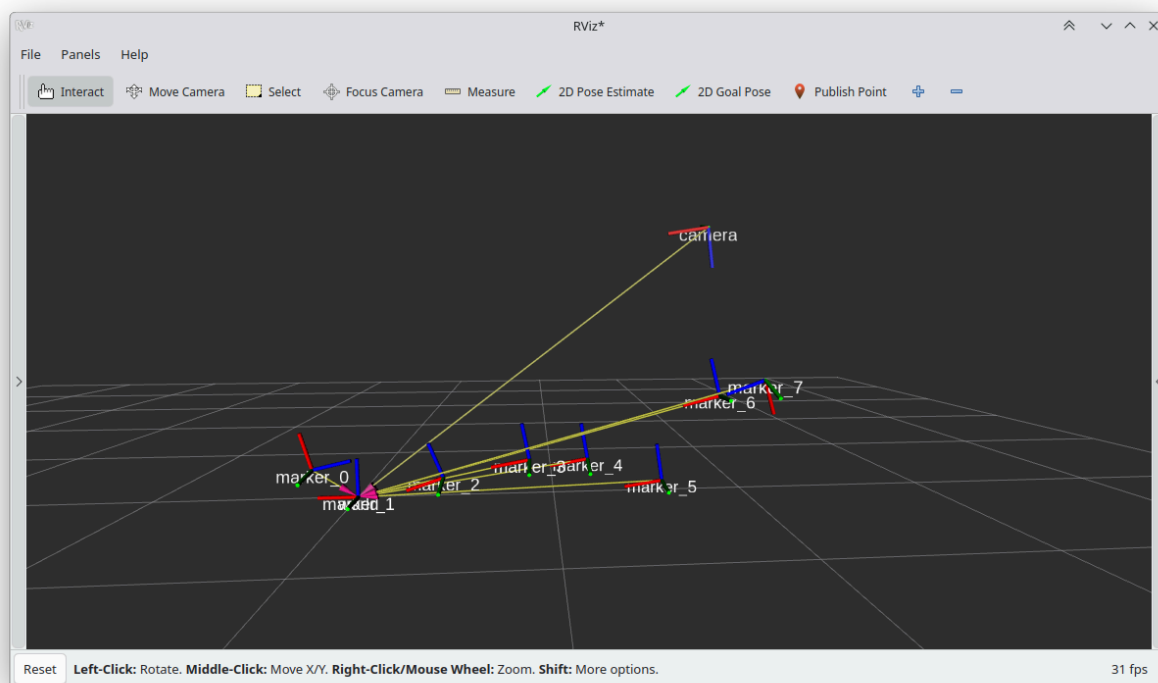


Рис. 6. Расположение ArUco-маркеров в глобальных координатах.

Чтобы частично сгладить эти эффекты, станем 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 слиш-

ком мал из-за чего в кадр попадают только 2 маркера и карта строится неверно:

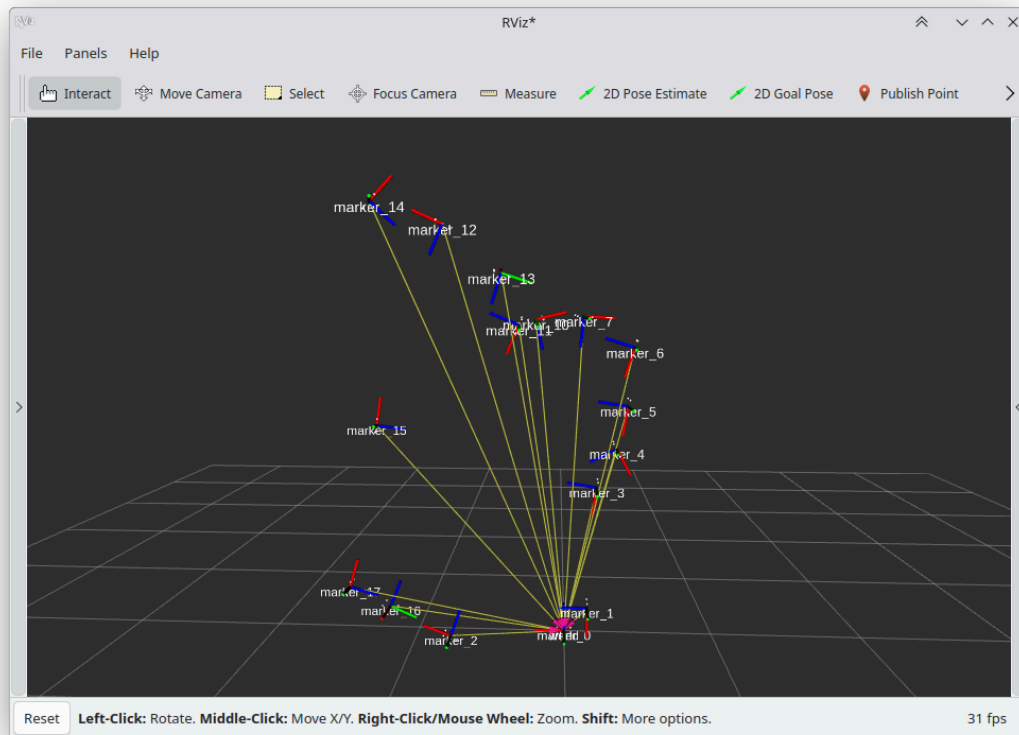


Рис. 7. Первая попытка построения карты алгоритмом VO SLAM.

