Общие материалы для роботов WHEELTEC ROS

Применение ROS для роботов:

Лидар и 2D-навигация

Камеры и обработка изображений

Содержание

1 Использование и пояснение функции отслеживания с помощью лидара ... 4
1.1 Краткое описание функции ... 4
1.2 Инструкция по использованию ... 4
1.3 Примечания ... 6
1.4 Пояснение функций ... 7
2 Использование и объяснение функции картографирования с помощью лидара ... 19
2.1 Краткое описание функции ... 19
2.2 Инструкция по использованию ... 19
2.3 Примечания ... 21
3 2D Навигация: Использование и описание ... 26
3.1 Обзор функции ... 26
3.2 Инструкция по использованию ... 27
3.3 Примечания ... 29
3.4 Объяснение функционала ... 34
4. Функция визуального слежения: использование и пояснение ... 37
4.1 Краткое описание функции ... 37
4.2 Способ использования ... 37
4.3 Примечания ... 40
4.4 Объяснение функционала ... 42
5 Функция следования по линии: использование и пояснение ... 49
5.1 Краткое описание функции ... 49
5.2 Способ использования ... 49
5.3 Объяснение программы следования по линии ... 52
6 Функция KCF-слежения ... 58
6.1 Краткое описание ... 58
6.2 Способ использования ... 58
6.3 Важные замечания ... 60
6.4 Объяснение функционала ... 62
7 Функция распознавания AR-меток: использование и пояснение ... 64
7.1 Краткое описание ... 64
7.2 Установка и описание пакета AR-меток ... 65
7.3 Способ использования ... 65

7.5 Объяснение функционала 68
8 Функция слежения за AR-меткой: использование и пояснение 70
8.1 Краткое описание 70
8.2 Способ использования 70
8.3 Важные замечания 71
8.4 Объяснение функционала 72

1 Использование и пояснение функции отслеживания с помощью лидара

1.1 Краткое описание функции

Функция отслеживания с использованием лидара работает на основе 360° сканирования окружающей среды в реальном времени. Лидар находит ближайший обнаруживаемый объект и начинает его отслеживание.

Если во время отслеживания будет найден новый объект, расположенный ближе, робот (машинка) прекратит следовать за текущим объектом и начнёт отслеживать новый, более близкий объект.

В конечном итоге робот сохраняет определённое расстояние до отслеживаемого объекта, а его передняя часть направлена на цель.

1.2 Инструкция по использованию

Перед использованием необходимо убедиться в следующем:

1. Верхний компьютер подключён к **Wi-Fi робота**.
2. Выполнено **SSH-подключение** к роботу.

Шаги:

1. **Запуск функции отслеживания лидара:** Откройте терминал и введите команду:

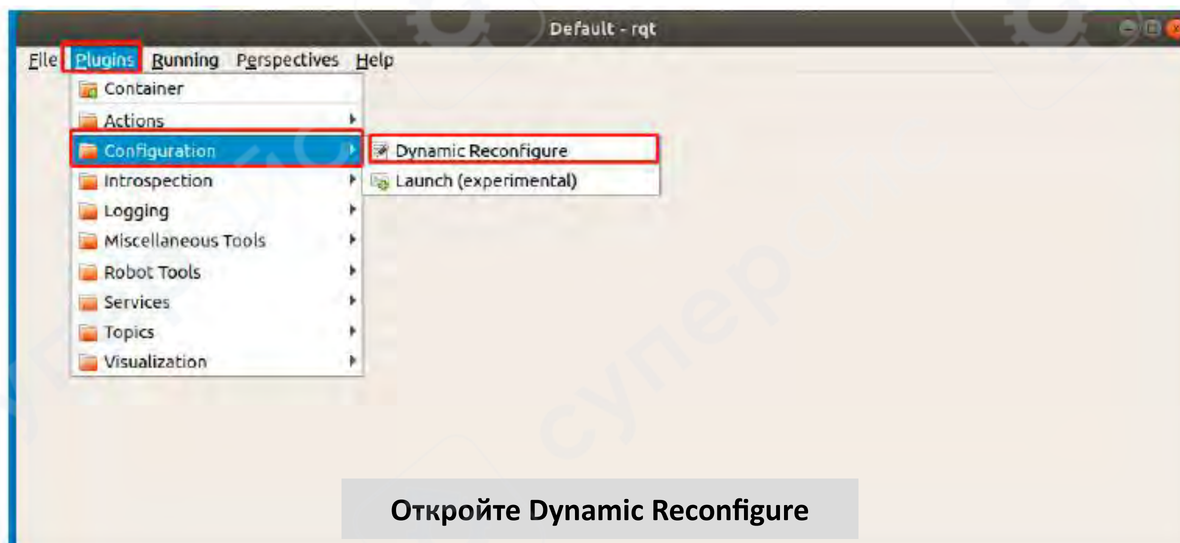
```
roslaunch simple_follower laser_follower.launch
```

2. **Настройка параметров в реальном времени через rqt:**

- Введите команду для запуска rqt:

```
rqt
```

- После успешного запуска появится окно **Default - rqt**.
- Перейдите в верхнее левое меню и выберите: **Plugins → Configuration → Dynamic Reconfigure**.



Откройте Dynamic Reconfigure

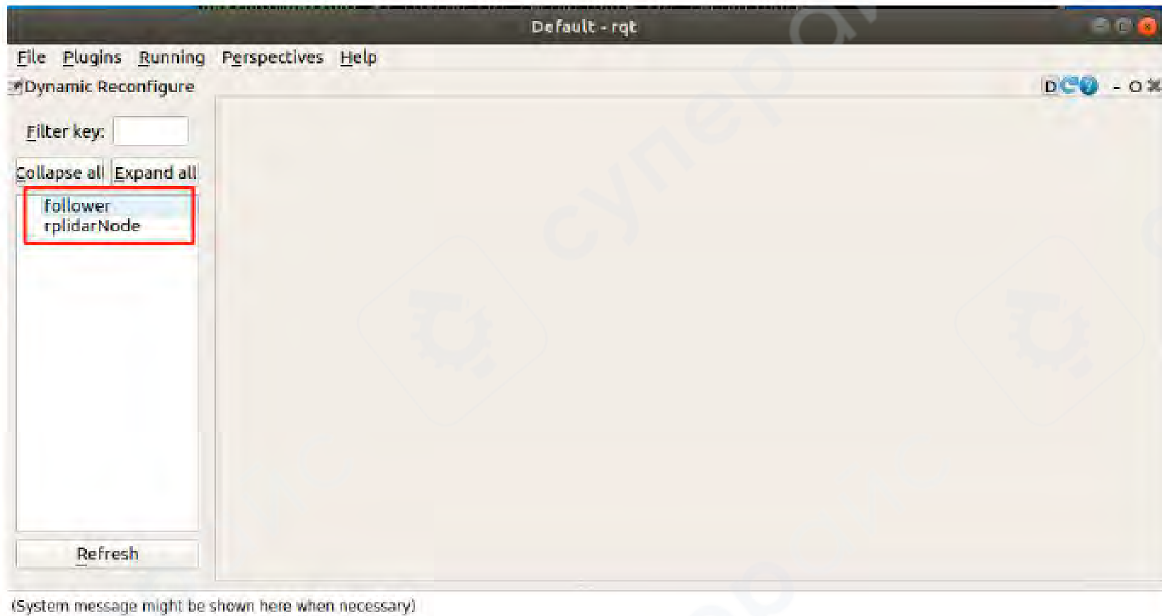
3. **Настройка параметров в Dynamic Reconfigure:**

После открытия **Dynamic Reconfigure** отобразится следующее:

- В левой панели задач будут показаны **два узла для настройки параметров**:
  - **follower**

- rplidarNode

*(Если узлы не отображаются, нажмите **Refresh** в нижнем левом углу для обновления списка).*

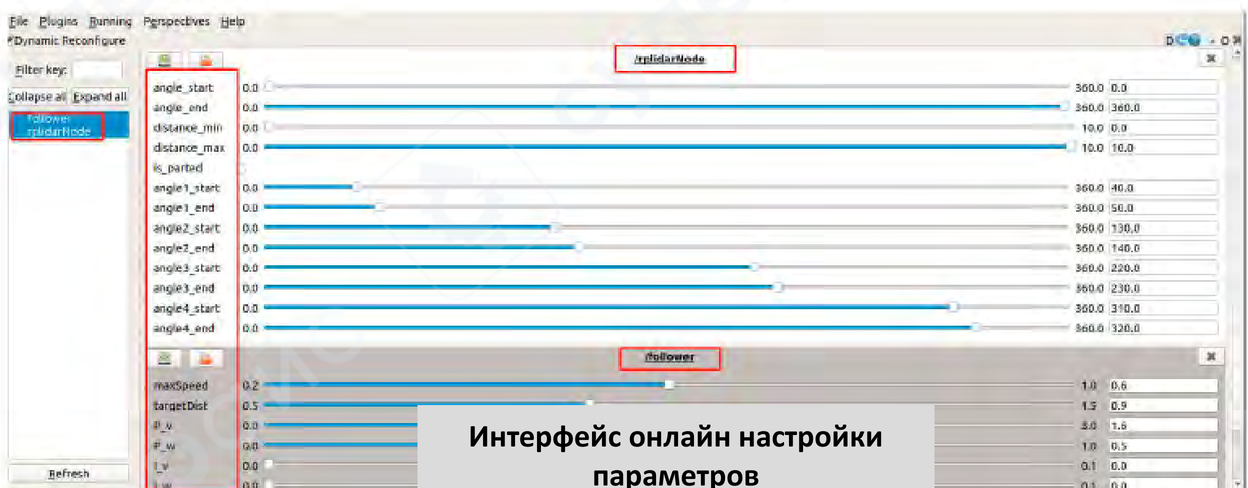


Интерфейс Dynamic Reconfigure

- Выберите узел **follower** или **rplidarNode** (можно выбрать оба).
- Войдите в интерфейс настройки параметров.
- Доступно **21 параметра** для изменения в реальном времени.
  - Для изменения параметров можно:
    - **Перемещать ползунок** на шкале.
    - **Вводить конкретные значения** в поле ввода и нажимать **Enter**.

4) Применение изменений:

- После настройки параметров робот начнёт реагировать в соответствии с новыми значениями.



Интерфейс онлайн настройки параметров

- Логи параметров (INFO) будут отображаться в терминале, где выполняется команда laser_follower.launch.

1.3 Примечания

1. Запуск функции отслеживания с помощью лидара

После выполнения команды laser_follower.launch, если в терминале **не отображаются красные ошибки**, запуск прошёл успешно.

```
[WARN] [1625489905.525372]: 1
starting
[INFO] [1625489906.904107]: PID initialised with P:[1.6, 0.5], I:[0.0, 0.0], D:[0.0, 0.0]
[INFO] [1625489906.907751]: max_speed:0.6,targetDist:0.9
[INFO] [1625489906.927095]: PID initialised with P:[1.6, 0.5], I:[0.0, 0.0], D:[0.0, 0.0]
RPLIDAR S/N: C95399F6C9E59AD2C5E59CF734813412
[ INFO] [1625489907.206649599]: Firmware Ver: 1.29
[ INFO] [1625489907.206730849]: Hardware Rev: 7
[ INFO] [1625489907.208271422]: RPLidar health status : 0
[ INFO] [1625489907.753604546]: current scan mode: Sensitivity, max_distance: 12.0 m, Point number: 7.9K , angle_compensate: 2
[INFO] [1625489907.885021]: lost connection
[WARN] [1625489909.124291]: laser no object found
[WARN] [1625489909.128987]: laser nothing found
```

Успешный запуск

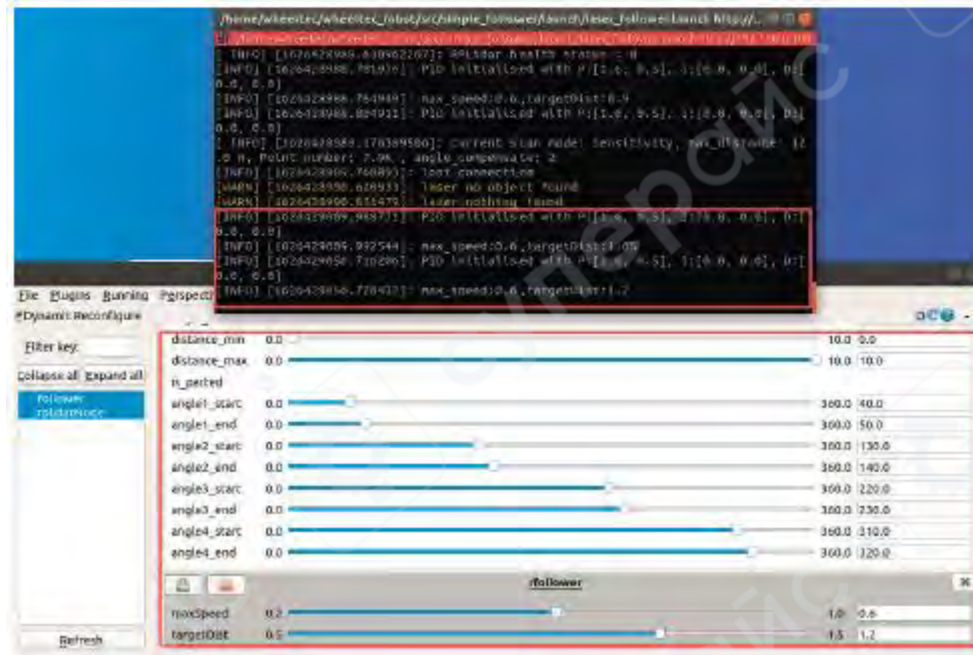
После запуска функции отслеживания робот будет постоянно искать **наиболее близкий объект** в пределах диапазона сканирования лидара. После обнаружения подходящей цели робот начнёт следовать за ней, сохраняя **оптимальное расстояние** (среднее значение), а его передняя часть будет направлена на цель (угол между роботом и объектом равен **0°**).

Внимание:

- При запуске функции отслеживания необходимо обеспечить, чтобы пространство не было **слишком узким**, а расстояние между объектами составляло **не менее 1 м**.
- Также следует обратить внимание на **высоту расположения лидара**. Поскольку лидар может сканировать только объекты, находящиеся **на одном уровне с ним**, необходимо избегать размещения **низких объектов** на пути следования робота, чтобы предотвратить столкновения.

2. Настройка параметров через rqt

- **targetDist**: При настройке параметра его значение должно быть **не менее 0.1**. В противном случае контроллер PID может воспринять разницу как слишком малую и проигнорировать её.
- **distance_min** и **distance_max**: При настройке данных параметров, если значение **distance_min** больше **distance_max**, система автоматически **поменяет их местами**, чтобы **distance_max** было больше **distance_min**.
- **angle_start** и **angle_end**: Эти параметры определяют **углы сканирования**.
- **angle1_start ~ angle4_end**: Эти параметры определяют **углы исключения (блокировки)**.
  - Углы исключения следует задавать **немного больше**, чем фактически требуемые углы блокировки.

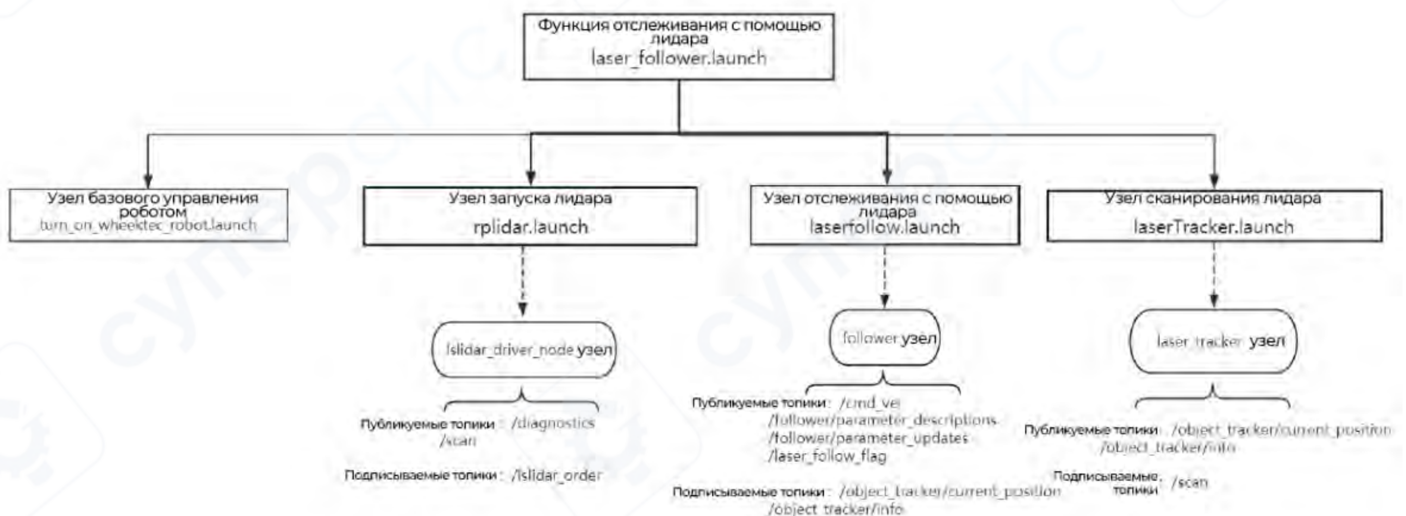


1.4 Пояснение функций

Функция отслеживания с помощью лидара активируется путём запуска laser_follower.launch. В файле laser_follower.launch содержатся **четыре launch-файла**:

- **turn_on_wheeltec_robot.launch** – запускает узел базового движения робота.
- **rplidarNode**, **laser_follow.py** и **laserTracker.py** – реализуют основные функции. *(Примечание: rplidarNode – это имя узла, а его исходный файл находится в **rplidar_ros** и соответствует файлу **node.cpp**).*

1. Запуск функции отслеживания с помощью лидара



Структура запуска функции отслеживания с помощью лидара

2. Взаимосвязь узлов функции отслеживания

```
rqt_graph
```

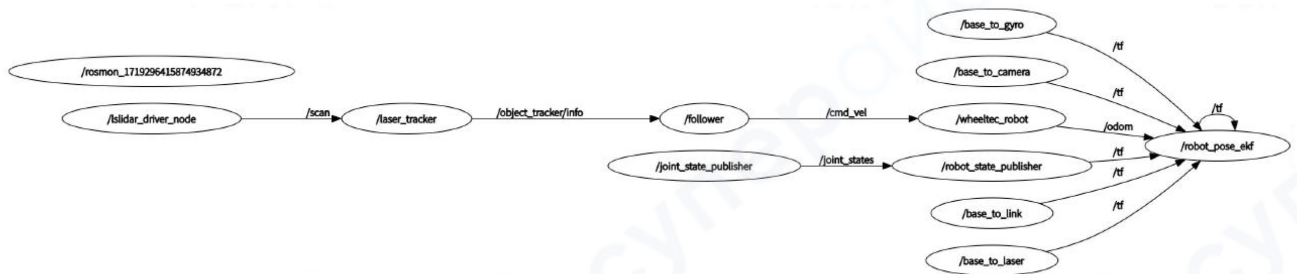
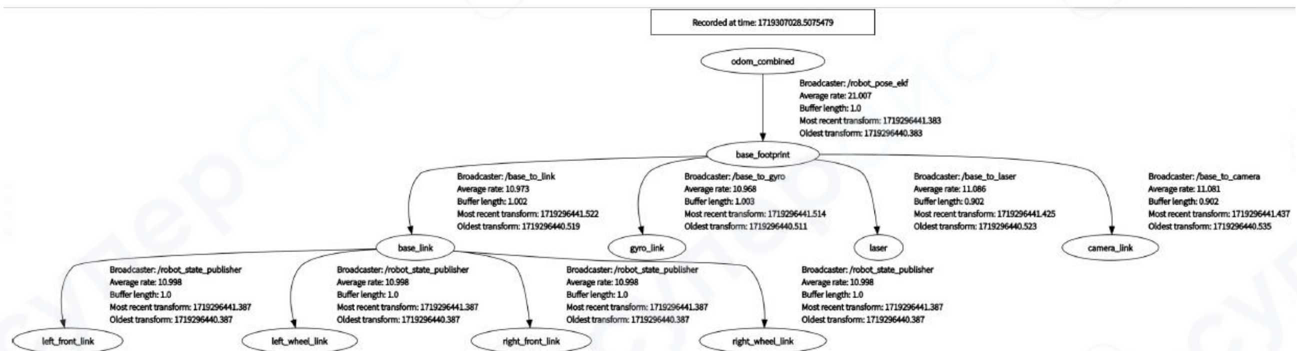


Схема взаимосвязи узлов функции отслеживания с помощью лидара

3. TF-дерево функции отслеживания с помощью лидара

Для просмотра TF-дерева выполните следующую команду:

```
rosrun rqt_tf_tree rqt_tf_tree
```



TF-дерево функции построения карты

4. Описание параметров

(1) Диапазон сканирования лидара

Диапазон сканирования определяется четырьмя параметрами: **angle_start**, **angle_end**, **distance_min** и **distance_max**.

- **angle_start** и **angle_end** – задают угол сканирования.
- **distance_min** и **distance_max** – задают радиус сканирования.

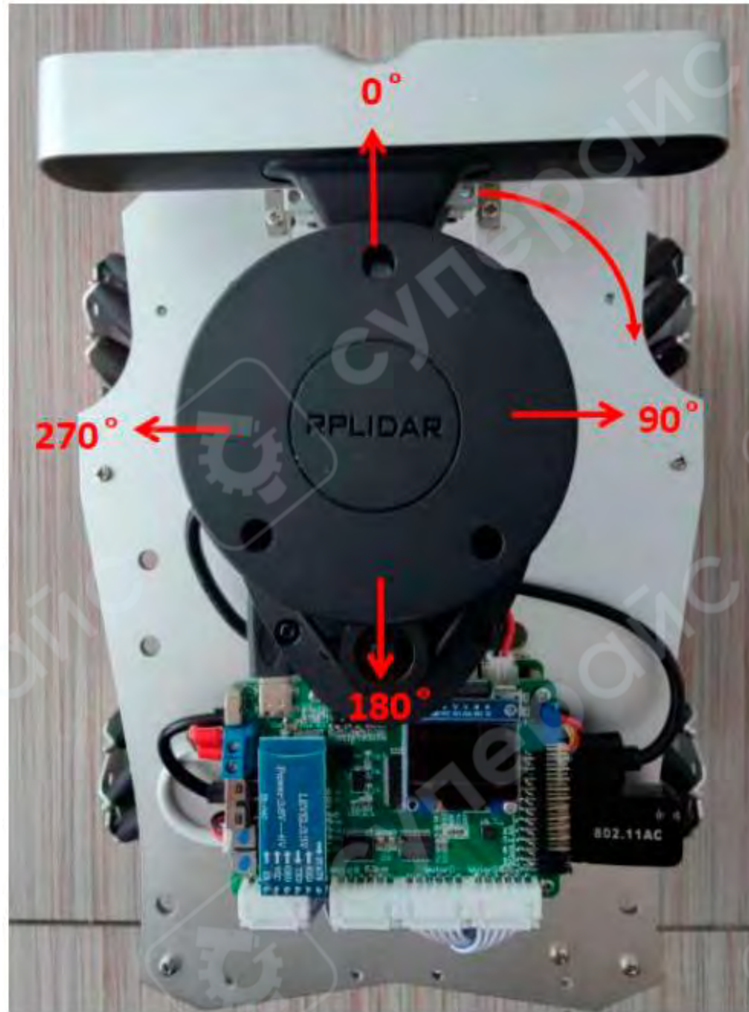
Таким образом, лидар сканирует область, ограниченную **углом** между **angle_start** и **angle_end** и **радиусом** между **distance_min** и **distance_max**.

Описание параметров:

- **angle_start**: начальный угол сканирования лидара (углы отсчитываются от **направления передней части робота**, которое считается **0°**, и увеличиваются **по часовой стрелке**, см. схему ниже).
- **angle_end**: конечный угол сканирования лидара.

*(Примечание: параметры **angle_start** и **angle_end** не ограничивают физический диапазон сканирования лидара, а **отфильтровывают данные** вне этого диапазона на уровне программы).*

- **distance_min**: минимальный радиус сканирования лидара (единица: **метры**, м).
- **distance_max**: максимальный радиус сканирования лидара.



Угол сканирования радара

(2) Многоугловое блокирование для лидара

Многоугловое блокирование состоит из **9 параметров**: **is_parted** и **angle1_start ~ angle4_end**.

- **is_parted** – это **переключатель**:
  - Если параметр установлен в **true**, включается многоугловое блокирование, и параметры **angle_start** и **angle_end** (из диапазона сканирования) становятся **неактивными**. В этом случае активируются параметры **angle1_start ~ angle4_end**.
  - Если параметр установлен в **false**, наоборот, активны **angle_start** и **angle_end**, а параметры **angle1_start ~ angle4_end** становятся **неактивными**.

При **включении многоуглового блокирования** лидар будет игнорировать диапазоны углов:

- **angle1_start ~ angle1_end**
- **angle2_start ~ angle2_end**
- **angle3_start ~ angle3_end**
- **angle4_start ~ angle4_end**

Если значения **start** и **end** совпадают, то **угол блокироваться не будет**. Таким образом, можно **контролировать количество блокируемых углов**, задавая одинаковые значения для **start** и **end**.

Важно: Параметры блокирования углов **нужно отличать** от параметров диапазона сканирования лидара.

- **Диапазон сканирования** определяет **углы, которые будут сканироваться**.
- **Блокирование углов** определяет **углы, которые будут игнорироваться**.

Кроме того, **углы блокирования** следует задавать **немного больше**, чем фактические углы, которые необходимо заблокировать.

Описание параметров:

- **is_parted**: включает или выключает многоугловое блокирование лидара.
  - При включении параметры **angle1_start ~ angle4_end** становятся активными, а параметры **angle_start** и **angle_end** – неактивными.
- **angle1_start**: начальный угол для первого диапазона блокировки.
- **angle1_end**: конечный угол для первого диапазона блокировки. *(Параметры **angle2_start ~ angle4_end** аналогичны **angle1_start ~ angle1_end**, поэтому их описание не повторяется.)*

(3) Среднее расстояние и максимальная скорость

Среднее расстояние, при котором робот остаётся в **статичном состоянии** по отношению к цели, определяется параметром **targetDist**.

Максимальная скорость движения робота при отслеживании цели задаётся параметром **maxSpeed**.

Описание параметров:

- **maxSpeed**: максимальная скорость робота при отслеживании с помощью лидара (единица измерения: **м/с**).
- **targetDist**: среднее расстояние между роботом и целью, при достижении которого робот остаётся в статичном состоянии (единица измерения: **м**).

(4) PID-значения

Значения **PID** регулируются с помощью **шести параметров**: **P_v**, **P_w**, **I_v**, **I_w**, **D_v**, **D_w**.

- **P_v** и **P_w** определяют **скорость реакции** робота.
- **I_v** и **I_w** обеспечивают более **точное приближение** к целевому значению и уменьшают ошибку.
- **D_v** и **D_w** уменьшают **колебания** робота вблизи целевого значения, сокращая **перерегулирование** и **время стабилизации**.

Описание параметров:

- **P_v**: пропорциональный коэффициент для **линейной скорости**.
- **P_w**: пропорциональный коэффициент для **угловой скорости**.
- **I_v**: интегральный коэффициент для **линейной скорости**.
- **I_w**: интегральный коэффициент для **угловой скорости**.
- **D_v**: дифференциальный коэффициент для **линейной скорости**.
- **D_w**: дифференциальный коэффициент для **угловой скорости**.

5. Изменение параметров

Существует **два способа** изменения параметров функции отслеживания с помощью лидара:

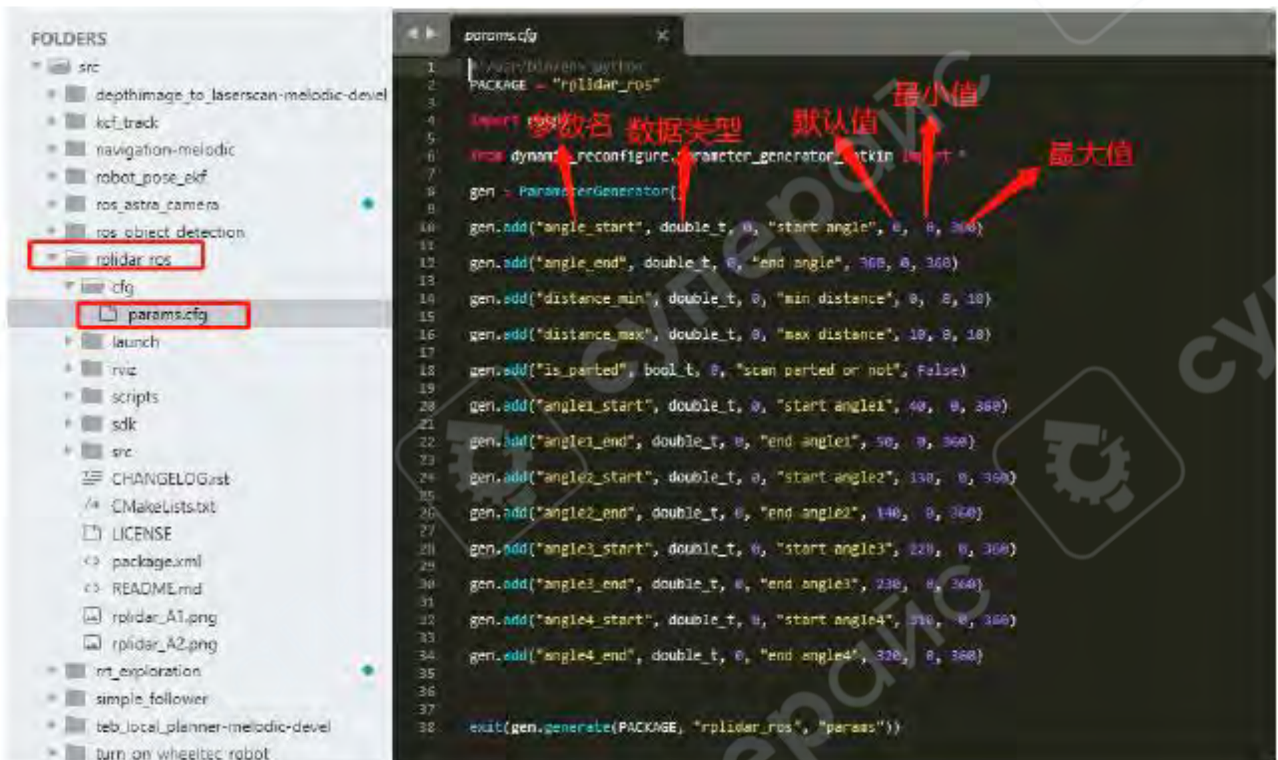
1. **Изменение значений в исходных файлах:**
   - Параметры можно изменить **напрямую в файлах**:
     - Значения в **launch-файлах** вступают в силу **сразу после сохранения** и **не требуют перекомпиляции**.
     - Значения в **cfg-файлах** требуют **перекомпиляции** для применения изменений.
   - Такой способ делает изменения **долгосрочными**, однако необходимо **перезапустить узел**, чтобы изменения вступили в силу.
2. **Онлайн настройка через rqt:**
   - Позволяет **настроить параметры в реальном времени** во время работы узла.
   - Изменения вступают в силу **немедленно**, без необходимости перезапуска узла.
   - Однако параметры, изменённые через **rqt**, действуют **только в текущем сеансе работы узла**. После перезапуска узла они сбрасываются на значения по умолчанию, хранящиеся на **сервере параметров**.
   - Таким образом, **rqt** используется как инструмент **отладки**, но для сохранения параметров на постоянной основе их необходимо изменить в **исходных файлах**.

Примечание:

- На изображении ниже представлен **cfg-файл** для онлайн-настройки параметров узла лидара через **rqt**.
- В этом файле можно изменить **минимальные и максимальные значения**, чтобы задать **верхний и нижний пределы** для онлайн настройки параметров.
- Однако **значения по умолчанию** в этом файле будут **переопределены** значениями с **сервера параметров**, поэтому изменения значений по умолчанию **не вступят в силу**.
- Для изменения значений по умолчанию необходимо редактировать **launch-файлы**, за исключением **шести параметров PID** (P_v, P_w, I_v, I_w, D_v, D_w), которые **не хранятся на сервере параметров**.

Внимание:

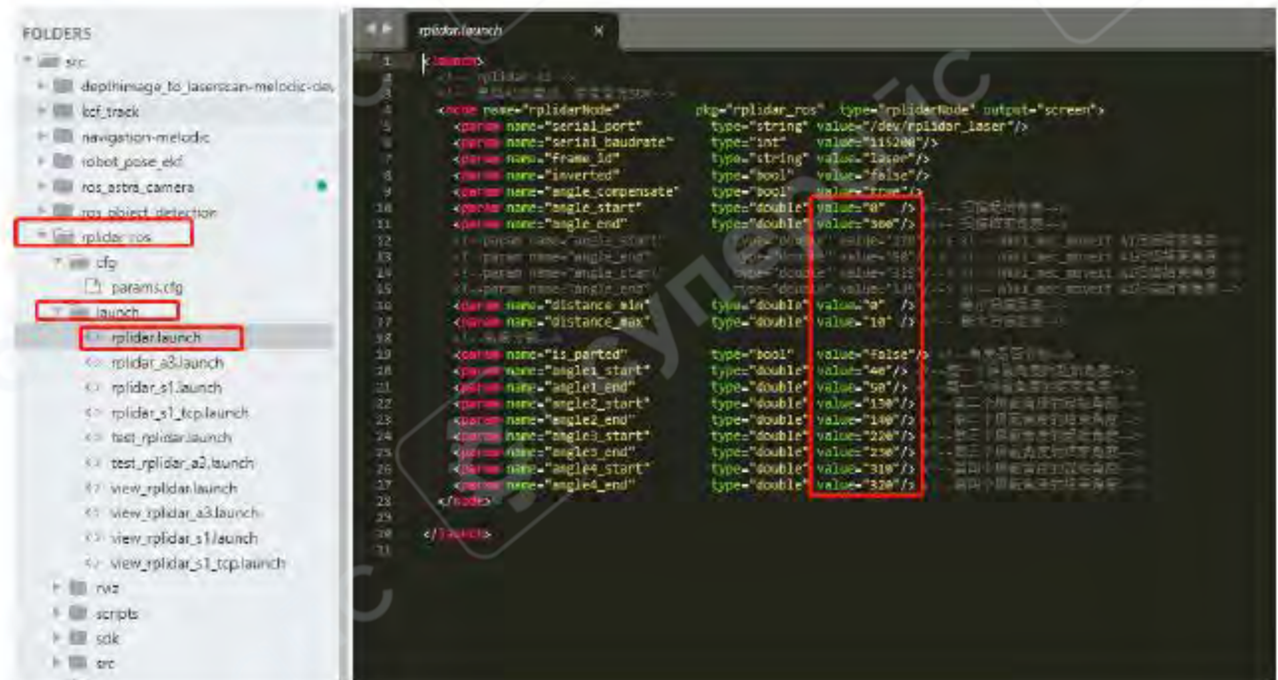
- При изменении минимальных и максимальных значений параметров необходимо учитывать **пределы производительности** лидара и робота.
- Изменения в **cfg-файлах** требуют **перекомпиляции** для вступления в силу. Перед компиляцией необходимо убедиться, что **системное время** установлено правильно.



cfg-файл узла лидара

Инструкция по изменению параметров в launch-файлах:

- В каждом **launch-файле**, соответствующем функциональному узлу, можно изменить значение параметра, отредактировав **атрибут value** в секции **param**.
- После сохранения **launch-файла** изменения вступают в силу **немедленно**, без необходимости компиляции.

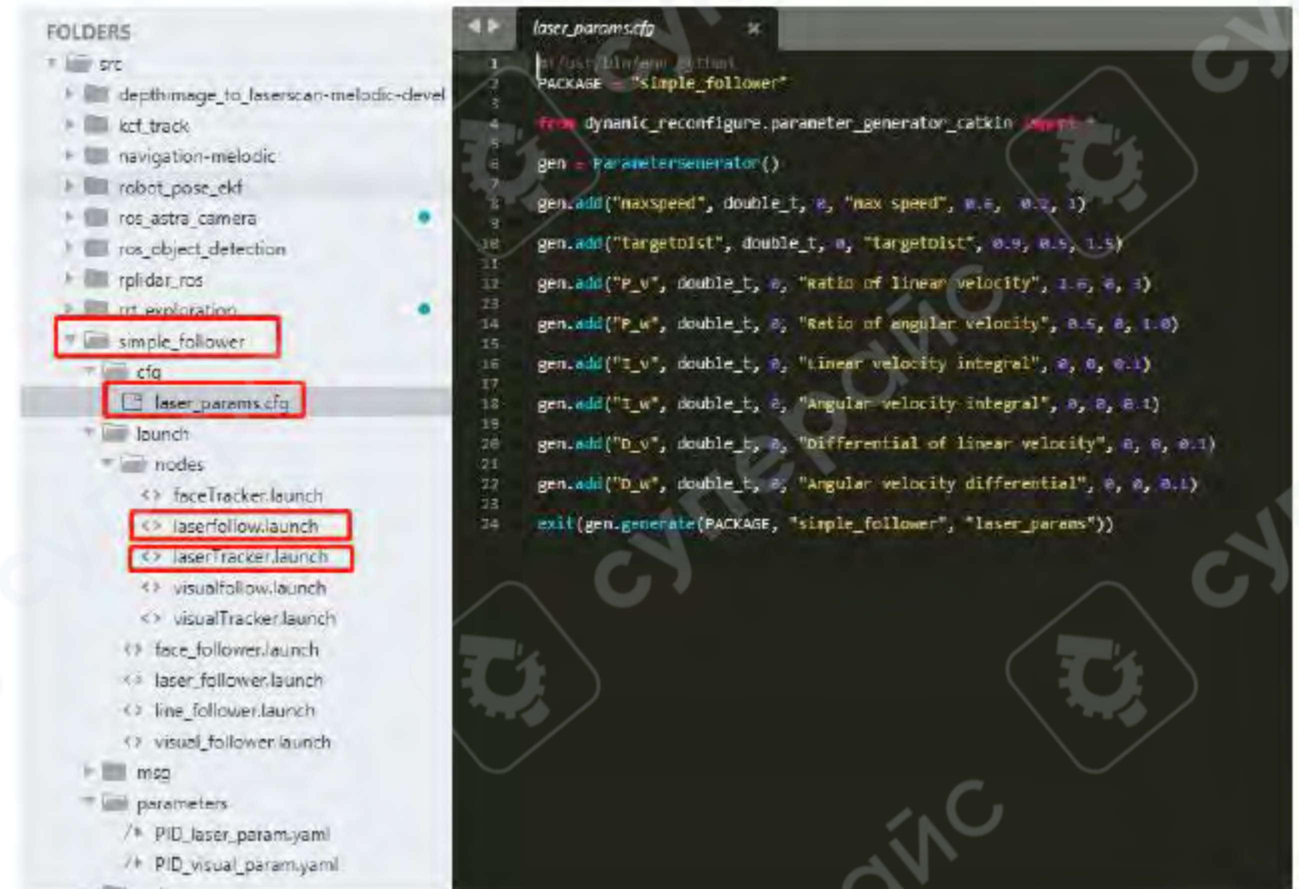


launch-файл узла лидара

На изображении ниже показаны **cfg-файл** и **launch-файл** для функции отслеживания с помощью лидара. Основные параметры, которые можно изменить:

- **Максимальная скорость**
- **Среднее расстояние (targetDist)**
- **PID-параметры**

Способ изменения параметров аналогичен способу для узла лидара и здесь **не будет повторяться**.



cfg-файл и launch-файл функции отслеживания с помощью лидара

6. Процесс работы функции отслеживания с помощью лидара

1. **Сканирование и публикация данных**

Лидар выполняет **360° сканирование**, получает данные и отправляет их на обработку в узел лидара. После обработки данные публикуются в топике **/scan** в виде сообщений типа **LaserScan**.

2. **Обработка данных в laserTracker.py**

Узел **laserTracker.py** содержит **scanSubscriber**, который подписывается на топик **/scan** и принимает сообщения **LaserScan** с данными о точках 360° сканирования. Далее из полученных данных определяется **ближайшая и корректная цель**.

- Если расстояние до цели **превышает максимальный радиус** сканирования лидара, цель считается **некорректной**. В этом случае узел публикует сообщение типа **StringMsg** в топик **/object_tracker/info**.

- Узел **laser_follow.py** с помощью **trackerInfoSubscriber** подписывается на **/object_tracker/info** и при получении сообщения выводит в терминал лог:

“laser: nothing found” (цель не найдена).

3. **Расчёт угла и публикация позиции**

Если расстояние до цели находится **в пределах диапазона сканирования**, производится **вычисление угла** к цели. Результат публикуется в топик **/object_tracker/current_position** в виде сообщения типа **Position**, которое содержит:

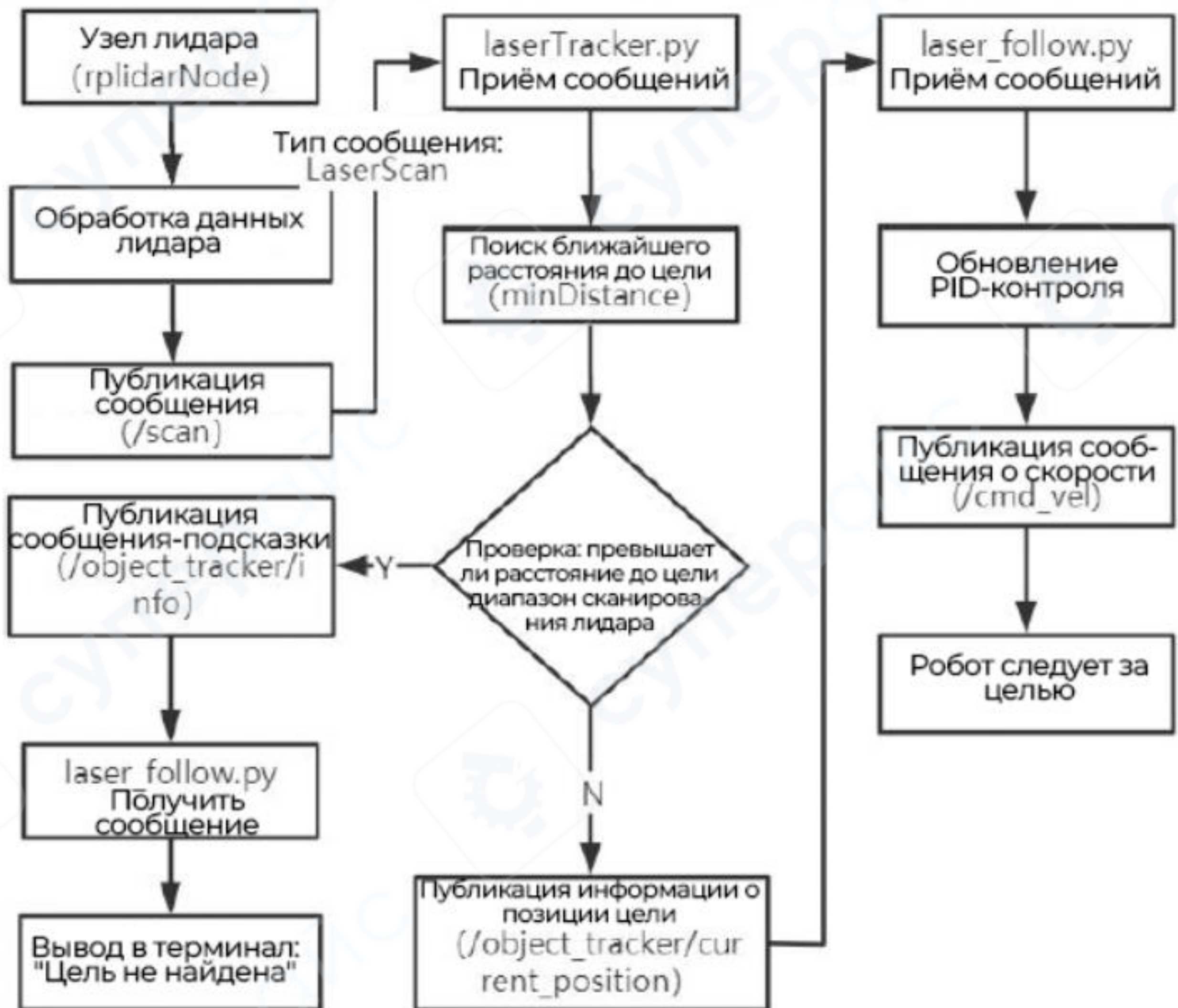
- **угол к цели**,
- **расстояние до цели**.

Узел **laser_follow.py** содержит **positionSubscriber**, который подписывается на **/object_tracker/current_position** и при получении сообщения обновляет **PID-контроль**.

4. **Публикация скорости и следование за целью**

После обновления параметров PID-контроля узел публикует сообщение типа **Twist** в топик **/cmd_vel**, чтобы управлять **скоростью робота**.

- Если расстояние до цели достигает **среднего значения (targetDist)** и угол равен **0°**, скорость робота устанавливается равной **0**.



Слежение за ходом выполнения программы

7. Ключевой разбор исходного кода функции отслеживания лидара

Исходный файл: rplidar_ros/src/node.cpp

```
for (size_t i = 0; i < node_count; i++) {
   float read_value = (float) nodes[i].dist_mm_q2/4.0f/1000;
   // angle_cur - текущий угол сканирования
   angle_cur = i*360/node_count;

   if(angle_end>=angle_start) {
      if(angle_cur>angle_end || angle_cur<angle_start) read_value=0;
   } else {
      if(angle_cur>angle_end && angle_cur<angle_start) read_value=0;
   }

   if (read_value <= distance_min || read_value >= distance_max)
      scan_msg.ranges[i] = std::numeric_limits<float>::infinity();
   else
      scan_msg.ranges[i] = read_value;

   scan_msg.intensities[i] = (float) (nodes[i].quality >> 2);
}
```

Описание:

Этот участок кода ограничивает **диапазон сканирования лидара**.

- **angle_cur**: текущий угол сканирования.
- **read_value**: расстояние до препятствия для текущего угла.

Логика работы:

1. Проверяется, попадает ли **angle_cur** в диапазон **angle_start ~ angle_end**. Если угол не попадает, **read_value** обнуляется.
2. Если **read_value** меньше минимального радиуса (**distance_min**) или больше максимального радиуса (**distance_max**), значение устанавливается в **бесконечность**. Это означает, что на этом угле препятствия нет.
3. Таким образом, **лидар продолжает 360° сканирование**, но данные за пределами установленных углов и радиусов **фильтруются**.

Исходный файл: laserTracker.py

```
def registerScan(self, scan_data):
   ranges = np.array(scan_data.ranges)
   sortedIndices = np.argsort(ranges)  # Сортировка по расстоянию
   minDistanceID = None
   minDistance = float('inf')

   if(not(self.lastScan is None)):
      for i in sortedIndices:
```

```
                tempMinDistance = ranges[i]
                windowIndex = np.clip([i-self.winSize, i+self.winSize+1], 0, len(self.lastScan))
                window = self.lastScan[windowIndex[0]:windowIndex[1]]

                with np.errstate(invalid='ignore'):
                    if(np.any(abs(window-tempMinDistance)<=self.deltaDist)):
                        minDistanceID = i
                        minDistance = ranges[minDistanceID]
                        break
        self.lastScan = ranges
```

Описание:

Эта часть кода предназначена для **поиска ближайшей действительной цели**:

- **ranges**: массив расстояний для всех точек сканирования.
- С помощью np.argsort() массив сортируется по значениям от меньших к большим, а затем **перебирается** от ближайших точек к дальним.
- **window**: окно данных, представляющее часть диапазона текущего сканирования, которое сравнивается с предыдущим сканированием.
- Если разница между текущим и предыдущим значением в пределах **deltaDist**, точка считается допустимой, и цикл завершается.

```
    if(minDistance > scan_data.range_max):
        # Не найдено подходящей цели
        rospy.logwarn('laser no object found')
        self.infoPublisher.publish(StringMsg('laser:nothing found'))

    else:
        # Вычисление угла позиции объекта, 0 соответствует прямой вперед
        minDistanceAngle = scan_data.angle_min + minDistanceID * scan_data.angle_increment
        self.positionPublisher.publish(PositionMsg(minDistanceAngle, 42, minDistance))
        #self.infoPublisher.publish(StringMsg('successful'))
```

Описание:

- Если ближайшее расстояние (**minDistance**) превышает максимальный радиус сканирования (**range_max**), то публикуется предупреждающий лог о том, что цель **не найдена**.
- В противном случае вычисляется угол объекта (**minDistanceAngle**):
  - **angle_min**: начальный угол сканирования.
  - **angle_increment**: угол между соседними точками.
  - Угол и расстояние публикуются в виде сообщения типа **PositionMsg**.

Исходный файл: laser_follow.py

```
    # Dynamic Reconfigure Config
    def reconfigCB(self, config, level):
        self.max_speed = config.maxSpeed  # Присвоение максимальной скорости из настроек
        targetDist = config.targetDist  # Присвоение среднего значения расстояния
```

```
        # Получение PID-параметров
        PID_param['P'] = [config.P_v, config.P_w]
        PID_param['I'] = [config.I_v, config.I_w]
        PID_param['D'] = [config.D_v, config.D_w]

        # Создание объекта simplePID для обновления PID-контроля
        self.PID_controller = simplePID([0, targetDist], PID_param['P'], PID_param['I'],
PID_param['D'])
        rospy.loginfo("max_speed:{}, targetDist:{}".format(self.max_speed, targetDist))
        return config
```

Описание:

Этот фрагмент кода реализует **онлайн настройку параметров** с использованием **rqt Dynamic Reconfigure**.

- Параметры **максимальной скорости** и **среднего расстояния (targetDist)** обновляются.
- Создаётся объект **simplePID** для управления движением с обновлёнными параметрами **P, I, D**.

```
    def positionUpdateCallback(self, position):
        angleX = position.angleX  # Угол к цели
        distance = position.distance  # Расстояние до цели

        if(angleX > 0):
            angleX = angleX - 3.1415  # Коррекция угла: поворот на 180°
        else:
            angleX = angleX + 3.1415

        # Обновление PID-контроллера для получения новой скорости
        [uncliped_ang_speed, uncliped_lin_speed] = self.PID_controller.update([angleX, distance])

        # Ограничение скорости по максимальному значению
        angularSpeed = np.clip(-uncliped_ang_speed, -self.max_speed, self.max_speed)
        linearSpeed = np.clip(-uncliped_lin_speed, -self.max_speed, self.max_speed)

        # Создание сообщения Twist и публикация в топик /cmd_vel
        velocity = Twist()
        velocity.linear = Vector3(linearSpeed, 0, 0)  # Линейная скорость
        velocity.angular = Vector3(0, 0, angularSpeed)  # Угловая скорость
        self.cmdVelPublisher.publish(velocity)
```

Описание:

1. Угол к цели **корректируется на 180°** для соответствия ориентации вперёд.
2. **PID-контроллер** обновляет значения угловой и линейной скорости на основе текущего угла и расстояния.

3. Значения **ограничиваются** максимальной скоростью с помощью np.clip.
4. Сообщение типа **Twist** публикуется в топик **/cmd_vel** для управления движением робота.

Исходный файл: laser_follow.py

```
def update(self, current_value):
    current_value = np.array(current_value)  # [angleX distance]
    if(np.size(current_value) != np.size(self.setPoint)):
        raise TypeError('current_value and target do not have the same shape')

    if(self.timeOfLastCall is None):
        # Первое вызов функции PID: разница времени неизвестна
        # Управляющий сигнал не применяется
        self.timeOfLastCall = time.clock()
        return np.zeros(np.size(current_value))

    error = self.setPoint - current_value  # Значение ошибки

    # Остановка при малой величине ошибки
    if error[0] < 0.1 and error[0] > -0.1:  # error[0] – ошибка угла
        error[0] = 0
    if error[1] < 0.1 and error[1] > -0.1:  # error[1] – ошибка расстояния
        error[1] = 0

    # При небольшом целевом значении увеличиваем скорость за счет масштабирования ошибки
    if error[1] > 0 and self.setPoint[1] < 1.3:
        error[1] = error[1] * (1.3 / self.setPoint[1])

    P = error
    currentTime = time.clock()
    deltaT = (currentTime - self.timeOfLastCall)  # Разница во времени

    # Интегральная составляющая ошибки (текущая ошибка * время)
    self.integrator = self.integrator + (error * deltaT)
    I = self.integrator

    # Дифференциальная составляющая ошибки (разница ошибок / время)
    D = (error - self.last_error) / deltaT

    # Обновляем значения для следующего вызова
    self.last_error = error
    self.timeOfLastCall = currentTime

    # Возвращаем управляющий сигнал
```

```
        return self.Kp * P + self.Ki * I + self.Kd * D
```

Описание кода:

Этот фрагмент кода представляет метод для **обновления управляющих сигналов** PID-регулятора, определённый в классе **simplePID**.