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

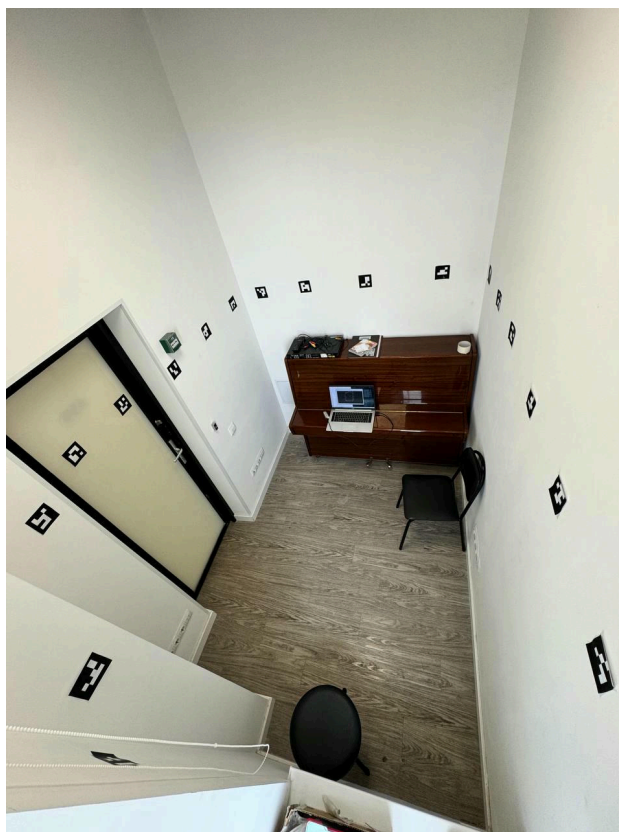
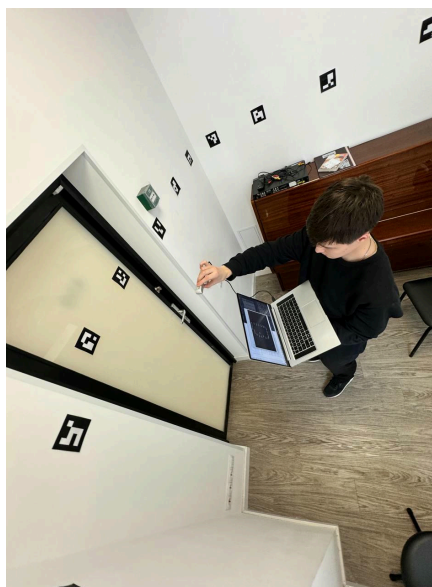


Рис. 8. Комната с ArUco-маркерами.

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

Результат работы алгоритма:





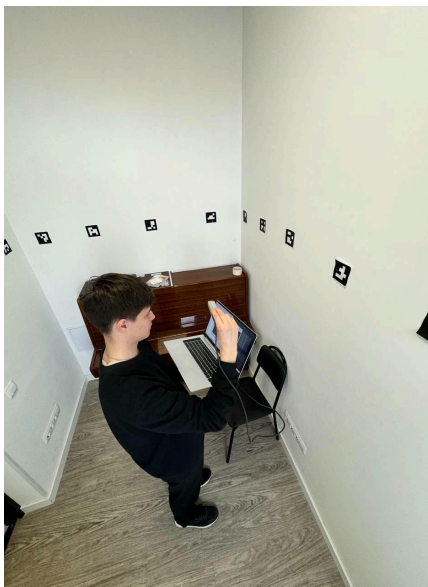


Рис. 9. Результат локализации робота

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.

## 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;
- Разработан 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;
- Реализован алгоритм ICP, разработана соответствующая нода;

- 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](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] S. Thrun, W. Burgard, и D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005. [Онлайн]. Доступно на: <https://docs.ufpr.br/~danielsantos/ProbabilisticRobotics.pdf>
- [4] ROS Wiki, «icp (ROS Fuerte) [Электронный ресурс]». [Онлайн]. Доступно на: <https://wiki.ros.org/icp>
- [5] PRBonn, «KISS-ICP [Электронный ресурс]». [Онлайн]. Доступно на: <https://github.com/PRBonn/kiss-icp>
- [6] Ishfaz, «ROS-Point-Cloud-ICP [Электронный ресурс]». [Онлайн]. Доступно на: <https://github.com/Ishfaz/ROS-Point-Cloud-ICP>
- [7] PyPI, «pyrealsense2 [Электронный ресурс]». [Онлайн]. Доступно на: <https://pypi.org/project/pyrealsense2/>
- [8] 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>
- [9] 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](https://vcg.isti.cnr.it/~cignoni/GMP2223/PDF/GP2123_12_RansacICP.pdf)