1. **current_value** – текущие значения угла и расстояния между роботом и целью, представленные в виде массива **[angleX distance]**.
2. **error** – отклонение (разница) между текущими значениями **current_value** и целевыми значениями **setPoint**.
   - Целевое значение **setPoint** по умолчанию – **[0 targetDist]** (где **0** – угол, а **targetDist** – среднее расстояние).
3. **Проверка малой ошибки:**
   - Если ошибка угла **error[0]** или расстояния **error[1]** меньше **0.1**, значение устанавливается в **0**, чтобы робот не колебался при минимальных отклонениях.
4. **Масштабирование ошибки:**
   - Если расстояние до цели меньше **1.3**, ошибка увеличивается для повышения скорости перемещения робота.
5. **Вычисление PID-компонентов:**
   - **P** – пропорциональная составляющая: текущая ошибка.
   - **I** – интегральная составляющая: накопленная ошибка за время **deltaT**.
   - **D** – дифференциальная составляющая: скорость изменения ошибки (разница ошибок за время).
6. **Обновление времени и ошибки:**
   - Текущее время и ошибка сохраняются для использования при следующем вызове.
7. **Возврат управляющего сигнала:**
   - Итоговый управляющий сигнал рассчитывается как: $\text{Control Signal} = K_p \cdot P + K_i \cdot I + K_d \cdot D$

   где **K_p**, **K_i**, **K_d** – коэффициенты PID-регулятора.

2 Использование и объяснение функции картографирования с помощью лидара

2.1 Краткое описание функции

Функция картографирования с использованием лазерного лидара реализуется с применением **алгоритма SLAM**.

Робот движется в **неизвестной среде** и в процессе перемещения использует информацию с датчиков, таких как **облако точек лидара** или **одометрические данные о позе**, для:

- Определения **своего местоположения**;
- **Построения окружающей среды** и **генерации двумерной сетчатой карты**.

После завершения создания карты робот сможет выполнять **навигацию** и **позиционирование** в данном окружении.

2.2 Инструкция по использованию

Перед использованием необходимо:

- Убедиться, что **верхний компьютер** подключён к **Wi-Fi робота**.

- Выполнить **SSH-доступ** к терминалу робота.

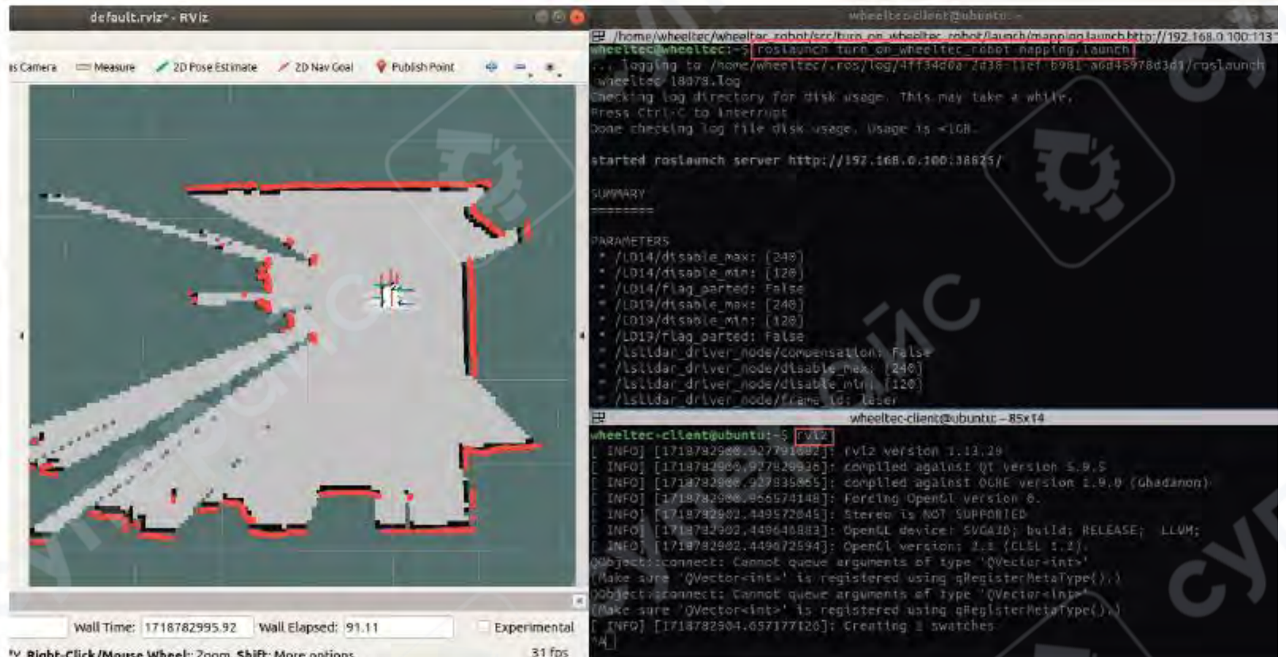
1. **Запуск функции картографирования:**

Введите следующую команду в терминале:

```
roslaunch turn_on_wheeltec_robot mapping.launch
```

2. **Запуск инструмента визуализации RViz** на главном компьютере для отображения процесса картографирования.

Результат выглядит, как показано на следующем рисунке.



Терминал картографирования

3. **Управление движением робота для завершения картографирования:**

Можно использовать **различные способы управления**, такие как:

- **Клавиатура**;
- **Bluetooth**;
- **Геймпад**;
- **Дистанционное управление через приложение**.

Для управления с **клавиатуры** используйте следующую команду:

```
roslaunch wheeltec_robot_rc keyboard_teleop.launch
```

(Примечание: узлы низкого уровня уже активированы по умолчанию в mapping.launch.)

4. **Сохранение карты после завершения картографирования:**

Введите следующую команду:

```
roslaunch turn_on_wheeltec_robot map_saver.launch
```

```
wheeltec@wheeltec:~$ roslaunch turn_on_wheeltec_robot map_saver.launch

... logging to /home/wheeltec/.ros/log/bb5fa174-c1bb-11eb-99a2-e45f0103e8eb/roslaunch-wheeltec-7062.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.0.100:37071/

SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.10

NODES
  /
    map_saver1 (map_server/map_saver)

ROS_MASTER_URI=http://192.168.0.100:11311

process[map_saver1-1]: started with pid [7093]
[ INFO] [1622430430.555555862]: Waiting for the map
[ INFO] [1622430430.843496621]: Received a 576 X 576 map @ 0.050 m/pix
[ INFO] [1622430430.843646806]: Writing map occupancy data to /home/wheeltec/wheeltec_robot/src/turn_on_wheeltec_robot/map
/WHEELTEC.pgm
[ INFO] [1622430430.885284121]: Writing map occupancy data to /home/wheeltec/wheeltec_robot/src/turn_on_wheeltec_robot/map
/WHEELTEC.yaml
[ INFO] [1622430430.885949380]: Done

[map_saver1-1] process has finished cleanly
log file: /home/wheeltec/.ros/log/bb5fa174-c1bb-11eb-99a2-e45f0103e8eb/map_saver1-1*.log
all processes on machine have died, roslaunch will exit
shutting down processing monitor...
... shutting down processing monitor complete
done
```

读取所建地图数据
保存地图文件的路径
会保存两个文件 分别是pgm和yaml文件
提示完成保存 该launch文件运行结束后会自动退出运行

Терминал сохранения карты

2.3 Примечания

1. Проблемы с положением робота

После запуска узла картографирования и открытия **RViz** может возникнуть следующая ситуация:

- Визуальное отображение робота в **RViz** показывает **перекос или наклон корпуса**.

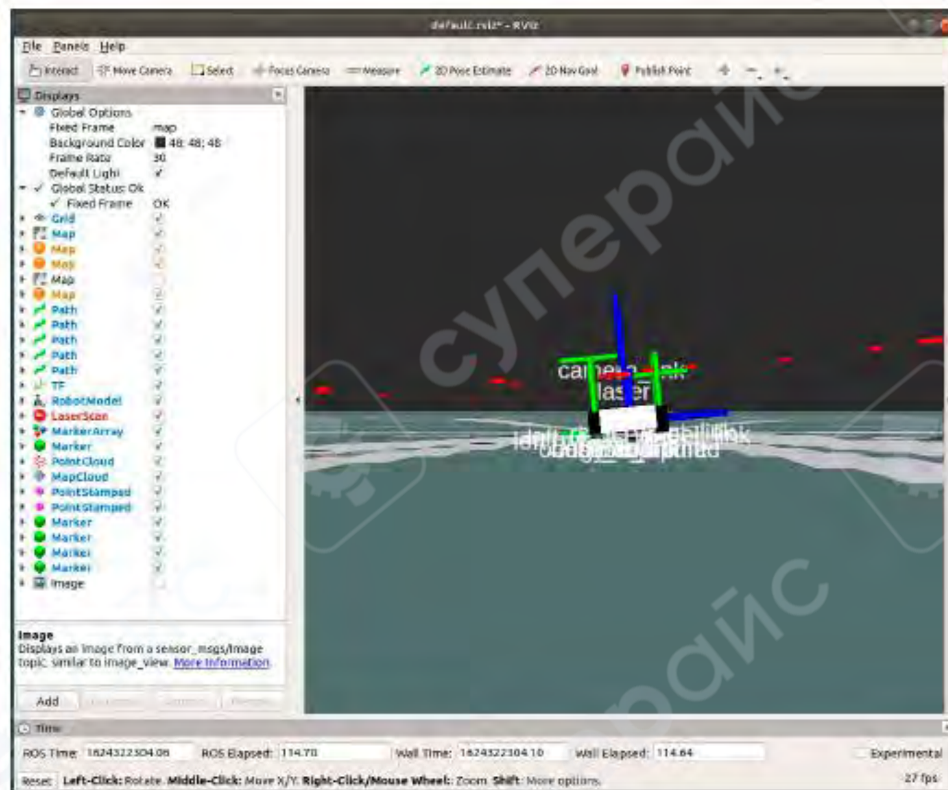
Последствия:

- Качество построения карты в таком состоянии будет **низким**.
- Карта может **сдвигаться** или отображаться некорректно.

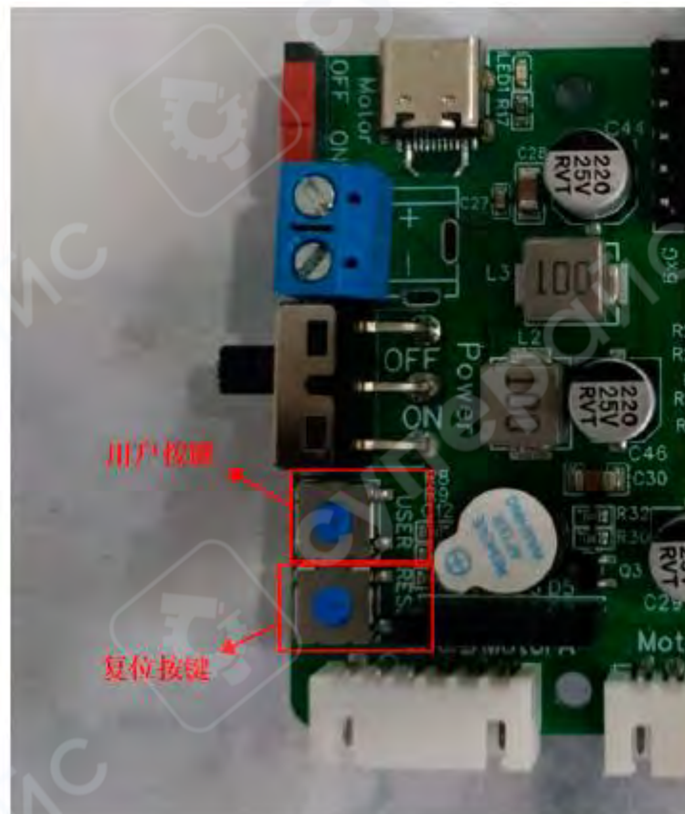
Решение:

1. Установите робота на ровную поверхность в зоне, где будет выполняться картографирование.
2. Нажмите кнопку **сброса (reset)** или **пользовательскую кнопку** на контроллере STM32.
3. Перезапустите узел картографирования.

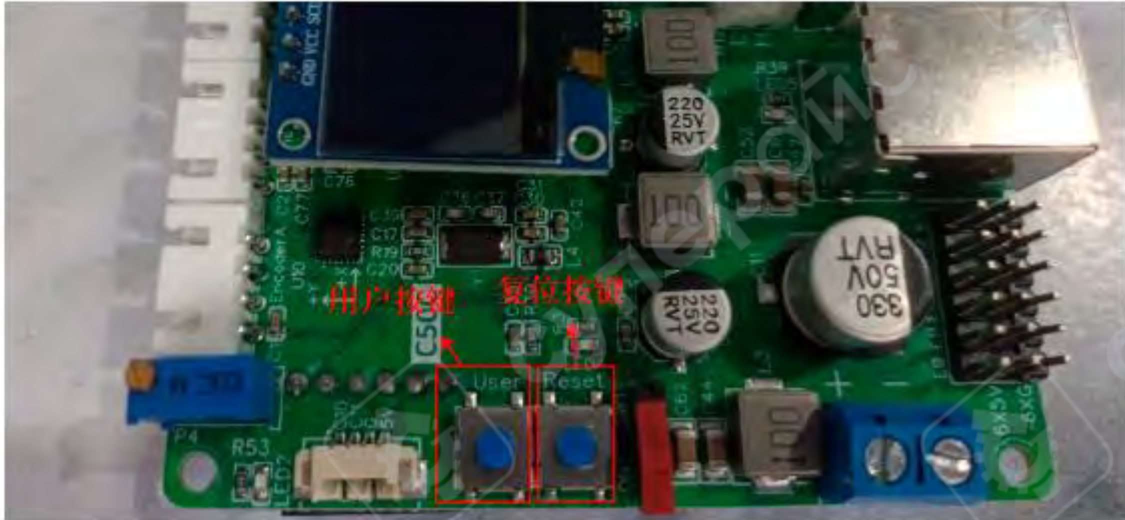
После сброса положения робота его ориентация будет исправлена, и можно продолжить процесс картографирования.



Перекос корпуса робота



Для ROS-образовательных роботов: Дважды нажмите кнопку пользователя или один раз нажмите кнопку сброса, чтобы сбросить ориентацию



Для больших ROS-исследовательских роботов: Один раз нажмите кнопку пользователя или кнопку сброса, чтобы сбросить ориентацию

2. Замечания по сохранению карты

- **Важно:** После выхода из узла картографирования карта **не может быть сохранена**. Поэтому необходимо **сначала сохранить карту**, а затем закрыть узел.
- **Способы сохранения карты:**

1. Используйте функцию **автоматического сохранения**.
2. Для сохранения карты в определённое место с конкретным именем выполните следующие шаги:

   ▪ Перейдите в папку, куда вы хотите сохранить карту. В данном примере используется путь к функциональному пакету **turn_on_wheeltec_robot**.

   ▪ Запустите узел сохранения карты, указав путь и имя файла после параметра -f.

Пример команды:

```
rosrun map_server map_saver -f WHEELTEC
```

После выполнения этой команды в указанной папке будут созданы два файла:

- **WHEELTEC.pgm** – файл изображения карты.
- **WHEELTEC.yaml** – файл конфигурации карты

```
wheeltec@wheeltec:~$ cd /home/wheeltec/wheeltec_robot/src/turn_on_wheeltec_robot/
wheeltec@wheeltec:~/wheeltec_robot/src/turn_on_wheeltec_robot$ rosrun map_server map_saver -f WHEELTEC
```

Пользовательское сохранение пути и имени файла карты

После сохранения вы можете просмотреть сохраненную карту в файловом менеджере как изображение в формате pgm.

WHEELTEC.pgm – изображение карты

```
image: /home/wheeltec/wheeltec_robot/src/turn_on_wheeltec_robot/map/WHEELTEC.pgm  #地图文件路径
resolution: 0.050000                           #地图分辨率单位 米/像素
origin: [ 5.000000, 15.200000, 0.000000]       #地图左下角像素的2D姿态为(x, y, yaw), yaw为逆时针旋转 (yaw=0 表示不旋转)
negate: 0                                      #白/黑/空闲/占用的语义是否应该颠倒
occupied_thresh: 0.65                          #占用概率大于此阈值的像素被视为完全占用
free_thresh: 0.196                             #占用概率小于此阈值的像素被视为完全空闲
```

WHEELTEC.yaml – файл конфигурации карты

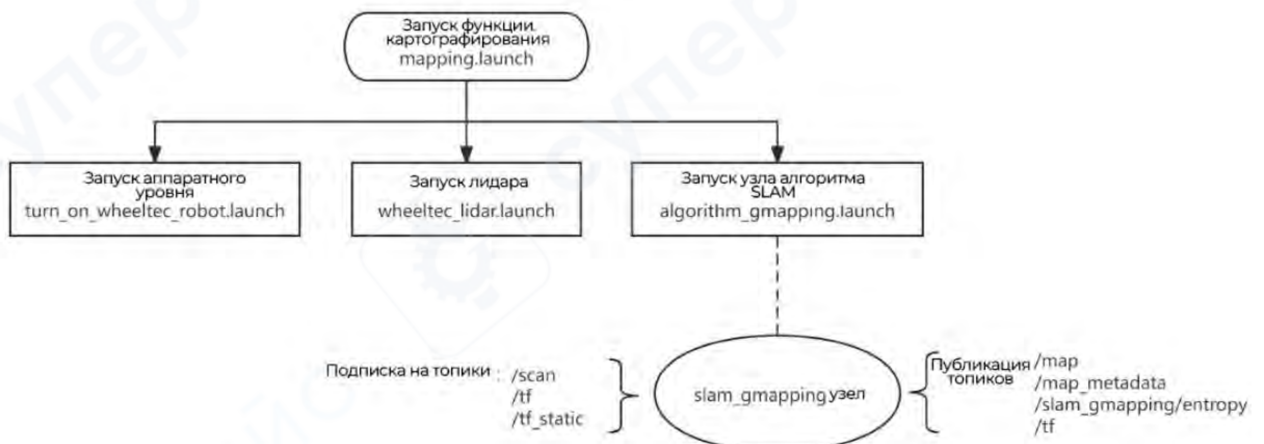
4. Объяснение функции

Файл запуска функции картографирования с использованием лидара находится в пакете **turn_on_wheeltec_robot** рабочей области.

Запуск выполняется через файл **mapping.launch**, который активирует узлы нижнего уровня, оборудование, а также узел алгоритма **SLAM**.

(В данном случае используется алгоритм **gmapping** по умолчанию. Подробное описание других алгоритмов картографирования доступно в разделе «Переключение алгоритмов картографирования и их преимущества и недостатки».)

1. Запуск функции картографирования с помощью лидара

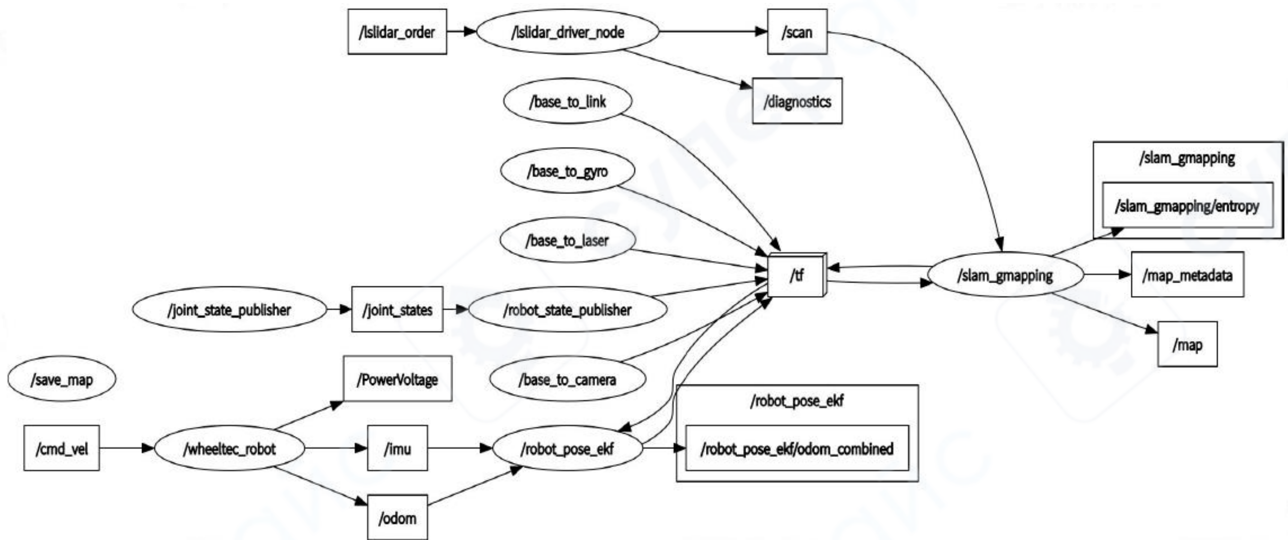


Структура запуска функции картографирования

2. Отображение узлов функции картографирования

Для просмотра связей узлов используйте:

```
rqt_graph
```

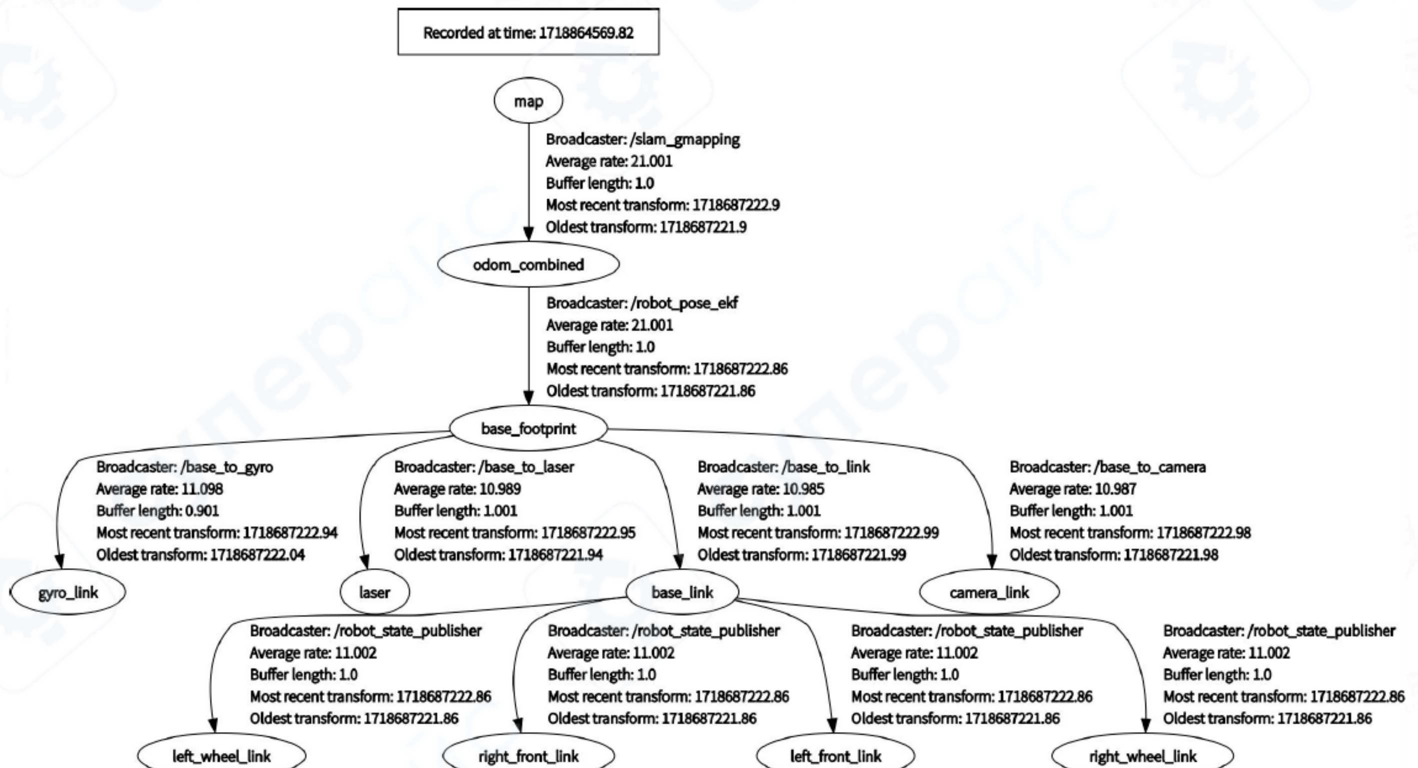


Граф узлов функции картографирования

3. TF-дерево для функции картографирования

Для визуализации TF-дерева используйте:

```
rosrun rqt_tf_tree rqt_tf_tree
```



TF-дерево функции картографирования

4. Описание файла запуска функции картографирования

Файл **mapping.launch** находится по пути:
turn_on_wheeltec_robot/launch

Пример содержимого файла **mapping.launch**:

```
<!-- Выбор алгоритма картографирования -->
<arg name="mapping_mode" default="gmapping" doc="opt: gmapping,hector,cartographer,karto" />

<!-- Включение навигации при картографировании, по умолчанию выключено -->
<arg name="navigation" default="false" />

<arg name="odom_frame_id" default="odom_combined" />

<!-- Включение лидара -->
<include file="$(find turn_on_wheeltec_robot)/launch/wheeltec_lidar.launch" />

<!-- Узел для сохранения карты через приложение -->
<node pkg="world_canvas_msgs" type="save" name="save_map" />
```

Описание:

- В файле **mapping.launch** можно выбрать алгоритм картографирования, задав параметр **mapping_mode**. По умолчанию используется **gmapping**, но также доступны:
  - **hector**,
  - **cartographer**,
  - **karto**.
- При выборе **cartographer** параметр **is_cartographer** автоматически устанавливается в **true**, что отключает фильтр EKF.
- Все четыре алгоритма устанавливаются в виде бинарных пакетов, исходный код недоступен для прямого просмотра. Для изучения алгоритмов можно обратиться к их репозиториям на **GitHub**.

Ссылки на репозитории GitHub для четырёх алгоритмов картографирования

Gmapping : https : / / github . com /ros-perception/slam_gmapping
Hector : https : / / github . com / tu-darmstadt-ros-pkg/hector_slam
Cartographer : https : / / github . com /cartographer-project/cartographer
Karto : https : / / github . com /ros-perception/slam_karto

3 2D Навигация: Использование и описание

3.1 Обзор функции

Функция **2D навигации** реализована с помощью **лидара** и алгоритма **SLAM**, а также основана на пакете **ROS Navigation**.

При 2D-навигации лазерный радар отображает **двумерную плоскость**.

Для навигации необходимо использовать карту, созданную ранее с помощью функции **2D картографирования**.

3.2 Инструкция по использованию

Перед началом работы:

- Убедитесь, что **верхний компьютер** подключен к **Wi-Fi робота** и выполнен **SSH-доступ** на стороне робота.

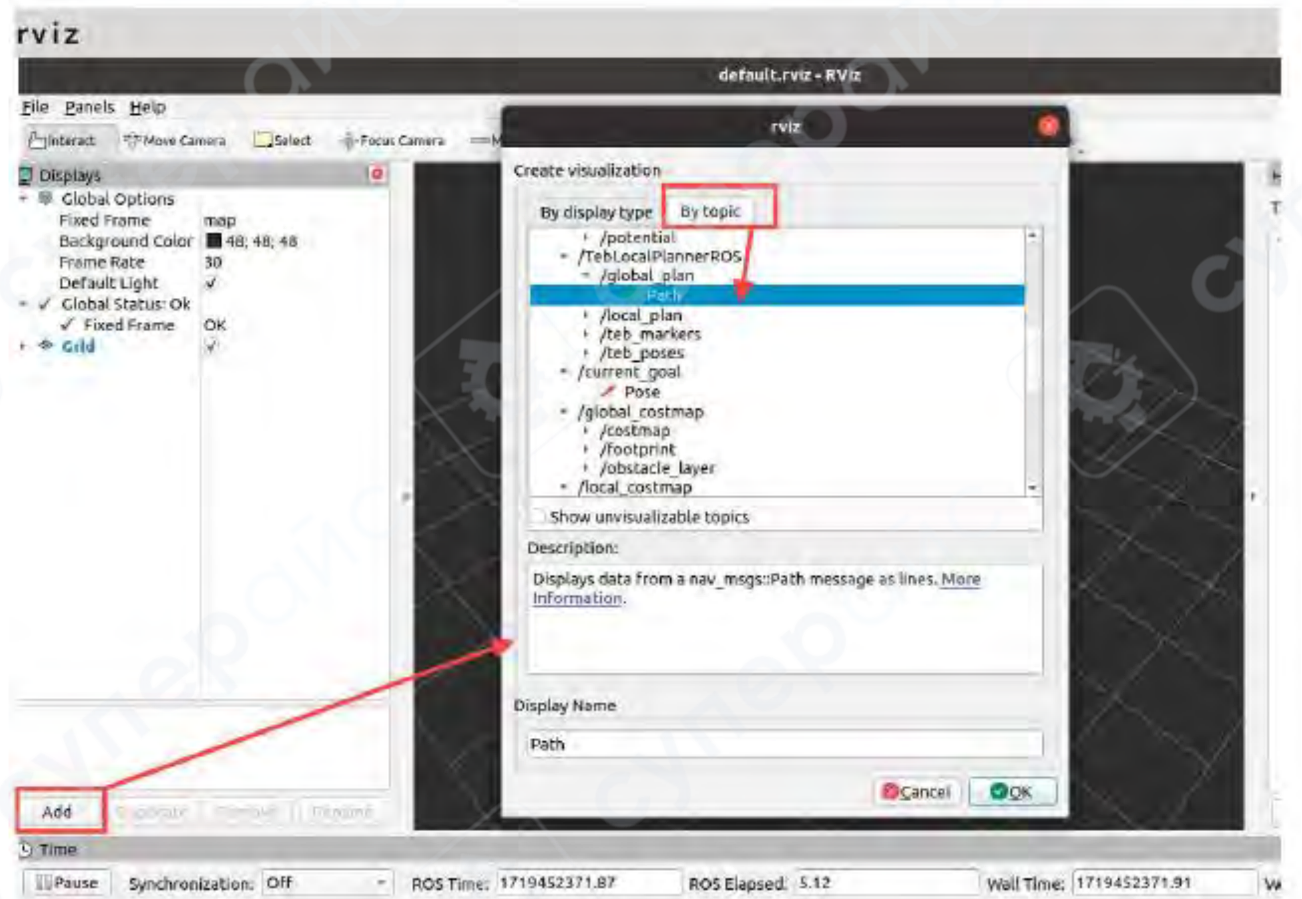
Шаги:

1. **Запуск 2D-навигации**
2. В терминале введите команду:

```
roslaunch turn_on_wheeltec_robot navigation.launch
```

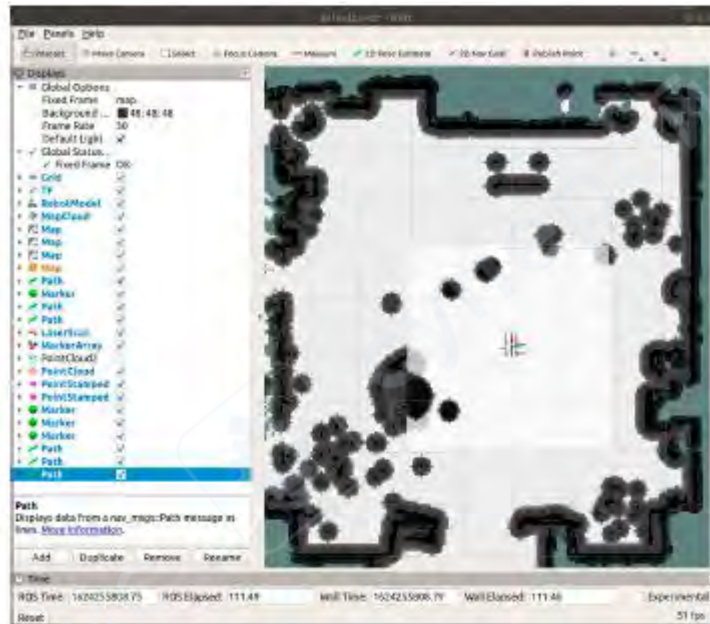
3. **Открытие RViz для визуализации**

- На верхнем компьютере откройте инструмент визуализации **RViz**.
- Нажмите кнопку **Add** → выберите **By topic**, чтобы добавить визуальные элементы через топики.



Добавление визуальных опций после запуска RViz

После добавления соответствующих топиков изображение в **RViz** будет выглядеть следующим образом:

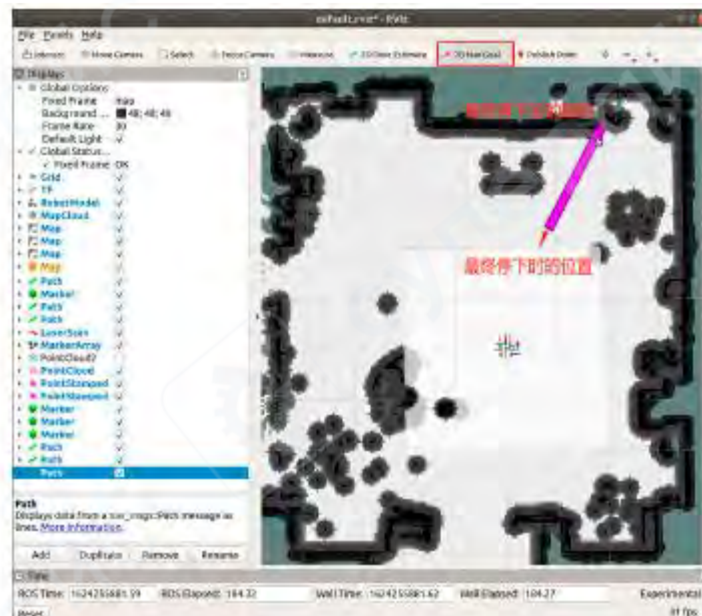


Интерфейс RViz после включения функции навигации

Режимы навигации:

1. Одноточечная навигация

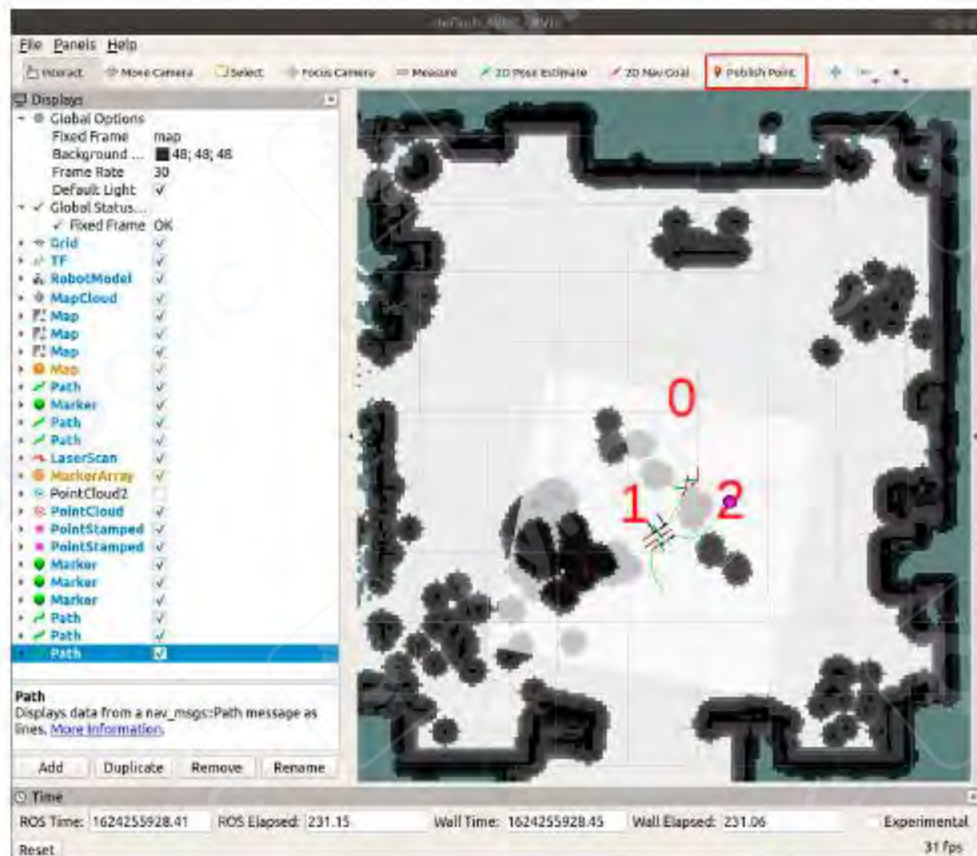
- В **RViz** нажмите на кнопку **2D Nav Goal**.
- На карте выберите **точку назначения** и **направление движения**:
  - **Нажмите левой кнопкой мыши**, чтобы выбрать точку на карте.
  - Не отпуская кнопку, **перетащите**, чтобы указать направление движения.
  - Отпустите кнопку, чтобы подтвердить выбор.
- Робот начнёт движение к заданной точке. В терминале **RViz** отобразятся координаты целевой точки.



Функция 2D-навигации — одноточечная навигация

2. Многоточечная навигация

- В **RViz** нажмите кнопку **Publish Point**, чтобы установить точки для многоточечной навигации.
- При установке **нескольких точек** робот будет двигаться **между этими точками** поочерёдно.
- **По умолчанию** направление для каждой точки совпадает с **начальным направлением** корпуса робота.



Функция 2D-навигации — многоточечная навигация

3.3 Примечания

1. Загрузка карты перед началом навигации

Для реализации функции 2D-навигации необходимо предварительно создать карту. Начальная точка навигации должна совпадать с начальной точкой построения карты. Рекомендуется выбирать заметный ориентир в качестве начальной точки при построении карты, чтобы уменьшить погрешность при размещении робота перед началом навигации. После установки робота в исходную точку карты запускайте функцию 2D-навигации. В примере используется карта WHEELTEC.yaml, созданная в процессе 2D-картографирования. После запуска RViz карта будет отображена на экране. **Важно:** RViz следует запускать только после полного запуска узла 2D-навигации, чтобы избежать ошибок загрузки карты.

2. Проблема с отсутствием движения робота

В режиме одноточечной навигации, если робот не двигается после указания целевой точки, проблема может быть связана с **неправильной настройкой межмашинного**

взаимодействия. Рекомендуется обратиться к функции «Межмашинное взаимодействие» для настройки.

Также причиной может быть **низкий уровень заряда**:

- Для **образовательного ROS-робота** заряд должен быть не ниже 10 В.
- Для **исследовательского ROS-робота** заряд должен быть не ниже 20 В.

Проверить уровень заряда можно на OLED-дисплее на плате STM32 в правом нижнем углу.

Дополнительные причины могут включать **аварийную кнопку или выключенный переключатель включения**. Статус можно проверить на дисплее контроллера (ON или OFF).

3. Повторная настройка точек многоточечной навигации

Для настройки направления точек многоточечной навигации можно обратиться к обсуждению на форуме:

http : / / bbs . wheeltec . net / forum.php?mod=viewthread&tid=2593

Если необходимо **сбросить навигационные точки**, введите команду с в терминале. После сброса робот переместится к следующей целевой точке, заданной ранее, и остановится. Затем можно заново задавать новые навигационные цели без необходимости перемещения робота вручную.

4. Проблема с расширенным слоем препятствий в процессе навигации

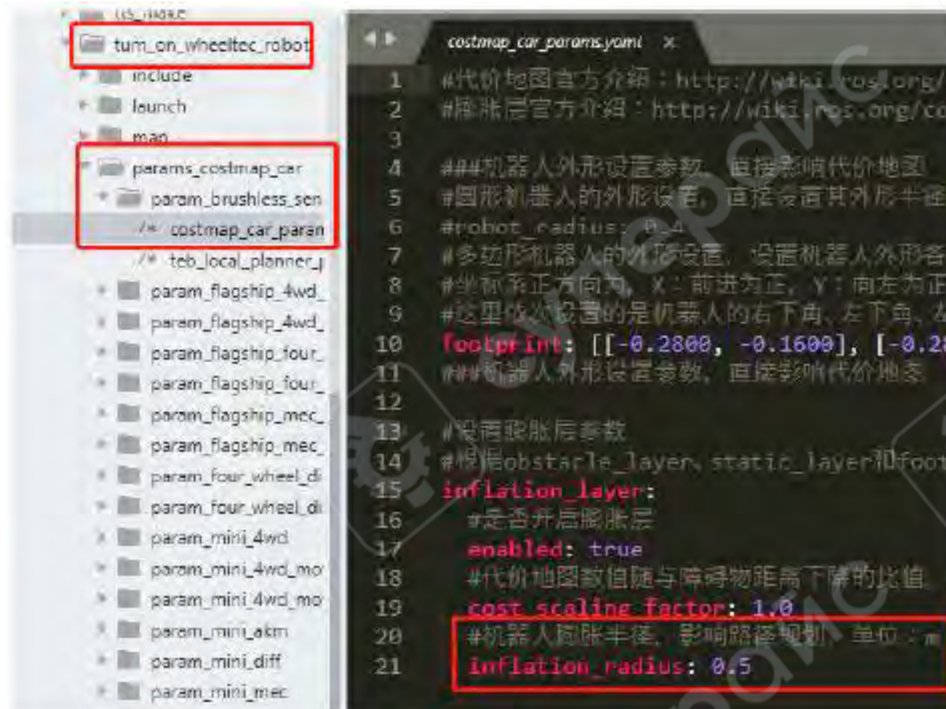
В процессе навигации робот может останавливаться на узких участках пути, даже если фактически он может пройти через промежуток между препятствиями. Это происходит из-за **расширенного радиуса препятствий** (inflation radius), который по умолчанию воспринимает область вокруг препятствия как непроходимую.

Для решения этой проблемы необходимо изменить параметр **inflation radius** в файле

/turn_on_wheeltec_robot/params_costmap_car/param_【carmode】/costmap_car_params.yaml.

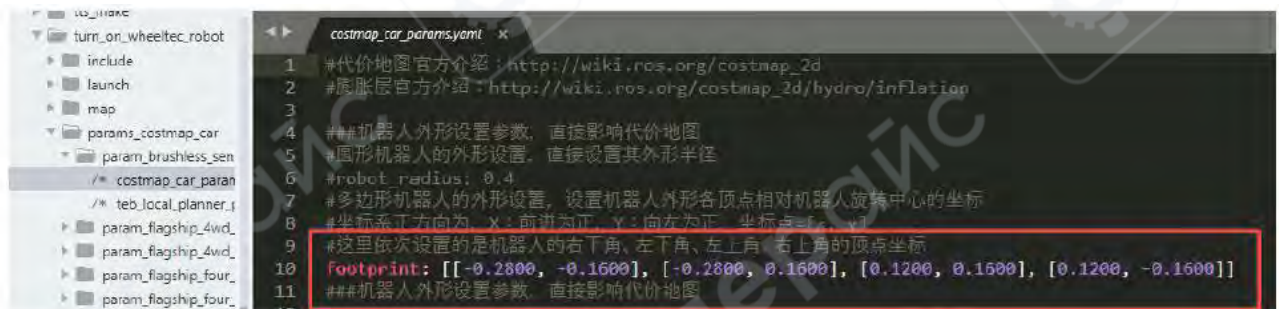
Примечание:

- 【carmode】 обозначает модель робота (выбрана перед отправкой пользователю).
- Проверьте модель робота в файле turn_on_wheeltec_robot.launch.



Изменение радиуса расширения (Inflation Radius)

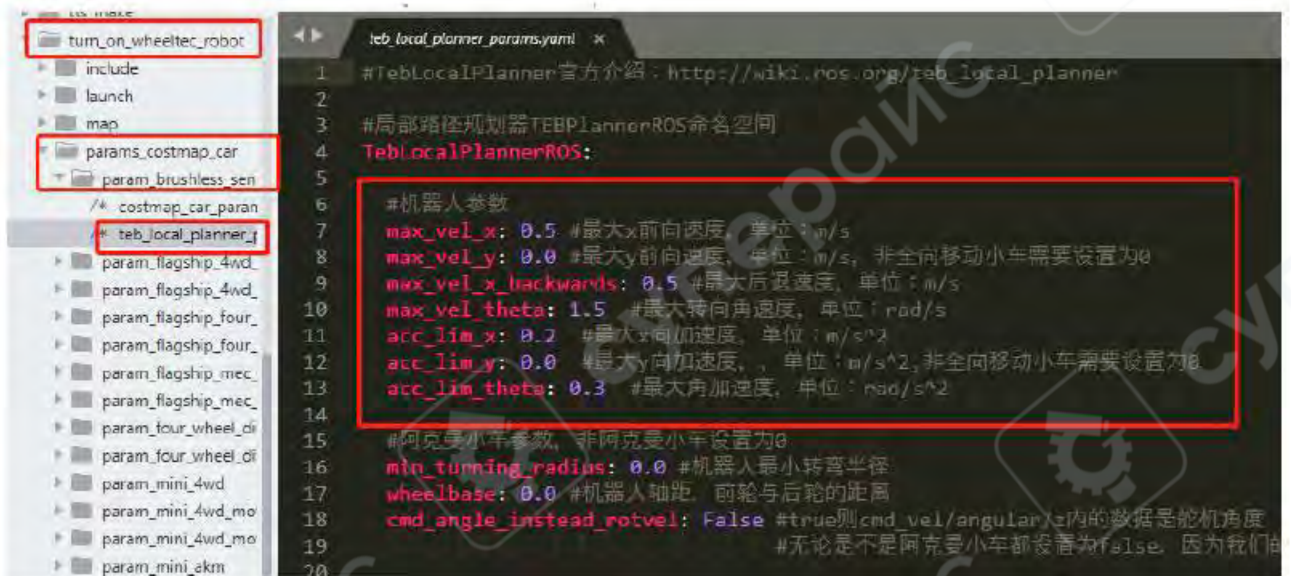
Если **расширенный радиус** меньше ширины робота, он не будет применяться. В этом случае учитывается **обводка робота** (footprint), также определяемая в том же файле. При необходимости можно изменить **footprint**, но это **не рекомендуется** из-за повышенного риска столкновений на узких участках.



Обводка робота (Footprint)

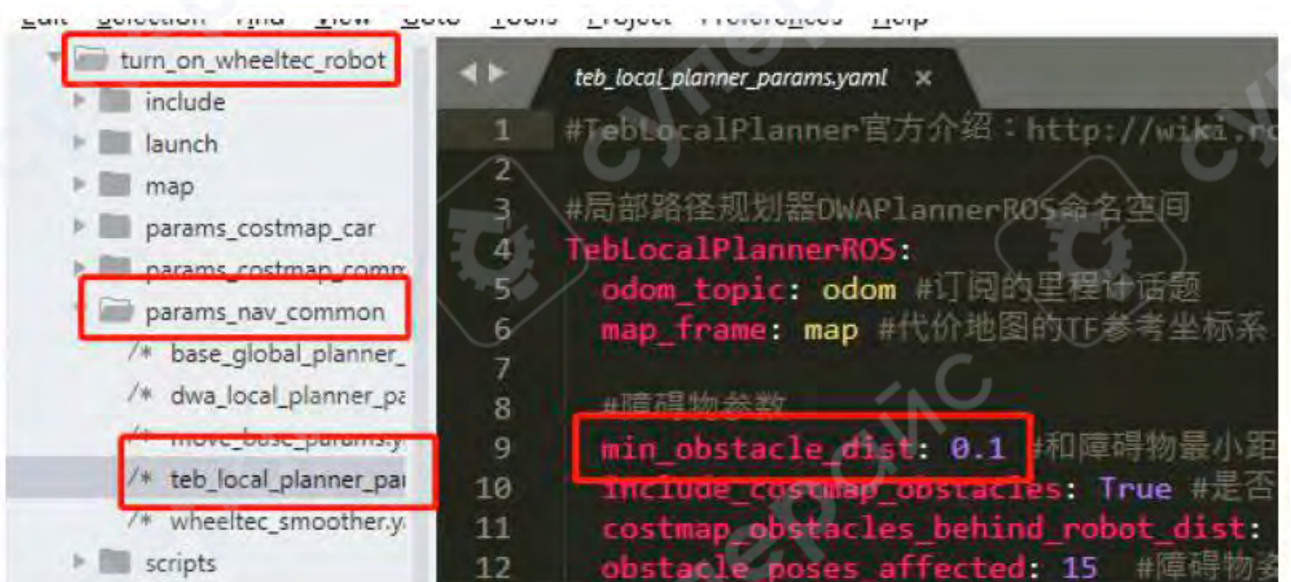
Дополнительные параметры:

- **Локальный планировщик пути**: В исходном коде по умолчанию используется **алгоритм TEB**. Параметры скорости можно изменить в файле:
- /turn_on_wheeltec_robot/params_costmap_car/param_【carmode】/teb_local_planner_params.yaml.



Изменение скорости навигации

- **Параметр min_obstacle_dist**: Определяет минимальное расстояние до препятствий относительно внешнего контура робота.



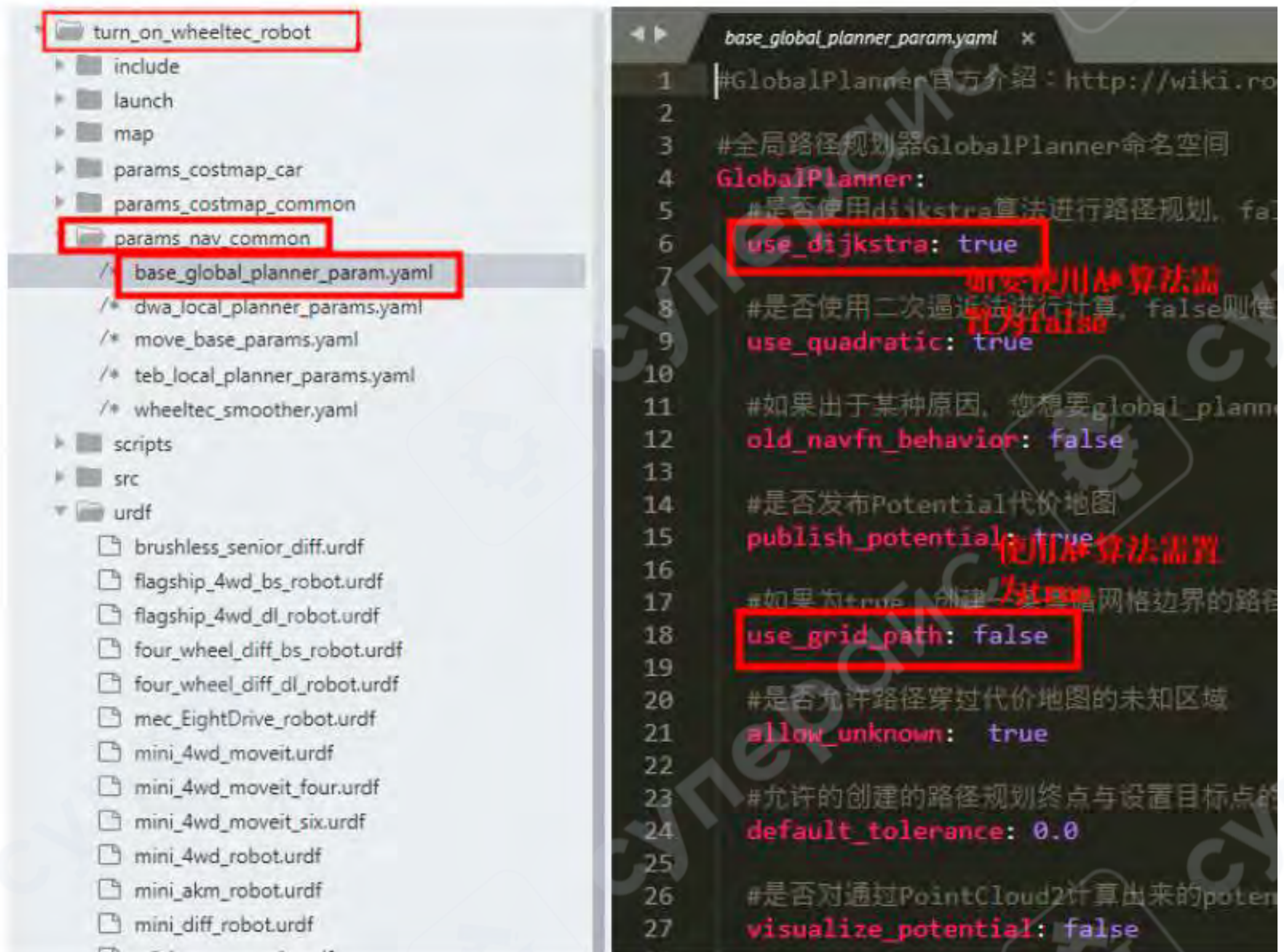
Параметры формы робота и минимального расстояния до препятствий

Смена глобального планировщика пути

Параметры глобального планировщика задаются в файле: /turn_on_wheeltec_robot/param_nav_common/base_global_planner_params.yaml.

По умолчанию используется алгоритм **Dijkstra**. Чтобы переключиться на **A***:

- Измените параметр use_dijkstra на false.
- Измените параметр use_grid_path на противоположное значение (true).

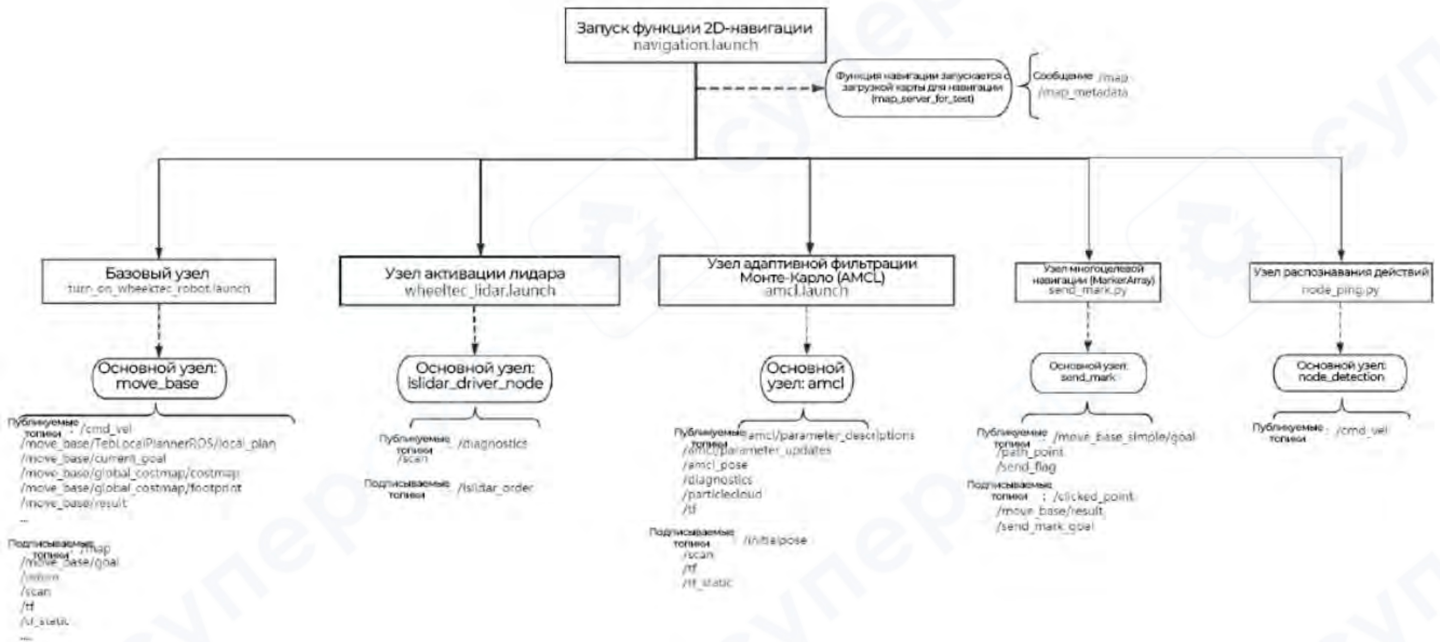


Переключение глобального алгоритма планирования пути

3.4 Объяснение функционала

Запуск узлов функции 2D-навигации осуществляется путем выполнения файла navigation.launch, расположенного в пакете turn_on_wheeltec_robot.

1. Запуск функции 2D-навигации

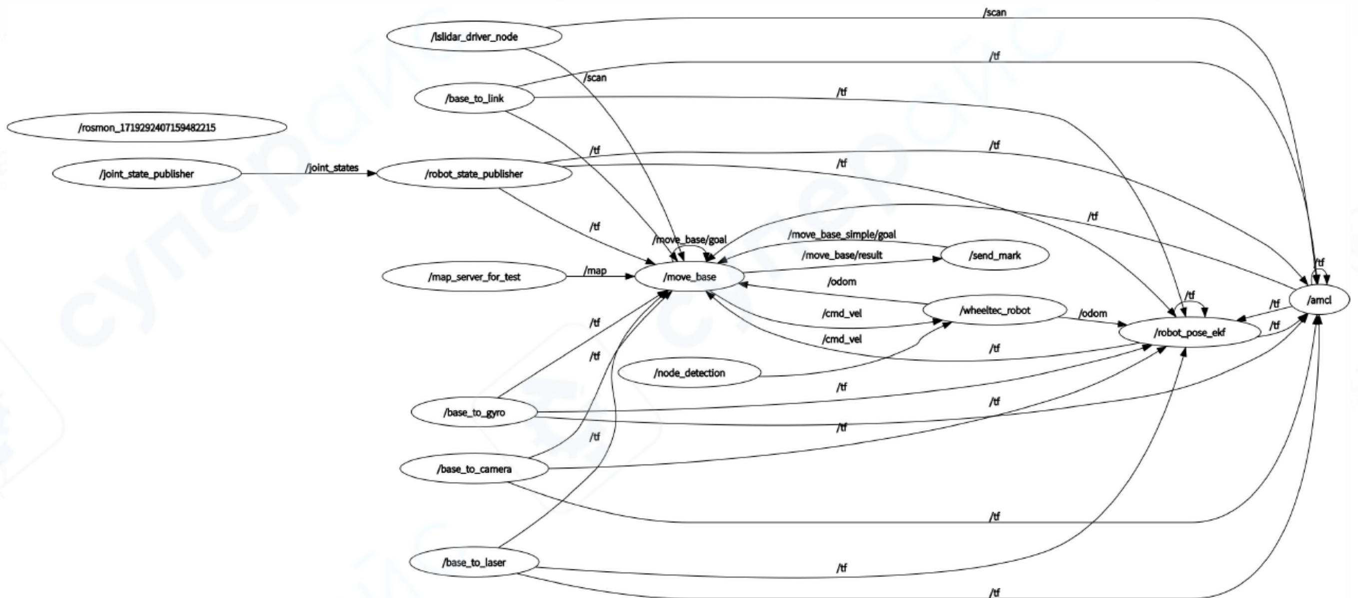


2D навигационная функция – структура запуска

2. Отношения между узлами функции 2D-навигации

Команда для просмотра узлов в графическом представлении:

```
rqt_graph
```

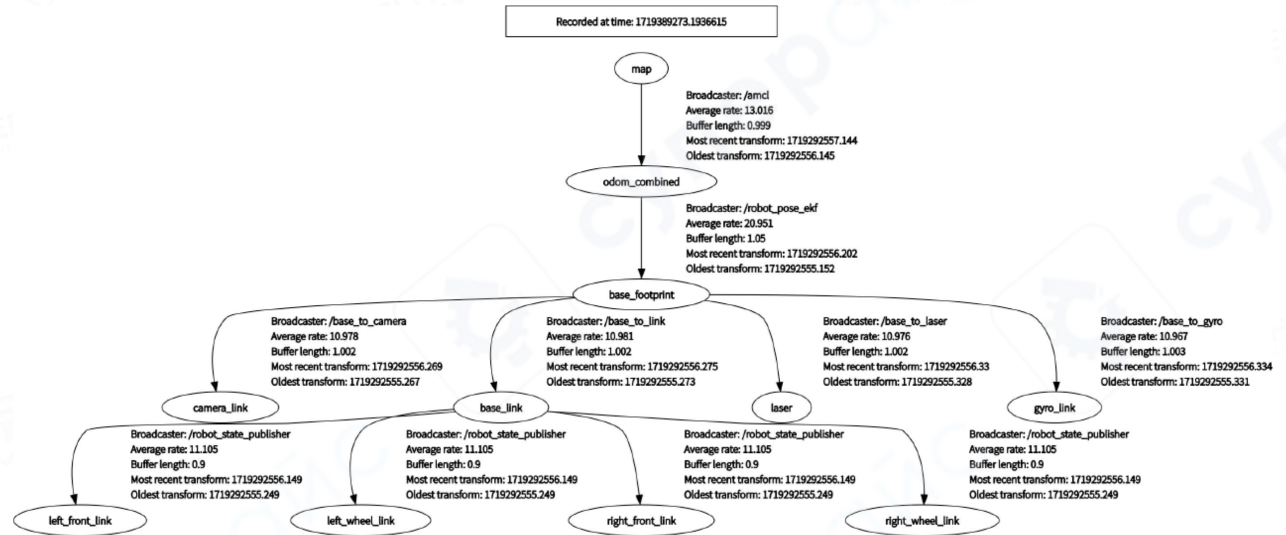


Граф узлов навигации

3. TF-дерево функции 2D-навигации

Команда для просмотра TF-дерева:

```
rosrun rqt_tf_tree rqt_tf_tree
```



TF-дерево 2D-навигации

4. Описание файла запуска функции 2D-навигации

```
<launch>
   <!-- Запуск базовых узлов робота с включением функции навигации -->
   <include file="$(find turn_on_wheeltec_robot)/launch/turn_on_wheeltec_robot.launch">
      <arg name="navigation" default="true"/>
   </include>

   <!-- Включение лидарного узла -->
   <include file="$(find turn_on_wheeltec_robot)/launch/wheeltec_lidar.launch" />

   <!-- Настройка карты для навигации -->
   <arg name="map_file" default="$(find turn_on_wheeltec_robot)/map/WHEELTEC.yaml"/>
   <node    name="map_server_for_test"    pkg="map_server"    type="map_server"    args="$(arg
map_file)">
   </node>

   <!-- Включение адаптивного Монте-Карло локализатора (AMCL) -->
   <include file="$(find turn_on_wheeltec_robot)/launch/include/amcl.launch" />

   <!-- Узел для работы с многоцелевой навигацией (MarkerArray) -->
   <node name='send_mark' pkg="turn_on_wheeltec_robot" type="send_mark.py">
   </node>
</launch>
```

Описание структуры:

1. **Запуск базовых узлов и навигации:**

- Включаются базовые узлы робота с параметром navigation, установленным в true.
- В файле turn_on_wheeltec_robot.launch задаются параметры навигационного алгоритма.

2. **Настройка карты для навигации:**
   - В параметре map_file задается путь к YAML-файлу карты (по умолчанию WHEELTEC.yaml).
   - Можно изменить путь и название файла для использования другой карты.
3. **Адаптивная Монте-Карло локализация (AMCL):**
   - Узел AMCL обеспечивает функцию локализации, публикуя данные о положении в виде трансформаций /tf.
4. **Многоцелевая навигация:**
   - Запускается узел send_mark.py, который отвечает за публикацию точек для многоточечной навигации.
   - Файл расположен в директории turn_on_wheeltec_robot.

4. Функция визуального слежения: использование и пояснение

4.1 Краткое описание функции

Функция визуального слежения позволяет роботу отслеживать объект определённого цвета. Эта функция реализуется с помощью аппаратного обеспечения камеры робота **WHEELTEC** и использует пакет **cv_bridge** в **ROS** для передачи изображений в **OpenCV**. OpenCV наделяет ROS мощными возможностями обработки изображений, что позволяет выполнять широкий спектр задач визуальной обработки.

4.2 Способ использования

Перед началом работы необходимо убедиться, что:

- Ваш компьютер подключён к WiFi-сети робота.
- Вы выполнили удалённый вход на робот через SSH.

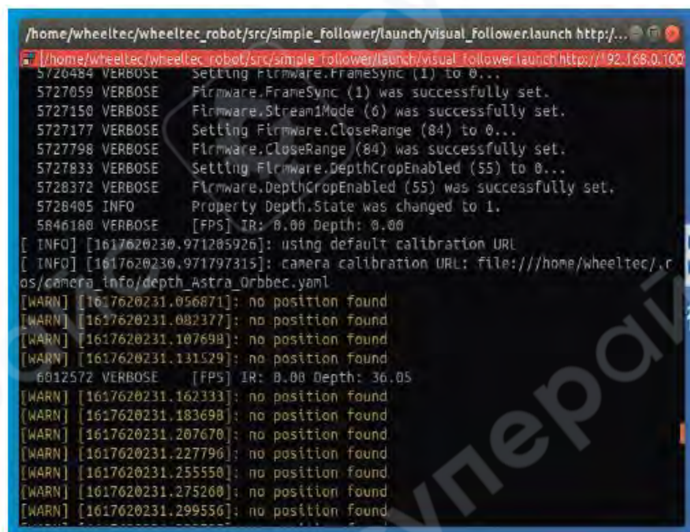
1. Запуск функции визуального слежения

После удалённого входа через SSH используйте следующую команду для запуска функции из пакета **simple_follower**:

```
roslaunch simple_follower visual_follower.launch
```

Если в терминале отсутствуют сообщения об ошибках (выделенные красным), запуск прошёл успешно.

После выполнения команды робот начнёт искать целевой объект. По умолчанию целевым объектом для визуального слежения является **красный объект**. Робот вычисляет скорость движения на основе положения целевого объекта и выполняет его слежение.



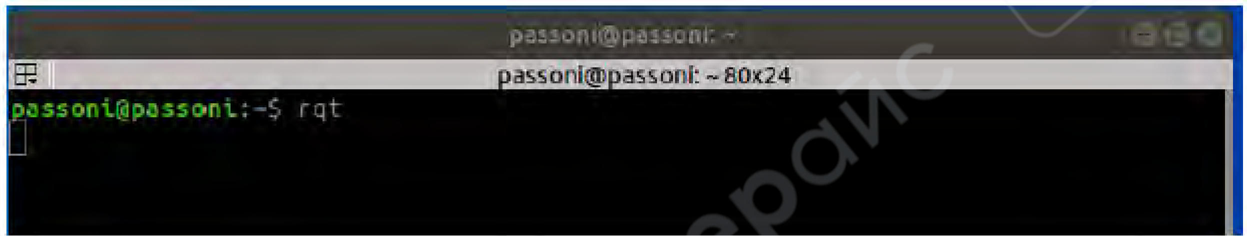
Информация в терминале после запуска узла: Робот начинает поиск целевого объекта.

2. Онлайн-настройка параметров с помощью rqt

После успешного запуска visual_follower.launch откройте новый терминал и введите команду:

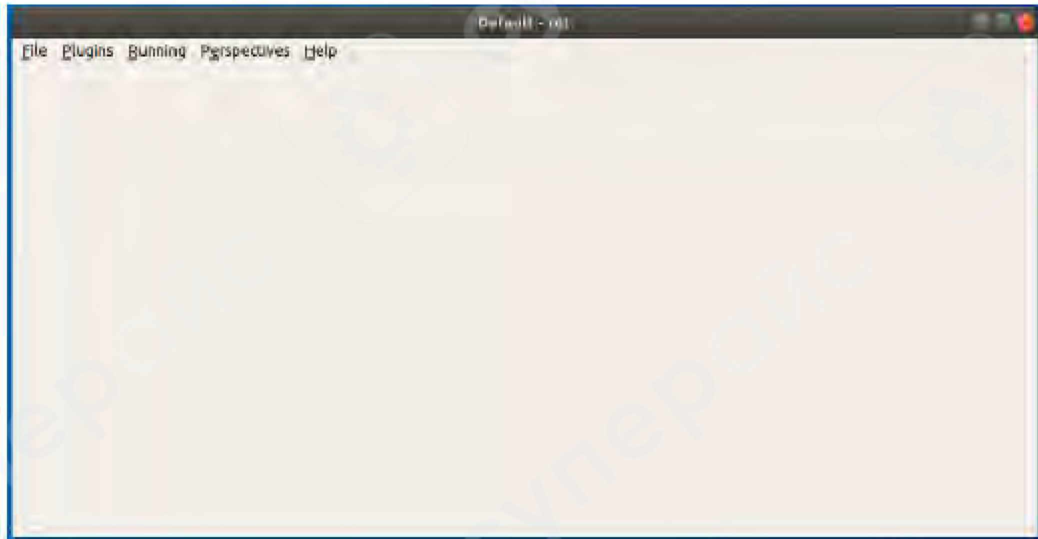
```
rqt
```

(Примечание: для этого **не требуется** повторный вход через SSH на роботе).



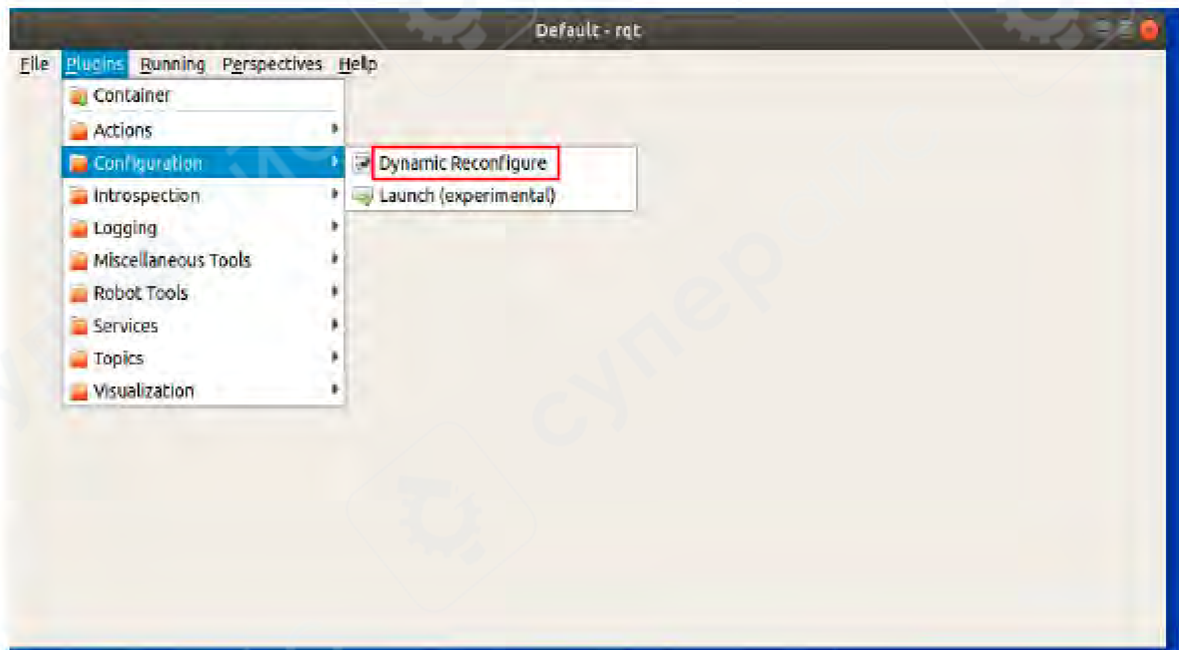
Ввод команды rqt в новом терминале

После запуска rqt откроется следующий интерфейс:



Состояние rqt по умолчанию

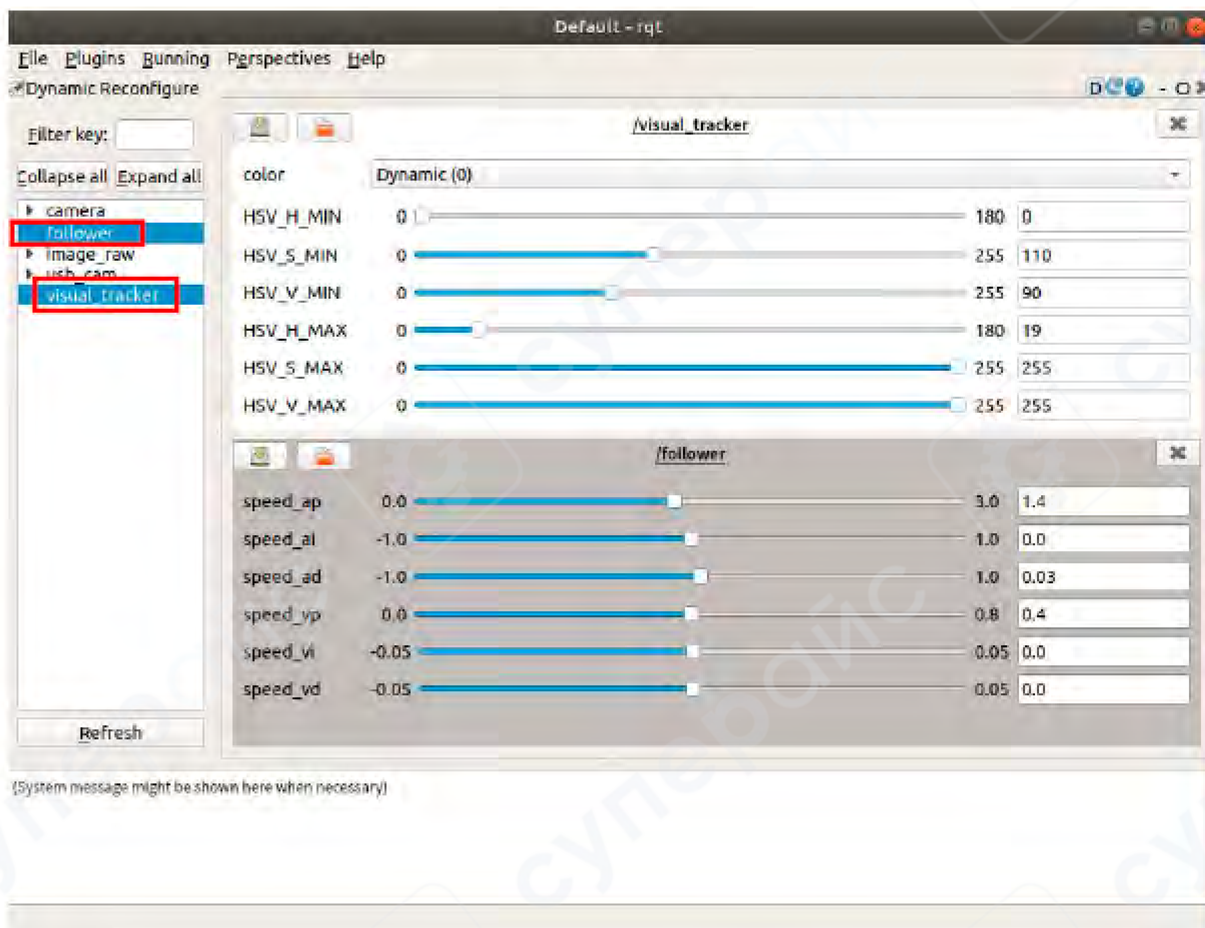
1. В верхнем меню выберите **Plugins** → **Configuration** → **Dynamic Reconfigure**.



Открытие инструмента динамической настройки параметров (Dynamic Reconfigure)

2. В списке выберите параметры **follower** и **visual_tracker**.

Откроются интерфейсы для динамической настройки узлов **follower** и **visual_tracker**.



Интерфейс динамической настройки параметров

3. Настройка параметров

На данном этапе можно изменять параметры, связанные с целевым цветом для слежения или настройкой **PID-скорости** робота. Параметры можно регулировать, используя:

- Ползунки (перемещая бегунок).
- Поля ввода для конкретных значений (введите значение и нажмите Enter).

Описание параметров для узла follower

Таблица 1: Описание параметра speed.

Имя параметра	Описание
speed_ap	Параметр угловой скорости P
speed_ai	Параметр угловой скорости I
speed_ad	Параметр угловой скорости D
speed_vp	Параметр линейной скорости P
speed_vi	Параметр линейной скорости I
speed_vd	Параметр линейной скорости D

Описание параметров для узла visual_tracker

Таблица 2: Соответствие параметра color

Значение параметра	Цвет
0	Динамически настраиваемый
1	Красный

2	Синий
3	Зелёный
4	Жёлтый

Таблица 3: Описание параметров HSV

Имя параметра	**Описание**
HSV_H_MIN	Минимальное значение оттенка (H) целевого объекта
HSV_H_MAX	Максимальное значение оттенка (H) целевого объекта
HSV_S_MIN	Минимальное значение насыщенности (S) целевого объекта
HSV_S_MAX	Максимальное значение насыщенности (S) целевого объекта
HSV_V_MIN	Минимальное значение яркости (V) целевого объекта
HSV_V_MAX	Максимальное значение яркости (V) целевого объекта

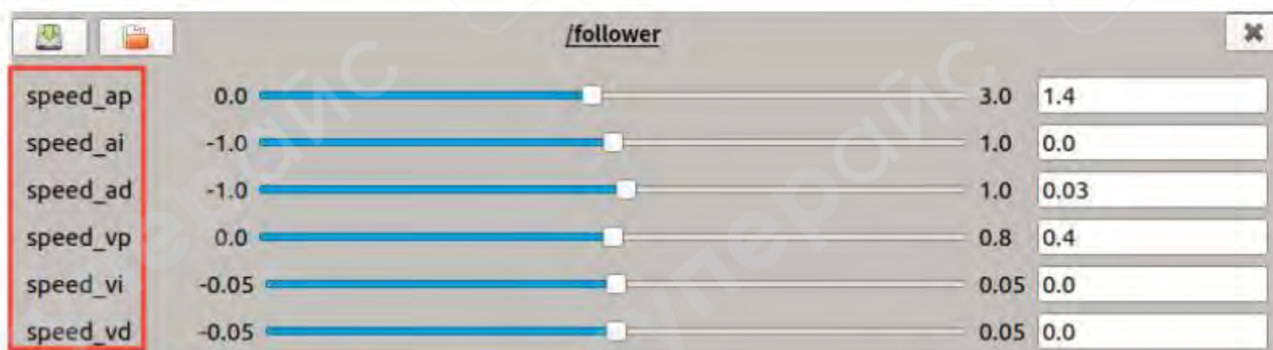
4.3 Примечания

1. Проблема движения робота

Если робот двигается вперёд-назад повторно, это может быть вызвано неправильно настроенными параметрами **PID**. При использовании **динамического инструмента настройки параметров** (rqt_reconfigure) можно в реальном времени отрегулировать параметры **PID** для скорости.

Рекомендации при настройке параметров PID:

- Избегайте слишком больших значений, так как это может привести к нестабильности робота и столкновениям.
- Во время работы функции визуального слежения робот **не обладает функцией автономного обхода препятствий**. Убедитесь, что область движения робота свободна от лишних предметов, чтобы избежать столкновений.



Регулировка параметров PID для скорости

2. Настройка параметров визуального слежения

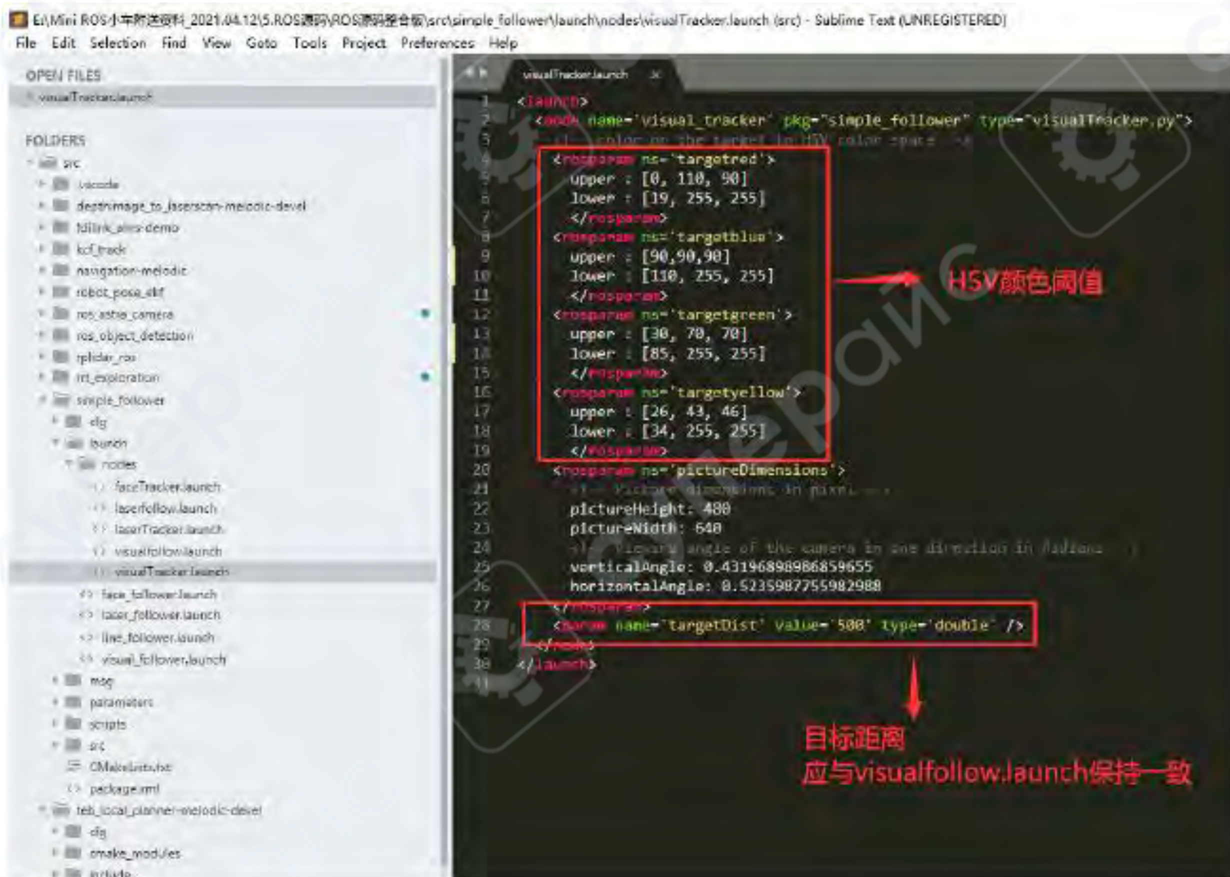
Настройку параметров можно выполнить двумя способами:

1. **Редактирование файла запуска (launch-файла):**
   - Параметры можно изменить напрямую в **launch-файле** узла, не требуя перекомпиляции кода. Изменения вступят в силу после **перезапуска узла**.
   - В файле visualfollow.launch можно изменить два ключевых параметра:
     - **maxSpeed** (максимальная скорость).
     - **targetDist** (целевое расстояние).

Примечание: Не изменяйте **maxSpeed** и **targetDist**, если в этом нет необходимости. Эти параметры уже имеют строго заданные ограничения.

Launch-файл узла follower

- В файле visualTracker.launch можно изменить **пороговые значения для разных цветов**.



Launch-файл узла visual_tracker

2. **Онлайн-настройка через rqt:**
- Параметры можно изменить во время работы узла с помощью **rqt**.
- Изменённые параметры будут действовать **только в текущем сеансе** работы узла.
- Онлайн-настройка rqt может использоваться как **инструмент отладки**.

Дополнительные ресурсы:

Пример использования dynamic_reconfigure: http : / / wiki . ros . org/dynamic_reconfigure/Tutorials

Настройка цвета целевого объекта:

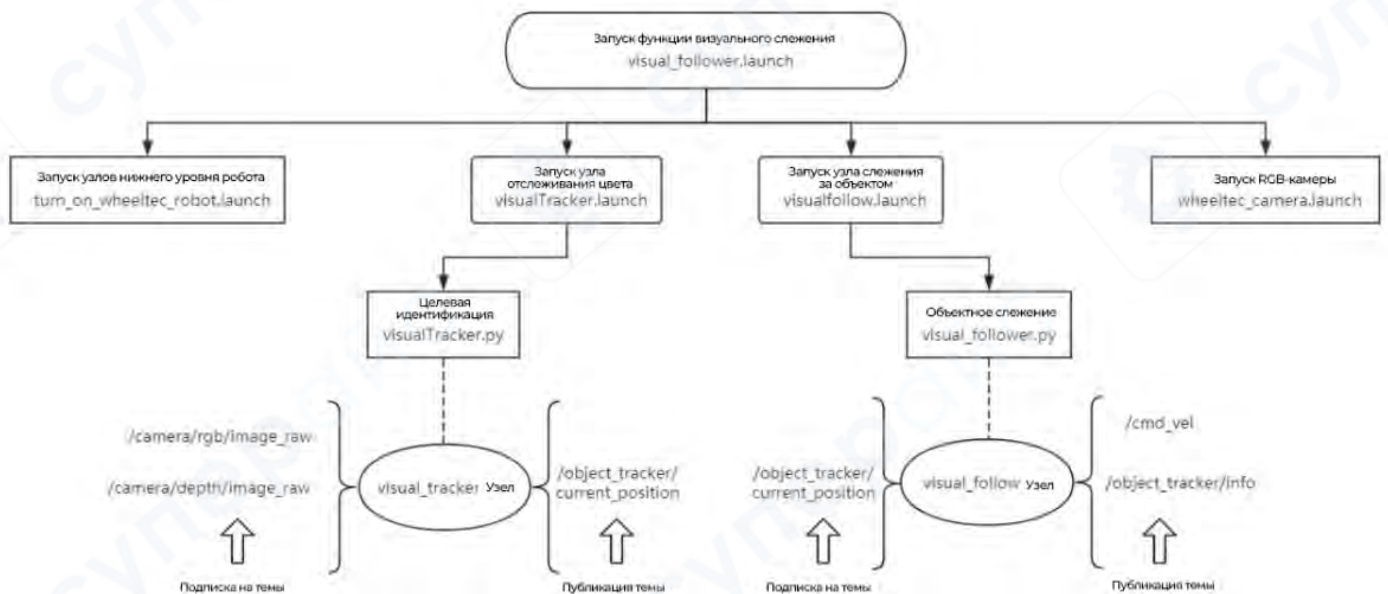
Для настройки цвета используется **HSV-модель**, поскольку обработка изображений в цветовом пространстве чаще всего выполняется именно в **HSV-пространстве**.

- Для каждого базового цвета необходимо задать **строго ограниченные диапазоны HSV-компонентов**.

4.4 Объяснение функционала

1. Каркас запуска функции визуального слежения

Запуск визуального слежения.



Каркас запуска функции визуального слежения

2. Отношения между узлами функции визуального слежения

Граф узлов rqt_graph

Граф отношений узлов функции визуального слежения

3. TF-дерево функции визуального слежения

Используется команда:

```
rosrun rqt_tf_tree rqt_tf_tree
```

Она позволяет отобразить TF-дерево, представляющее пространственные отношения между элементами.

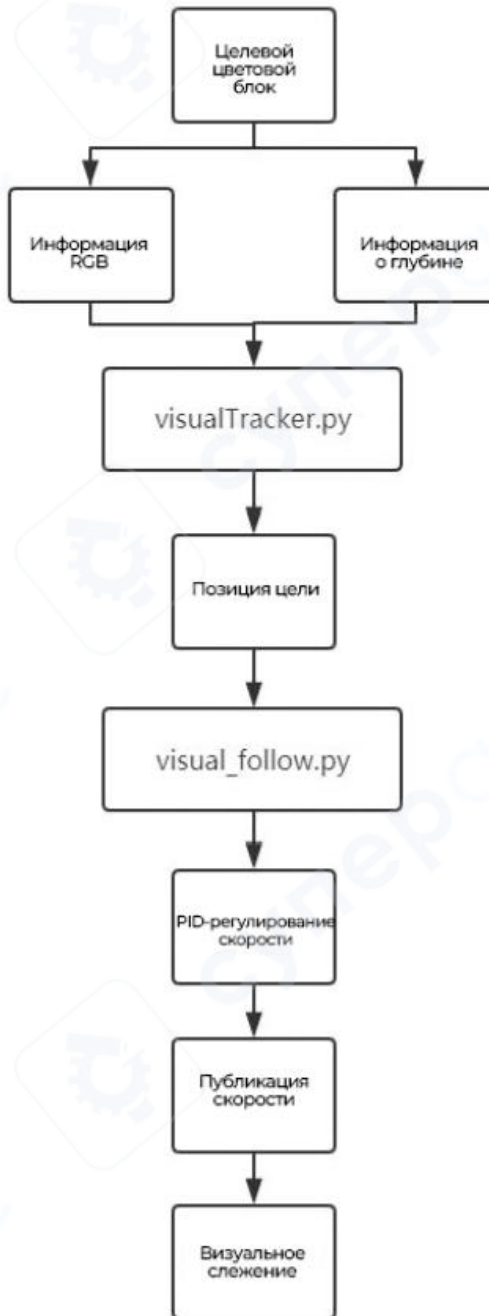
TF-дерево функции визуального слежения

4. Анализ процесса работы функции визуального слежения

Функция визуального слежения запускается через файл visual_follower.launch. Этот файл включает **пять launch-файлов**:

1. **turn_on_wheeltec_robot.launch** – запускает узлы нижнего уровня управления движением робота.
2. **wheeltec_camera.launch** – отвечает за активацию **RGB-камеры** и **камеры глубины**.
3. **visualTracker.py** – выполняет **распознавание целей**.
4. **visualfollow.py** – выполняет **слежение за объектом**.

Общий процесс работы функции визуального слежения:



Анализ процесса работы функции визуального слежения

5. Анализ ключевого исходного кода функции визуального слежения в файле visualTracker.py

Начнём с основной части **визуального распознавания** в файле visualTracker.py:

Узел visual_tracker одновременно подписывается на темы **RGB-камеры** и **камеры глубины**, а затем передаёт данные в функцию обратного вызова trackObject.

```
# message_filters подписка на тему RGB-камеры
im_sub = message_filters.Subscriber('/camera/rgb/image_raw', Image)

# Подписка на тему камеры глубины
dep_sub = message_filters.Subscriber('/camera/depth/image_raw', Image)

# Синхронизация изображений RGB и глубины по времени с небольшим допустимым смещением
self.timeSynchronizer = message_filters.ApproximateTimeSynchronizer([im_sub, dep_sub], 10, 0.5)

# Передача синхронизированных данных в функцию trackObject
self.timeSynchronizer.registerCallback(self.trackObject)

# Публикация текущей позиции цели
self.positionPublisher = rospy.Publisher('/object_tracker/current_position', PositionMsg, queue_size=3)

# Вызов динамического сервера параметров
self.color_obj = Server(Params_colorConfig, self.colorreconfigure)
```

Подробнее о message_filters:

- **message_filters.Subscriber** – обёртка подписчика ROS, используемая в качестве источника данных для других фильтров.
- **TimeSynchronizer** синхронизирует входящие данные на основе временных меток и передаёт их в **одну функцию обратного вызова**.
- **ApproximateTimeSynchronizer** – аналогичен TimeSynchronizer, но имеет дополнительный параметр для допуска по времени.

Обработка данных в функции trackObject

После передачи изображений в функцию **trackObject**, выполняется их обработка для обнаружения цели и определения её положения.

```
# Конвертация данных RGB-камеры в формат OpenCV
frame = self.bridge.imgmsg_to_cv2(image_data, desired_encoding='rgb8')

# Конвертация данных камеры глубины в формат OpenCV
depthFrame = self.bridge.imgmsg_to_cv2(depth_data, desired_encoding='passthrough')

# Преобразование RGB-изображения в HSV-формат
hsv = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)
```

```
        # Удаление фона и выделение областей нужного цвета по пороговым значениям
        org_mask = cv2.inRange(hsv, self.targetUpper, self.targetLower)

        # Морфологическая операция эрозии для удаления шумов и помех
        mask = cv2.erode(org_mask, None, iterations=4)

        # Поиск контуров на изображении и выделение внешних границ
        contours = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]

        newPos = None

        try:
            # Сортировка контуров и выбор самого большого по площади в качестве цели
            contour = sorted(contours, key=cv2.contourArea, reverse=True)[0]

            # Определение центра и расстояния до цели
            pos = self.analyseContour(contour, depthFrame)

            if newPos is None:
                newPos = pos

            self.lastPosition = pos
            self.publishPosition(pos)
            self.lastPosition = newPos

        except IndexError:
            # Логирование предупреждения при ошибке поиска
            rospy.logwarn('no position found')

            # Остановка робота при отсутствии цели
            posMsg = PositionMsg(0, 0, self.targetDist)
            self.positionPublisher.publish(posMsg)
```

Ключевые этапы обработки:

1. **Конвертация изображений:**
    - RGB-изображение конвертируется в HSV-формат для выделения целевого цвета.
    - Глубинное изображение используется для получения расстояния до цели.
2. **Выделение цели:**
    - **cv2.inRange** применяется для установки порогов и выделения объектов нужного цвета.
    - **Эрозия** уменьшает шумы и устраняет мелкие помехи на изображении.
3. **Поиск контуров:**
    - Контуры на изображении сортируются, и самый крупный контур принимается за цель.

- Положение цели определяется функцией **analyseContour**.

4. **Публикация положения цели:**
   - Найденная позиция публикуется в тему /object_tracker/current_position.
   - Если цель не найдена, робот останавливается путём публикации нулевой скорости.

Общее взаимодействие:

После обработки изображения и определения позиции целевой точки функция visual_follow берёт на себя задачу слежения за целью. Если цель не найдена, робот останавливается для предотвращения непредсказуемых движений.

6. Анализ ключевого исходного кода функции слежения visual_follow.py

Узел **visual_follow** подписывается на тему с текущей позицией цели, которая обработана **visual_tracker**, и, используя **PID-контроль**, публикует команды для управления скоростью робота.

Основные компоненты кода visual_follow.py:

```
# Публикация темы для управления скоростью
self.cmdVelPublisher = rospy.Publisher('/cmd_vel', Twist, queue_size=3)

# Подписка на тему с текущей позицией цели
self.positionSubscriber = rospy.Subscriber('/object_tracker/current_position',
                                PositionMsg,
                                self.positionUpdateCallback)

# Подписка на информационную тему об отслеживании
self.trackerInfoSubscriber = rospy.Subscriber('/object_tracker/info',
                                 StringMsg,
                                 self.trackerInfoCallback)

# Вызов динамического сервера параметров PID
self.speed_PID = Server(Params_PIDConfig, self.followreconfigure)
```

PID-контроль скорости

Функция **update** реализует классический PID-контроль:

```
# Вычисление ошибки
error = self.setPoint - current_value

# Если ошибка мала, движение останавливается
if error[0] < 0.1 and error[0] > -0.1:
   error[0] = 0
if error[1] < 100 and error[1] > -100:
   error[1] = 0

# Масштабирование ошибки при малом расстоянии до цели
if (error[1] > 0 and self.setPoint[1] < 1200):
   error[1] = error[1] * (1200 / self.setPoint[1]) * 0.7
```

```
# Расчёт PID-компонентов
P = error
currentTime = time.clock()
deltaT = (currentTime - self.timeOfLastCall)
self.integrator = self.integrator + (error * deltaT)
I = self.integrator
D = (error - self.last_error) / deltaT

# Обновление переменных
self.last_error = error
self.timeOfLastCall = currentTime

# Возвращение вычисленного значения скорости
return self.Kp * P + self.Ki * I + self.Kd * D
```

Основная функция обработки positionUpdateCallback

Эта функция принимает текущую позицию цели, вызывает PID-контроль и публикует соответствующие команды скорости:

```
# Получение угла и расстояния до цели
angleX = position.angleX
distance = position.distance

# Выполнение PID-контроля и получение скорости
[uncliped_ang_speed, uncliped_lin_speed] = self.update([angleX, distance])

# Ограничение значений угловой и линейной скорости
angularSpeed = np.clip(-uncliped_ang_speed, -self.max_speed, self.max_speed)
linearSpeed = np.clip(-uncliped_lin_speed, -self.max_speed, self.max_speed)

# Публикация сообщений о скорости
velocity = Twist()
velocity.linear = Vector3(linearSpeed, 0, 0.)
velocity.angular = Vector3(0., 0., angularSpeed)
self.cmdVelPublisher.publish(velocity)
```

Описание процесса работы

1. **Подписка на темы:**
   - /object_tracker/current_position – текущая позиция цели.
   - /object_tracker/info – информационная тема.
2. **PID-контроль:**
   - Ошибка между текущей позицией и целевой позицией используется для расчёта скорости на основе **P (пропорциональной), I (интегральной)** и **D (дифференциальной)** составляющих.

o Значения скорости ограничиваются с помощью **np.clip**, чтобы оставаться в допустимых пределах.

3. **Публикация скорости:**

o Полученные значения линейной и угловой скорости отправляются в тему **/cmd_vel**, которая управляет движением робота.

Результат работы программы

Робот постоянно получает скорость через команду **/cmd_vel** и движется в направлении цели, поддерживая заданное расстояние. Если цель не найдена, робот останавливается.

5 Функция следования по линии: использование и пояснение

5.1 Краткое описание функции

Функция **следования по линии** позволяет роботу WHEELTEC двигаться вдоль траектории, основанной на определённом цвете, с использованием аппаратного обеспечения камеры. В систему добавлен узел **обхода препятствий с использованием лидара**. Во время движения:

- Если на пути обнаруживается **препятствие**, робот останавливается.
- Если в поле зрения камеры **нет линии выбранного цвета**, робот также останавливается.

5.2 Способ использования

Перед началом работы:

- Подключите компьютер к Wi-Fi робота.
- Выполните удалённый вход через **SSH** на роботе.

1) Запуск функции следования по линии

Откройте терминал и введите команду для запуска **launch-файла**:

```
roslaunch turn_on_wheeltec_robot mapping.launch
```

После успешного запуска:

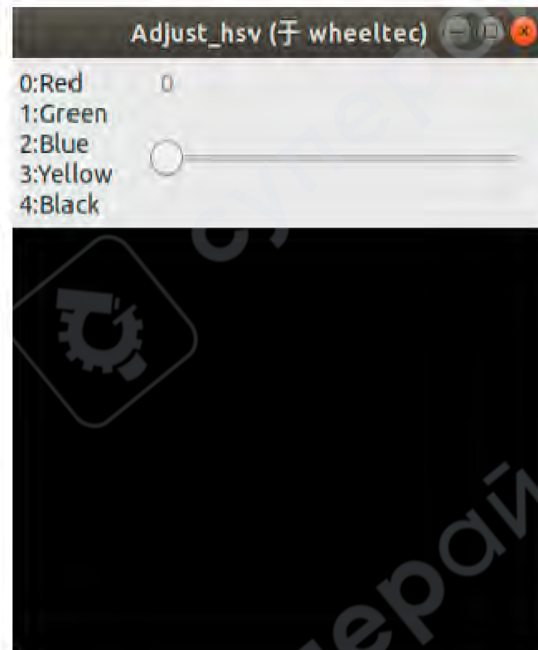
- В терминале появятся **предупреждающие сообщения** жёлтого цвета (это нормально и означает, что узлы следования по линии и обхода препятствий активированы).
- Появится **всплывающее окно**.

```
[ INFO] [1713869653.418409611]: pubscanthread
[ INFO] [1713869653.497345482]: output frame: odom_combined
[ INFO] [1713869653.529909741]: base frame: base_footprint
[ INFO] [1713869653.640009222]: Lidar is N10_P
[ INFO] [1713869653.643771000]: Opening PCAP file
port = /dev/wheeltec_lidar, baud_rate = 460800
open_port /dev/wheeltec_lidar OK !
[ INFO] [1713869653.682231759]: Initialised lslidar without error
[ INFO] [1713869653.766937963]: Initializing Odom sensor
[ INFO] [1713869653.827146648]: Odom sensor activated
[ INFO] [1713869653.850916500]: Kalman filter initialized with odom measurement
[ INFO] [1713869653.867968815]: Initializing Imu sensor
[ INFO] [1713869653.967057685]: Imu sensor activated
[ INFO] [1713869655.251911370]: Device "2bc5/0402@1/6" found.
Warning: USB events thread - failed to set priority. This might cause loss of data...
[ INFO] [1713869655.394171388]: device name: Orbbec Astra S
[WARN] [1713869655.398684]: 1
[WARN] [1713869655.513818]: laser no object found
[ INFO] [1713869656.452601943]: Starting color stream.
[ INFO] [1713869656.947524943]: using default calibration URL
[ INFO] [1713869656.947851776]: camera calibration URL: file:///home/wheeltec/.ros/camera_info
/rgb_Astra_Orbbec.yaml
```

Сообщения в терминале

2) Выбор цвета линии для следования

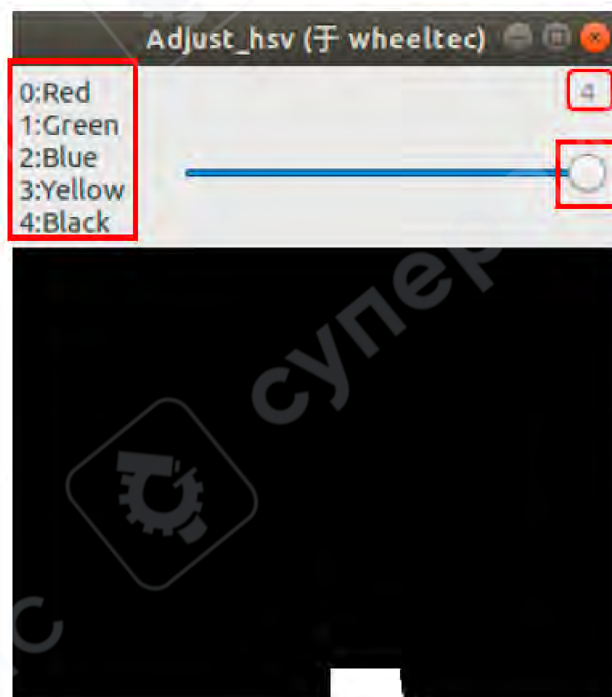
В появившемся **всплывающем окне** можно выбрать цвет линии, по которой робот будет двигаться.



Всплывающее окно

Пример:

- Для чёрного цвета переместите **ползунок** в положение **"4"**, согласно подсказке в окне.
- Цвет "4" соответствует **чёрному цвету**.



Выбор цвета

3) Просмотр изображения с камеры

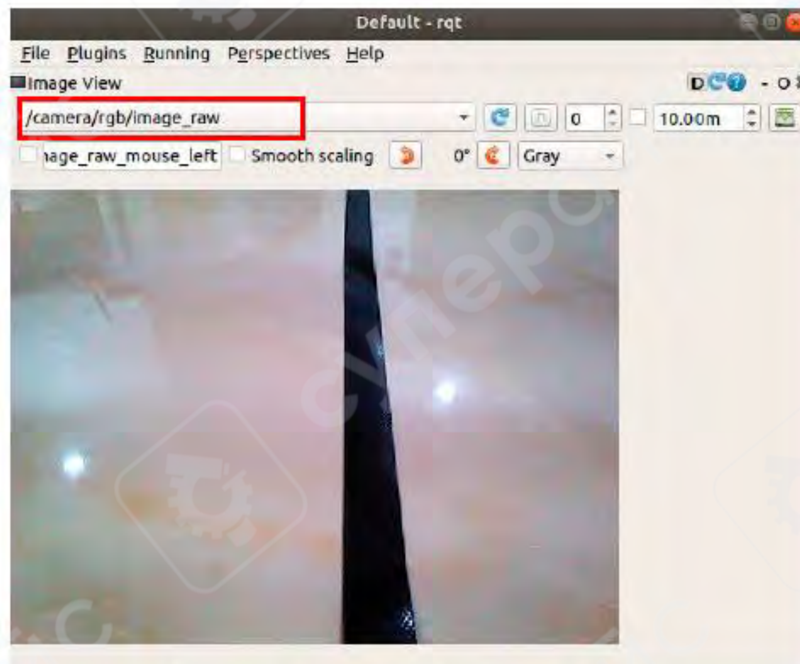
Откройте **rqt** в новом терминале для просмотра изображения с камеры:

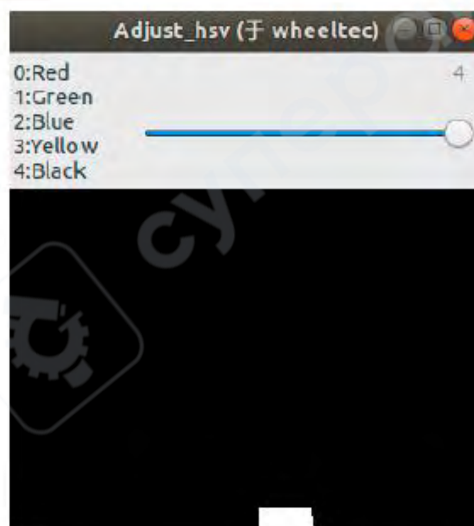
```
rqt_image_view
```

```
passoni@passoni:~$ rqt_image_view
```

Просмотр изображения с камеры через rqt_image_view

- В левом верхнем углу выберите соответствующую **RGB-тему** камеры.
- Визуализация покажет линию выбранного цвета (например, чёрную).
- Белая область на изображении указывает, что обработка данных с камеры успешно **выделила линию** выбранного цвета.
- Обратите внимание: отображаемый фрагмент линии ограничен **ближайшей частью** в поле зрения камеры, которая обрабатывается в коде.



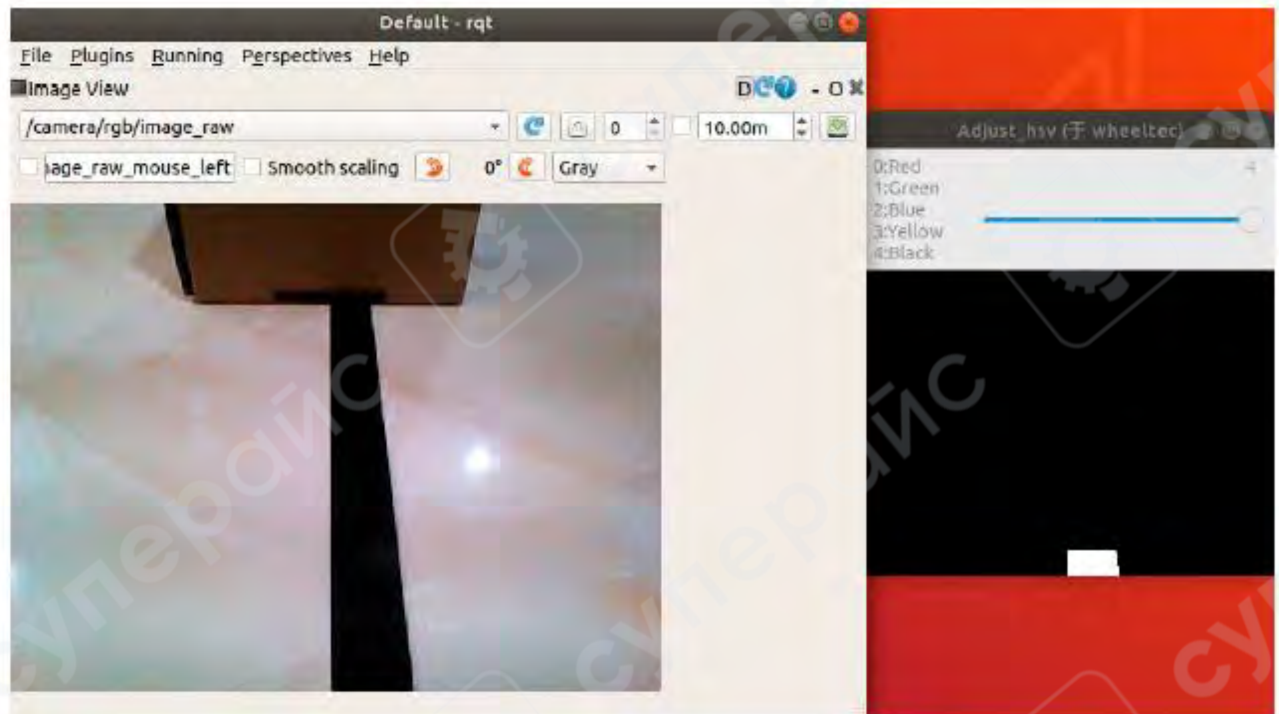


Изображение с камеры робота до и после обработки (всплывающее окно)

4) Обнаружение препятствий

Во время следования по линии:

- Если на линии обнаружено **препятствие**, лидар измеряет расстояние до него.
- Если расстояние оказывается слишком **маленьким**, система публикует **нулевую скорость**, останавливая движение робота.



Ситуация с препятствием впереди

5.3 Объяснение программы следования по линии

Функция **следования по линии** запускается с помощью файла **line_follower.launch**. Этот **launch-файл** довольно простой и предназначен для запуска различных узлов, которые совместно обеспечивают выполнение функций.

Содержимое файла line_follower.launch:

```
<launch>
  <!-- Запуск RGB-камеры -->
  <include file="$(find turn_on_wheeltec_robot)/launch/wheeltec_camera.launch" />

  <!-- Запуск узла следования по линии -->
  <node name="line_tracker" pkg="simple_follower" type="line_follow.py" />

  <!-- Запуск узла обхода препятствий -->
  <node pkg="simple_follower" type="avoidance" name="avoidance" />

  <!-- Запуск узлов нижнего уровня управления роботом -->
  <include file="$(find turn_on_wheeltec_robot)/launch/turn_on_wheeltec_robot.launch" />

  <!-- Запуск лидара (SLAMTEC) -->
```

```
    <include file="$(find turn_on_wheeltec_robot)/launch/wheeltec_lidar.launch" />

    <!-- Дополнительный узел для обработки данных лидара -->
    <include file="$(find simple_follower)/launch/nodes/laserTracker.launch" />
</launch>
```

Содержимое файла line_follower.launch

① Каркас запуска функции следования по линии:

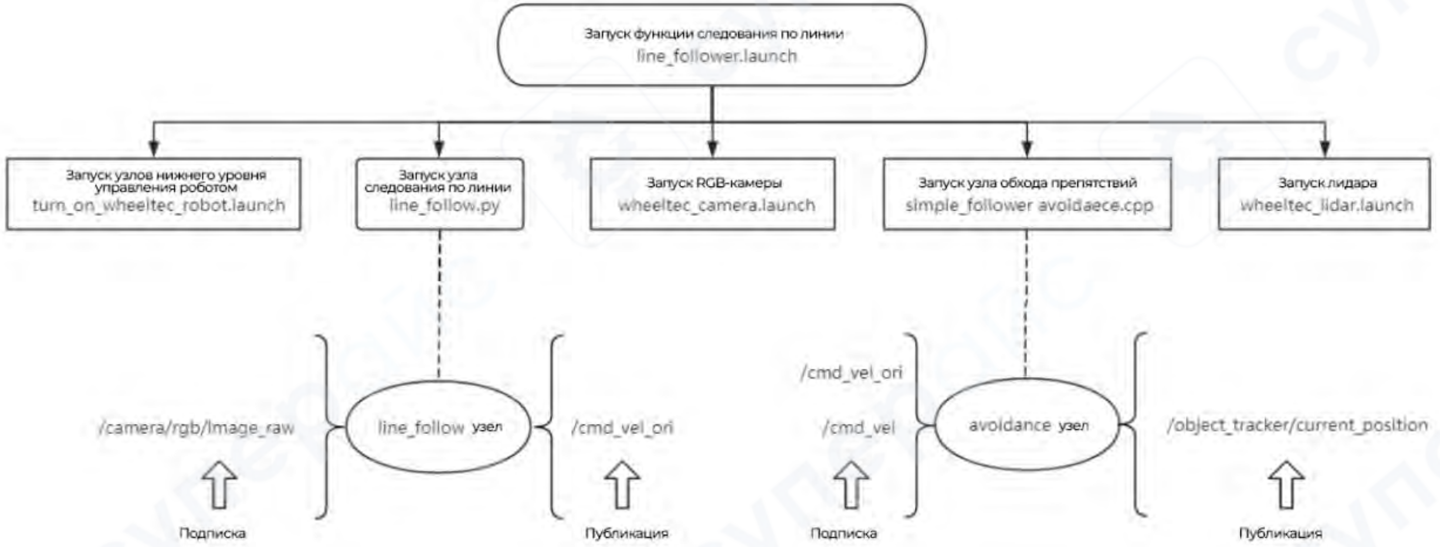
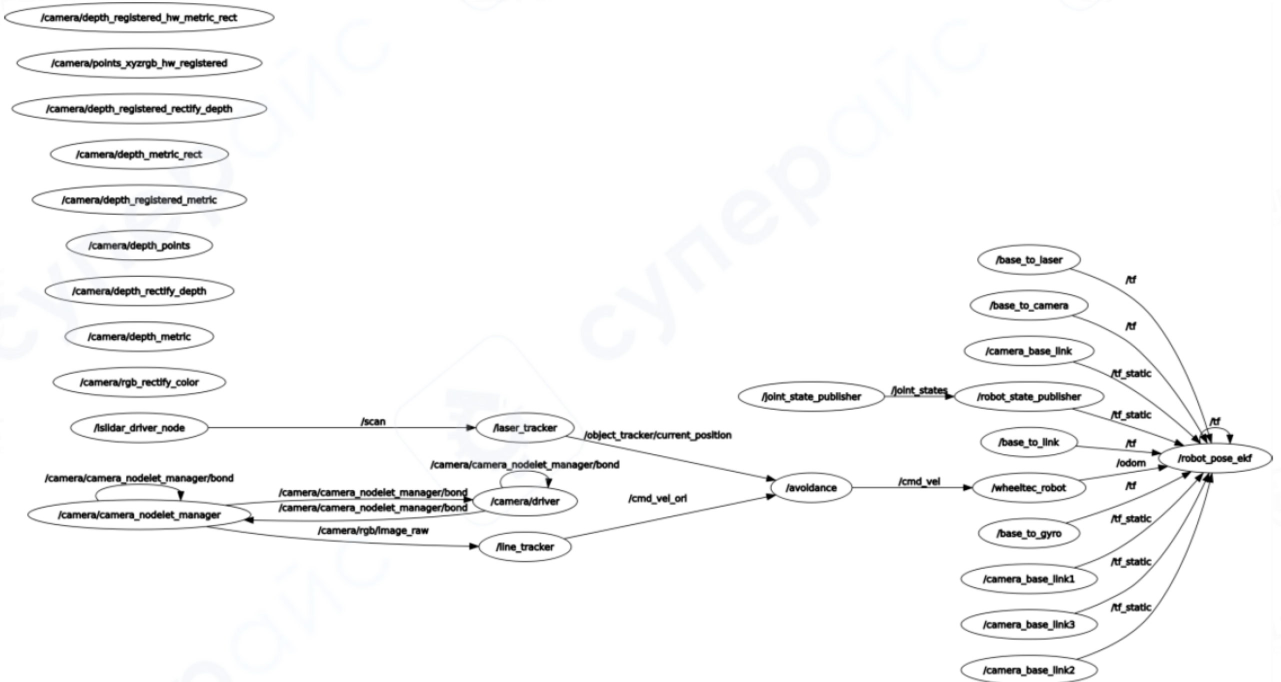


Схема запуска функции следования по линии

② Просмотр узлов функции следования по линии:

Для отображения взаимосвязи всех узлов используйте команду:

```
rqt_graph
```

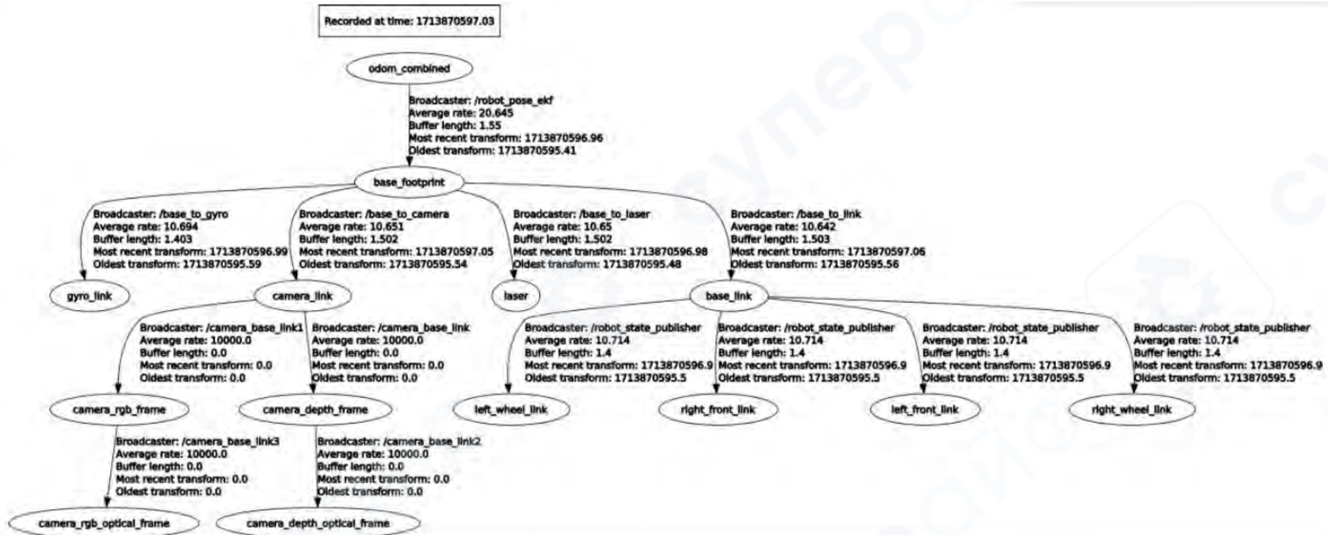


Граф отношений узлов функции следования по линии

③ **Просмотр TF-дерева функции следования по линии:**

Для просмотра пространственных отношений между узлами и координатными рамками используйте команду:

```
rosrun rqt_tf_tree rqt_tf_tree
```



TF-дерево функции следования по линии

④ **Анализ узла следования по линии**

Файл, связанный с реализацией функции следования по линии, – это **line_follower.py** из пакета **simple_follower**.

1. Установка HSV-порогов для различных цветов

Параметры HSV (оттенок, насыщенность, яркость) установлены для **пяти цветов**. Поскольку чувствительность к цветам зависит от камеры, значения нужно корректировать для других камер:

```
def nothing(s): pass
col_black = (0, 0, 0, 180, 255, 46)    # Чёрный
col_red   = (0, 100, 80, 10, 255, 255) # Красный
col_blue  = (90, 90, 90, 110, 255, 255) # Синий
col_green = (65, 70, 70, 85, 255, 255) # Зелёный
col_yellow= (26, 43, 46, 34, 255, 255) # Жёлтый
```

2. Окно выбора цвета

Создаётся всплывающее окно с **ползунком** для выбора одного из пяти цветов:

```
cv2.namedWindow('Adjust_hsv', cv2.WINDOW_NORMAL)
Switch = '0:Red\n1:Green\n2:Blue\n3:Yellow\n4:Black'
cv2.createTrackbar(Switch, 'Adjust_hsv', 0, 4, nothing)
```

3. Инициализация: подписчик и издатель

При инициализации узел подписывается на **тему RGB-камеры** и публикует команды скорости в cmd_vel_ori:

```
def __init__(self):
    self.bridge = cv_bridge.CvBridge()
    # Подписка на изображение с камеры
    self.image_sub = rospy.Subscriber("/camera/rgb/image_raw", Image, self.image_callback)
    # Публикация скорости
    self.cmd_vel_pub = rospy.Publisher("cmd_vel_ori", Twist, queue_size=1)
    self.twist = Twist()
```

4. Обработка изображения в функции image_callback

Ключевые этапы обработки:

1. **Конвертация изображения**
   - Преобразование данных из формата ROS в формат OpenCV.
2. **Изменение размера изображения**
   - Ускоряет обработку, повышая частоту кадров.
3. **Обработка изображения**
   - Применение **эрозии** и **дилатации** для удаления шумов.
4. **Определение центра цветного участка**
   - Используется **момент изображения** для нахождения геометрического центра цели.
5. **Регулировка угловой скорости с учётом смещения**
   - PID-контроль корректирует **угловую скорость**, чтобы удерживать цель по центру.

Код обработки:

```
image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
image = cv2.resize(image, (320, 240), interpolation=cv2.INTER_AREA)

# Морфологическая обработка
kernel = numpy.ones((5, 5), numpy.uint8)
hsv_erode = cv2.erode(hsv, kernel, iterations=1)
hsv_dilate = cv2.dilate(hsv_erode, kernel, iterations=1)

# Применение маски и определение цветного участка
mask = cv2.inRange(hsv_dilate, (lowerbH, lowerbS, lowerbV), (upperbH, upperbS, upperbV))
masked = cv2.bitwise_and(image, image, mask=mask)

# Вычисление центра тяжести
M = cv2.moments(mask)
if M['m00'] > 0:
    cx = int(M['m10'] / M['m00'])
    cy = int(M['m01'] / M['m00'])
    cv2.circle(image, (cx, cy), 10, (255, 0, 255), -1)

    # Расчёт ошибки и корректировка скорости
    erro = cx - w / 2 - 15
    d_erro = erro - last_erro
```

```
        self.twist.linear.x = 0.18
        self.twist.angular.z = -float(erro) * 0.005 - float(d_erro) * 0.000
        last_erro = erro
    else:
        self.twist.linear.x = 0
        self.twist.angular.z = 0

    # Публикация скорости
    self.cmd_vel_pub.publish(self.twist)
```

Объяснение работы

1. **CvBridge** конвертирует изображение из ROS в формат OpenCV.
2. Пороговые значения HSV используются для выделения области определённого цвета.
3. Применяются **морфологические операции** (эрозия и дилатация) для уменьшения шумов.
4. Центр цветного участка вычисляется с использованием моментов изображения.
5. Смещение от центра используется для корректировки **угловой скорости** робота.
6. Робот продолжает двигаться вперёд, пока **целевая линия** видна.
   - Если линия отсутствует в поле зрения, **линейная и угловая скорости** обнуляются, и робот останавливается.

⑤ Анализ узла обхода препятствий

Функция **обхода препятствий** реализована в узле **avoidance**, который обрабатывает два типа данных:

1. **Скорость**, публикуемую узлом следования по линии.
2. **Данные о позиции препятствий**, полученные от узла лидара.

На основе этих данных узел принимает решение:

- **Если препятствие обнаружено близко**, скорость обнуляется для остановки робота.
- **Если препятствий нет**, скорость, опубликованная узлом следования по линии, передаётся роботу без изменений.

Ключевые части кода avoidance.cpp

1. **Инициализация узла и создание подписчиков/издателей:**

```
ros::init(argc, argv, "avoidance");  // Инициализация ROS-узла
ros::NodeHandle node;               // Создание дескриптора узла

/*** Публикация команды управления скоростью ***/
ros::Publisher cmd_vel_Pub = node.advertise<geometry_msgs::Twist>("cmd_vel", 1);

/*** Подписка на тему скорости от узла следования по линии ***/
ros::Subscriber vel_sub = node.subscribe("cmd_vel_ori", 1, cmd_vel_ori_Callback);

/*** Подписка на тему с позицией препятствий от лидара ***/
```

```
	ros::Subscriber current_position_sub = node.subscribe("/object_tracker/current_position", 1, current_position_Callback);
```

2. **Основной цикл работы узла:**

```
double rate2 = 30;              // Частота работы узла – 30 Гц
ros::Rate loopRate2(rate2);

while (ros::ok()) {
  ros::spinOnce();           // Обработка входящих сообщений

  if (distance_judgment() && dis_angleX_judgment()) {    // Проверка на наличие препятствий
    temp_count++;          // Счётчик для фильтрации ложных срабатываний

    if (temp_count > 5) {    // Если препятствие обнаружено 5 раз подряд
      cmd_vel_Pub.publish(geometry_msgs::Twist());  // Остановка робота
      temp_count = 0;       // Сброс счётчика
    }
  } else {
    temp_count = 0;          // Сброс счётчика при отсутствии препятствий
    cmd_vel_Pub.publish(cmd_vel_msg);  // Публикация исходной скорости
  }

  ros::spinOnce();
  loopRate2.sleep();         // Ожидание следующего цикла
  }
return 0;
}
```

Объяснение работы узла:

1. **Подписка на темы:**
   - **cmd_vel_ori** – исходная скорость, опубликованная узлом следования по линии.
   - **/object_tracker/current_position** – позиция препятствий, предоставляемая лидаром.
2. **Логика обработки:**
   - Проверка расстояния и направления препятствия с помощью функций distance_judgment() и dis_angleX_judgment().
   - Если препятствие обнаружено **5 раз подряд** (для фильтрации шумов лидара), скорость обнуляется, и робот **останавливается**.
   - Если препятствий нет, скорость из узла **следования по линии** публикуется без изменений.
3. **Защита от шумов:**
   - Используется переменная temp_count для подсчёта последовательных срабатываний. Только после **5 последовательных подтверждений** робот останавливается.

Результат работы узла:

- Робот двигается вдоль линии, пока препятствие не обнаружено.
- При обнаружении препятствия робот останавливается, чтобы избежать столкновения.
- Шумы и ложные срабатывания от лидара фильтруются с помощью подсчёта.

6 Функция KCF-слежения

6.1 Краткое описание

Функция **KCF-слежения** реализует слежение за целью с использованием алгоритма **KCF** (Kernel Correlation Filter – ядровой корреляционный фильтр). Этот метод позволяет камере распознавать объект, который пользователь **выделяет в окне**, после чего робот начинает следовать за этим объектом.

6.2 Способ использования

Перед началом работы:

- Подключите компьютер к **Wi-Fi** робота.
- Выполните **SSH-подключение** к роботу.

1) Запуск функции KCF-слежения

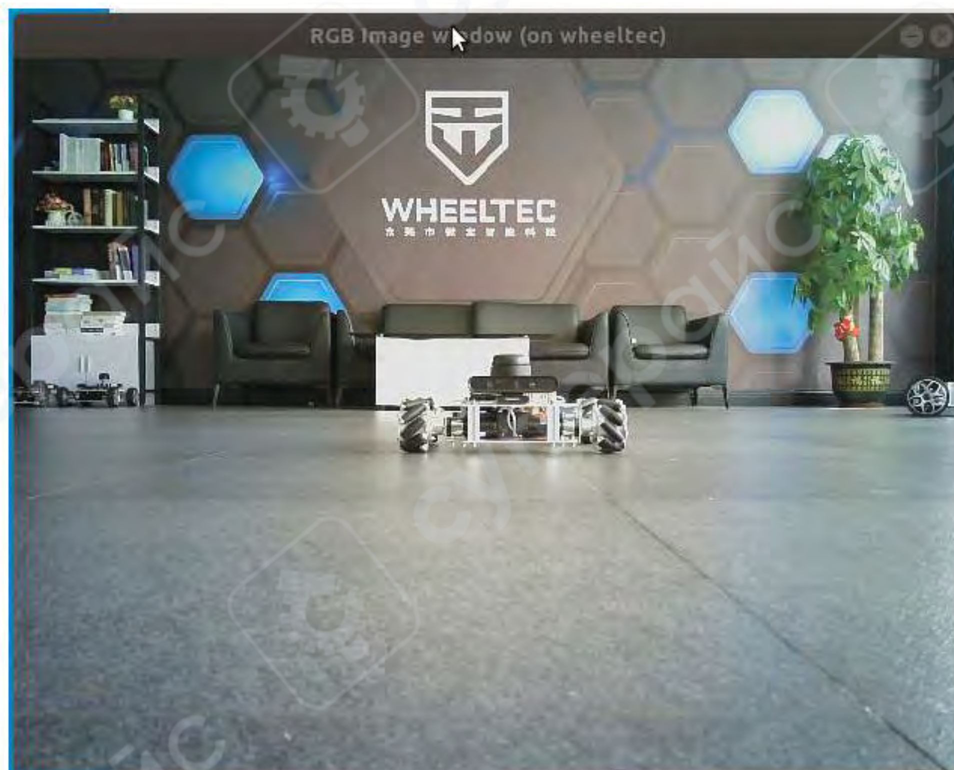
Откройте терминал и введите команду:

```
roslaunch kcf_track kcf_tracker.launch
```

2) Выбор цели для слежения

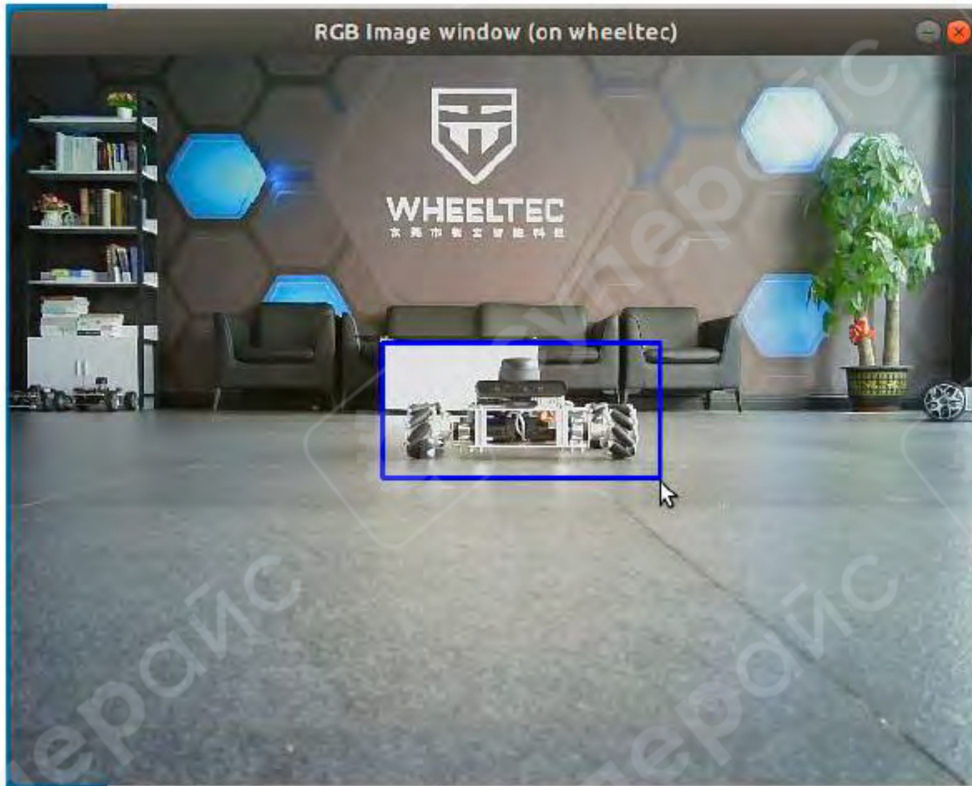
После запуска узла:

- Появится **небольшое окно** с **реальным изображением** с камеры робота.



Окно RGB-изображения показывает реальное видео с камеры

- **Зажмите левую кнопку мыши** и **выделите объект** в окне, за которым робот должен следовать.



Выделение целевого объекта для слежения

- **Синий цвет рамки** указывает на процесс выделения.
- После того как рамка будет отпущена, её цвет изменится на **жёлтый**, и робот начнёт **движение за выделенным объектом**.

Целевой объект для слежения выделен

6.3 Важные замечания

1. Проблемы при слежении робота

- При **выделении объекта для слежения** убедитесь, что целевой объект **достаточно контрастирует** с фоном, чтобы избежать ситуации, когда робот начнёт следовать за похожими объектами.
- Движение **целевого объекта не должно быть слишком быстрым**, иначе робот может потерять цель и стать неуправляемым.
- На некоторых платформах управления, таких как **Jetson Nano**, функция KCF может работать с **большой задержкой**, что ухудшает качество слежения. В этом случае рекомендуется использовать **VNC** для удалённого входа на рабочий стол ROS и запуска функции KCF **локально**.

2. Удалённый доступ через VNC

VNC позволяет управлять рабочим столом управляющей системы робота без необходимости подключения физического монитора.

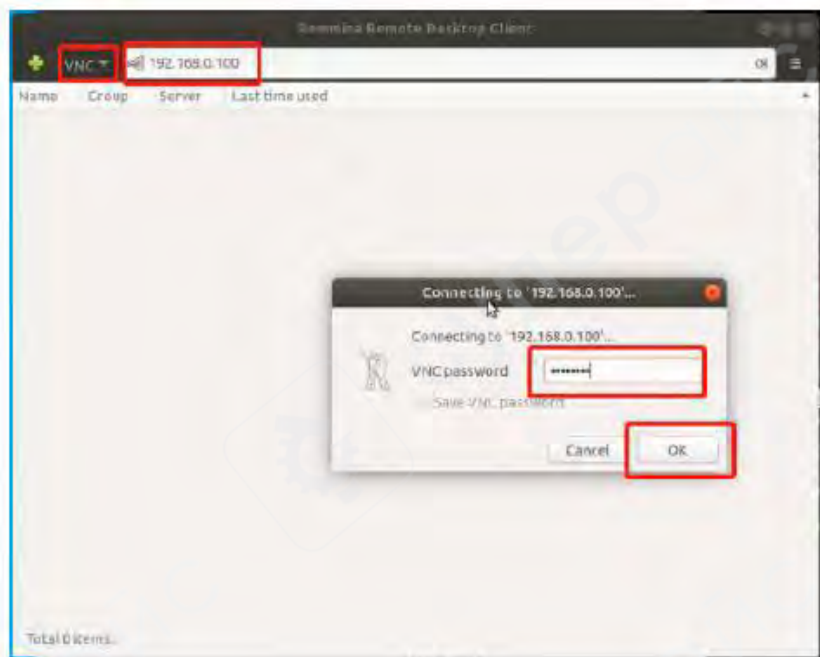
Шаги для подключения через VNC:

1. **Подключитесь к Wi-Fi робота**.
2. Откройте **Remmina** (предустановленное программное обеспечение для удалённого доступа в Ubuntu):
   - В левом нижнем углу виртуальной машины найдите и запустите **Remmina**.

Выполнение VNC-подключения с помощью Remmina

3. В Remmina:
   - Выберите **VNC** в верхнем левом углу.
   - Введите **IP-адрес робота** (по умолчанию **192.168.0.100**).
   - Дождитесь запроса пароля и введите пароль: **dongguan** (пароль по умолчанию).
   - Нажмите **OK** для входа на удалённый рабочий стол.

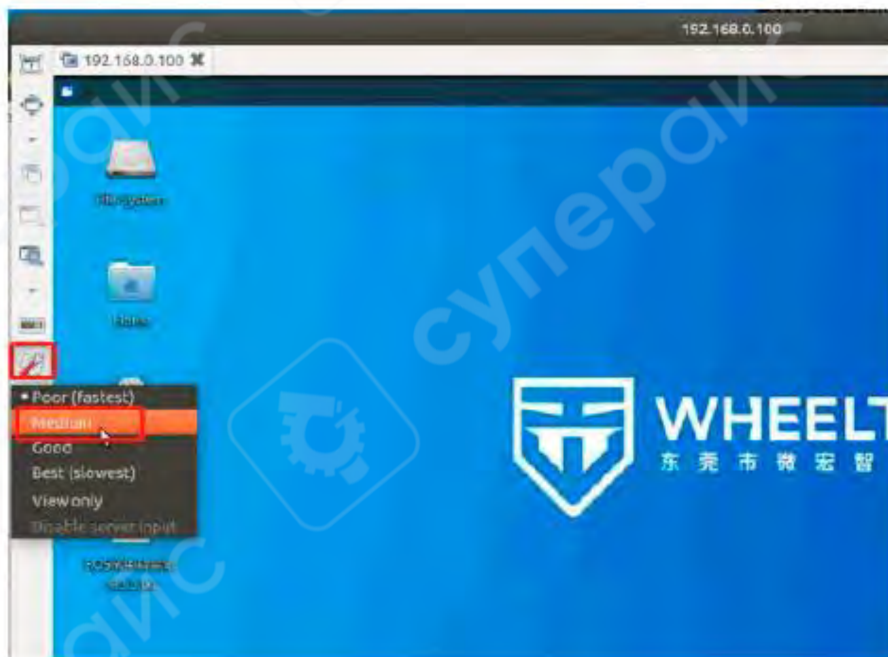
Теперь вы можете выполнять команды напрямую с удалённого рабочего стола робота, **без дополнительного SSH-входа**.



Установка удалённого соединения

3. Проблемы с качеством удалённого подключения

- Качество изображения на удалённом рабочем столе может быть **низким по умолчанию**.
- В настройках **Remmina** можно выбрать более высокое качество изображения:
    - **poor (плохое)** – стандартное значение, минимальные задержки.
    - **medium (среднее)** – более чёткое изображение, но с увеличенной задержкой.
- **Рекомендация:** не устанавливайте слишком высокое качество изображения, так как это приведёт к **увеличению задержки** и снижению частоты обновления экрана.

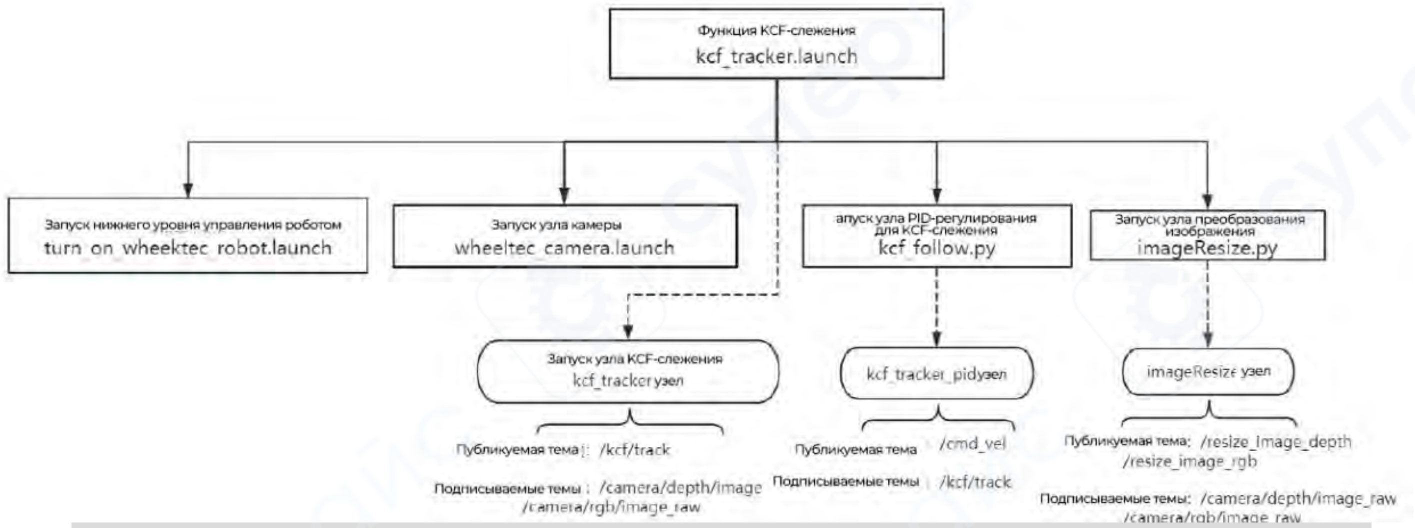


Интерфейс отображения удалённого рабочего стола

6.4 Объяснение функционала

1. Запуск функции KCF-слежения

Функция KCF-слежения запускается с помощью файла **kcf_track.launch**.

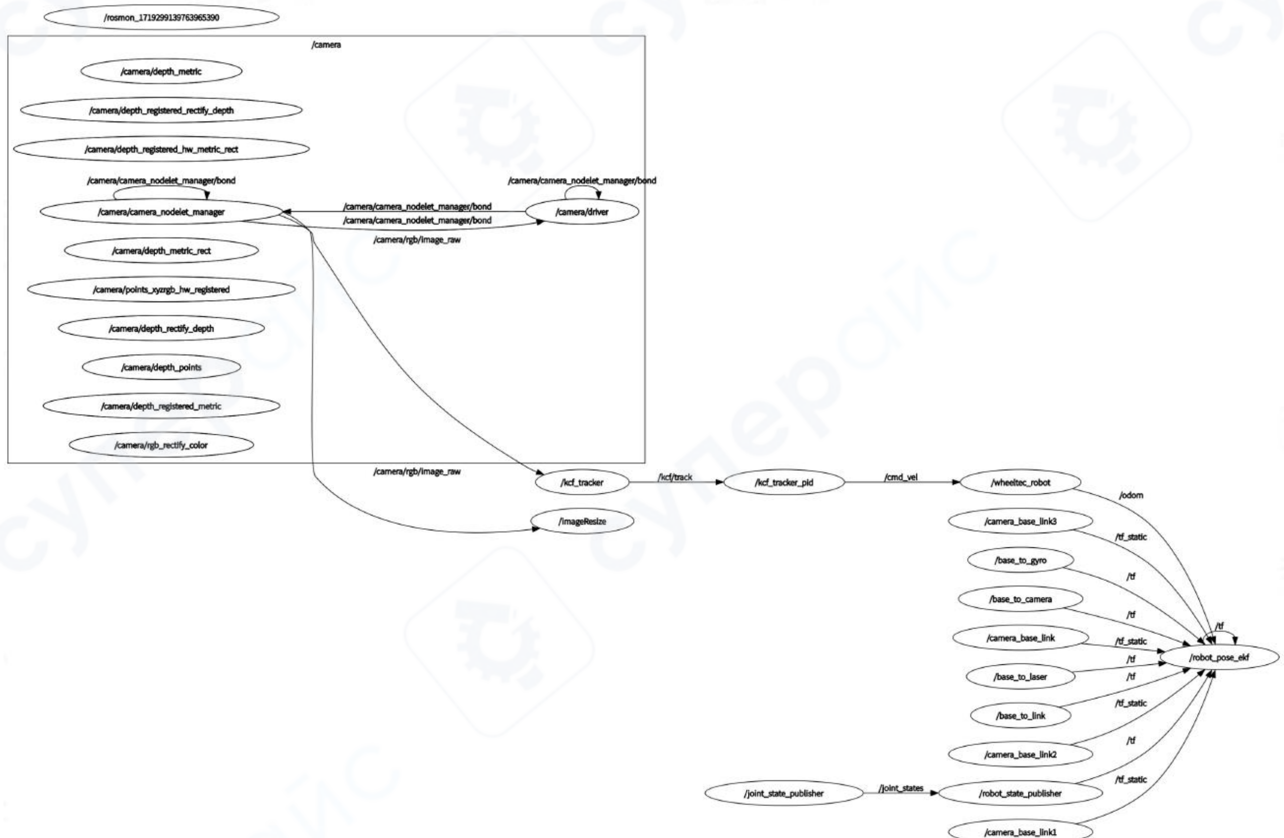


Каркас запуска функции KCF-слежения

2. Граф узлов KCF-слежения

Для отображения отношений между узлами используйте команду:

```
rqt_graph
```

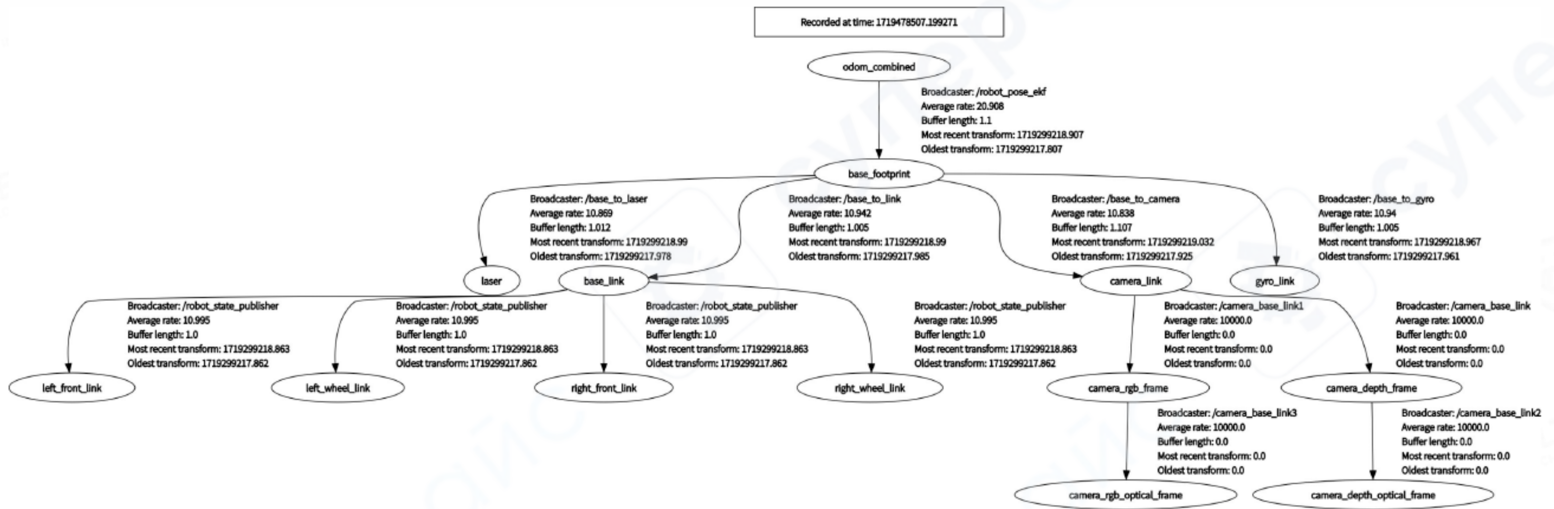


Граф узлов KCF-слежения

3. TF-дерево функции KCF-слежения

Для просмотра пространственных отношений и координатных систем используйте команду:

```
rosrun rqt_tf_tree rqt_tf_tree
```



TF-дерево функции KCF-слежения

4. Описание файла запуска функции KCF

Файл **launch** для KCF-слежения можно разделить на **четыре части**:

1. Запуск базовых узлов управления движением и камеры

```
<include file="$(find turn_on_wheeltec_robot)/launch/turn_on_wheeltec_robot.launch" />
<include file="$(find turn_on_wheeltec_robot)/launch/wheeltec_camera.launch" />
```

Эта часть запускает **узел управления движением** и **узел камеры**, которые предоставляют необходимые данные.

2. Узел преобразования изображения imageResize.py

Этот узел конвертирует изображения из **ROS-формата** в **OpenCV-формат** для последующей обработки:

```
<node name='imageResize' pkg="kcf_track" type="imageResize.py">
</node>
```

3. Запуск KCF-слежения kcf_tracker

Узел использует темы RGB и глубины для обработки изображения и отслеживания объекта:

```
<node name='kcf_tracker' pkg="kcf_track" type="kcf_node">
  <param name="rgb_topic" value="/camera/rgb/image_raw" type="string" />
  <param name="depth_topic" value="/camera/depth/image" type="string" />
</node>
```

4. Узел PID-регулирования скорости kcf_follow.py

Этот узел регулирует скорость робота на основе **PID-контроля**. Здесь задаются параметры для линейной и угловой скорости:

```
<node name='kcf_tracker_pid' pkg="kcf_track" type="kcf_follow.py">
  <param name='line_maxSpeed' value='0.3' type='double' />
  <param name='angular_maxSpeed' value='0.4' type='double' />
```

```
    <param name='dis_Kp' value='0.1' type='double' />
    <param name='dis_Kd' value='0.5' type='double' />
    <param name='dis_setPoint' value='1.2' type='double' />
    <param name='ang_Kp' value='0.002' type='double' />
    <param name='ang_Kd' value='0.001' type='double' />
    <param name='ang_setPoint' value='320' type='int' />
</node>
```

Основные узлы KCF-слежения

1. **kcf_tracker узел**
   - Инициализируется в файле **runtracker.cpp**.
   - Обрабатывает изображение в выделенной области и публикует команды скорости.
   - Включает три основных функции для обработки изображения и слежения.

2. **kcf_tracker_pid узел**
   - Определён в файле **kcf_follow.py**.
   - Отвечает за регулирование скорости робота с использованием **PID-контроля**.

7 Функция распознавания AR-меток: использование и пояснение

7.1 Краткое описание

В **ROS** предоставляется пакет **ar_track_alvar**, который реализует функцию распознавания **AR-меток**.

Функция распознавания AR-меток использует камеру для обнаружения AR-меток в её поле зрения и позволяет получать **TF-преобразования** (координаты и ориентация) и визуализировать их в виде маркеров.

7.2 Установка и описание пакета AR-меток

Команда для установки пакета:

```
sudo apt-get install ros-melodic-ar-track-alvar
```

Пакет ar_track_alvar предоставляет четыре основных функции:

1. **Генерация AR-меток** разного размера, разрешения и с различными данными/ID-кодами.
2. **Распознавание и отслеживание** позы **одной AR-метки**. При необходимости можно интегрировать данные глубины для более точной оценки позы.
3. **Распознавание и отслеживание целых групп меток** ("составных меток"). Это позволяет:
   - Обеспечить **стабильное** определение позы.
   - Повысить устойчивость к **частичному перекрытию** меток.
   - Отслеживать **многосторонние цели**.
4. **Автоматический расчёт пространственных отношений** между метками в составе группы на основе изображений с камеры. Это избавляет пользователя от необходимости вручную измерять и вводить координаты в XML-файл.

Замечание: Пакет устанавливается в **/opt/ros/melodic/share/ar_track_alvar**. Для изучения пакета можно воспользоваться **страницей ROS Wiki** (http : / / wiki . ros . org/ar_track_alvar) или исходным кодом на **GitHub**.

7.3 Способ использования

Перед началом работы:

- Подключитесь к Wi-Fi робота.
- Выполните удалённый вход на робот через **SSH**.

1) Запуск функции распознавания AR-меток

Введите команду для запуска **launch-файла**:

```
roslaunch turn_on_wheeltec_robot ar_label.launch
```

2) Генерация меток для распознавания

Введите команду для создания AR-меток:

```
rosrun ar_track_alvar createMarker -s 5 0
```

- **-s** – параметр, где:
  - Первый **число** – длина стороны метки (в сантиметрах).
  - Второе **число** – ID метки.

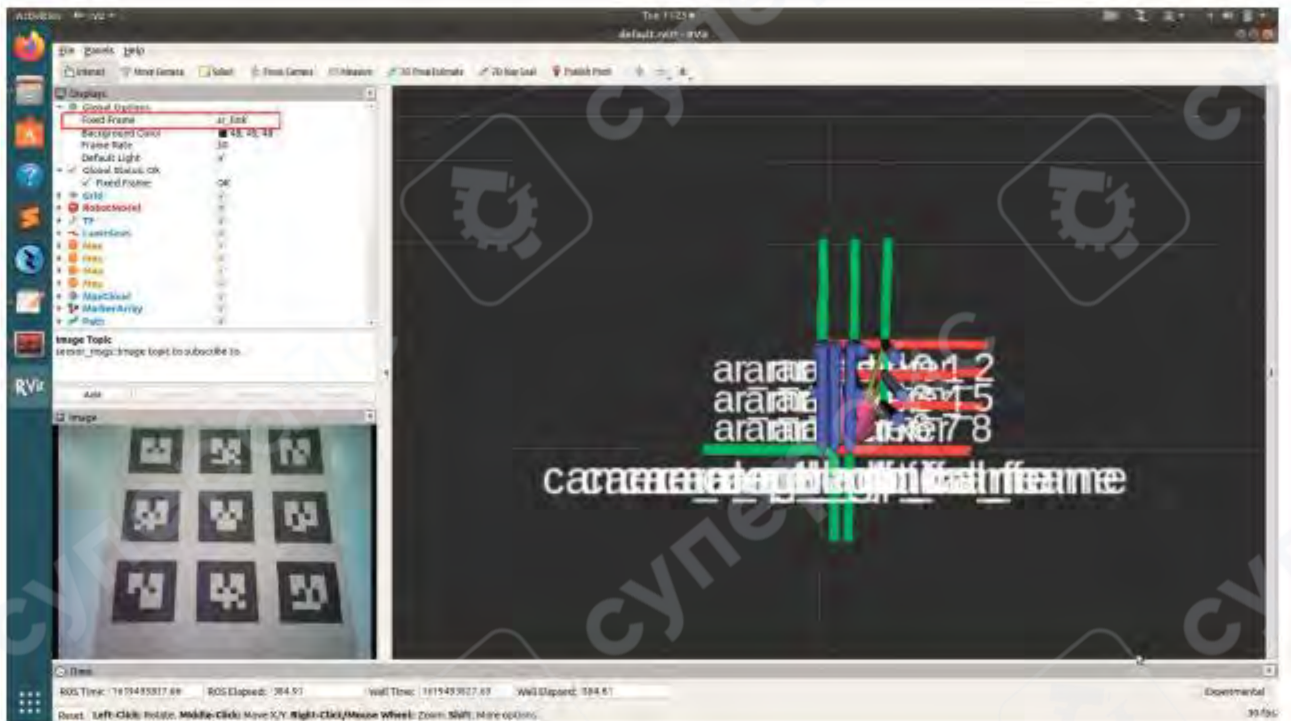
```
wheeltec@wheeltec:~/wheeltec_robot/src/turn_on_wheeltec_robot$ rosrun ar_track_alvar
 createMarker -s 5 0
```

Путь сохранения меток:

Сгенерированные метки сохраняются в текущей **рабочей директории** терминала. Например, если команда запущена в директории **turn_on_wheeltec_robot**, метки будут сохранены там же.

3) Визуализация результатов с использованием RViz

- Откройте инструмент **RViz** на верхнем компьютере.
- В **Fixed Frame** выберите базовую координатную систему **ar_link** или **camera_link**.
- Поместите AR-метку перед камерой робота.
- В RViz будет отображено **TF-преобразование** AR-метки (координаты и ориентация в пространстве).



Отображение AR-меток в RViz (с TF-преобразованиями)

7.4 Важные замечания

1. Загрузка AR-меток для распознавания

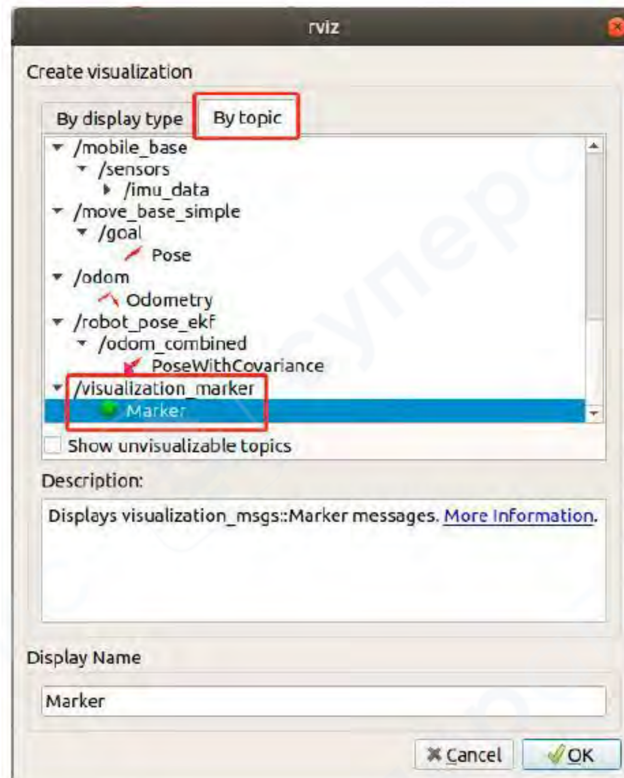
Необходимые для распознавания **AR-метки** можно скачать с **ROS Wiki**:

- Ссылка на изображения AR-меток с номерами **0–8**:

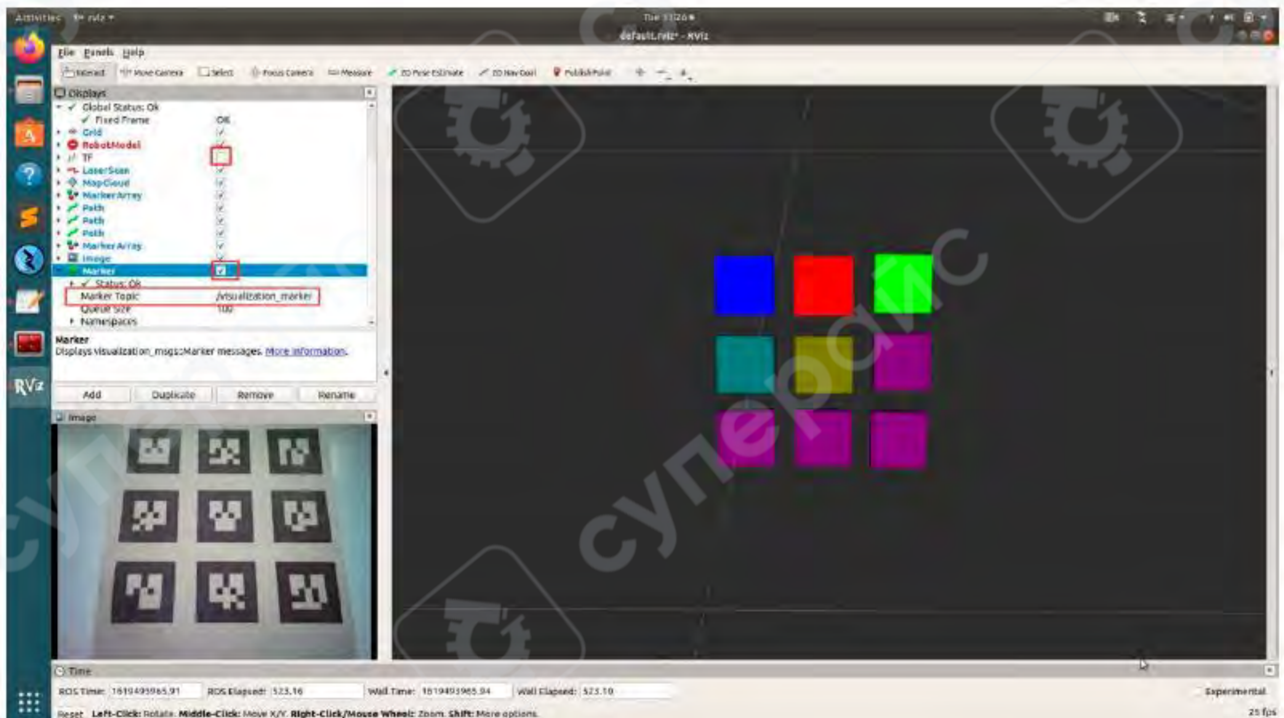
http : / / wiki . ros . org / ar_track_alvar?action=AttachFile&do=view&target=markers0to8.png

2. Альтернативный способ отображения AR-меток в RViz

1. В нижнем левом углу RViz нажмите **Add**.
2. Добавьте тему **/visualization_marker**.
3. Уберите галочку с параметра **TF**.
4. Теперь AR-метки будут отображаться в виде **небольших кубов**.



Добавить из By topic раздел /visualization_marker → Marker



Отображение AR-меток в RViz (без выбора TF)

Примечание:

При распознавании AR-меток в терминале могут появляться **ошибки**. Это связано с тем, что камера не может одновременно обрабатывать **облако точек** и распознавать AR-метки. Эти ошибки **можно игнорировать**, так как они не влияют на работу функции.

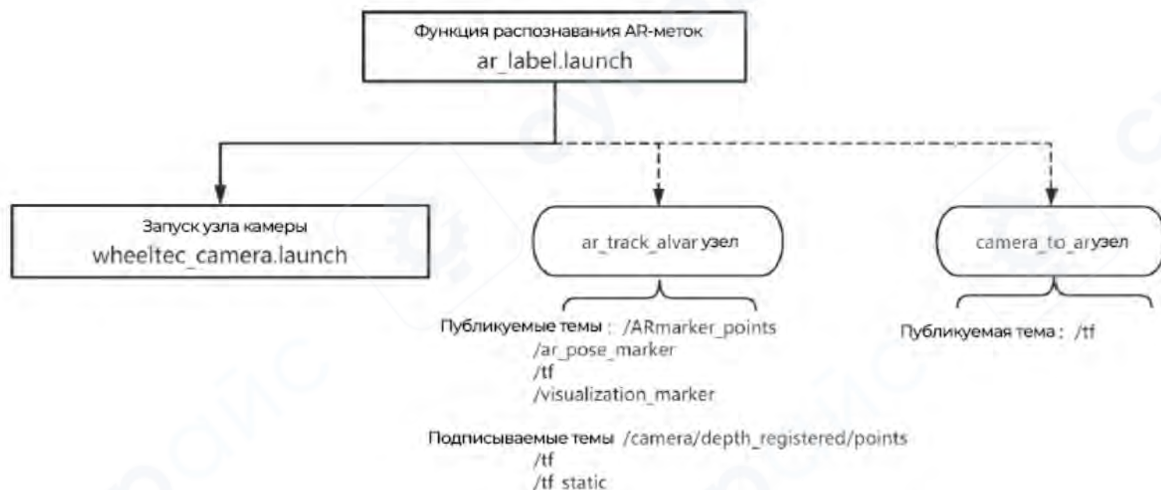
7.5 Объяснение функционала

Функция распознавания AR-меток запускается с помощью файла **ar_label.launch**.

1. Запуск функции распознавания AR-меток

Команда для запуска:

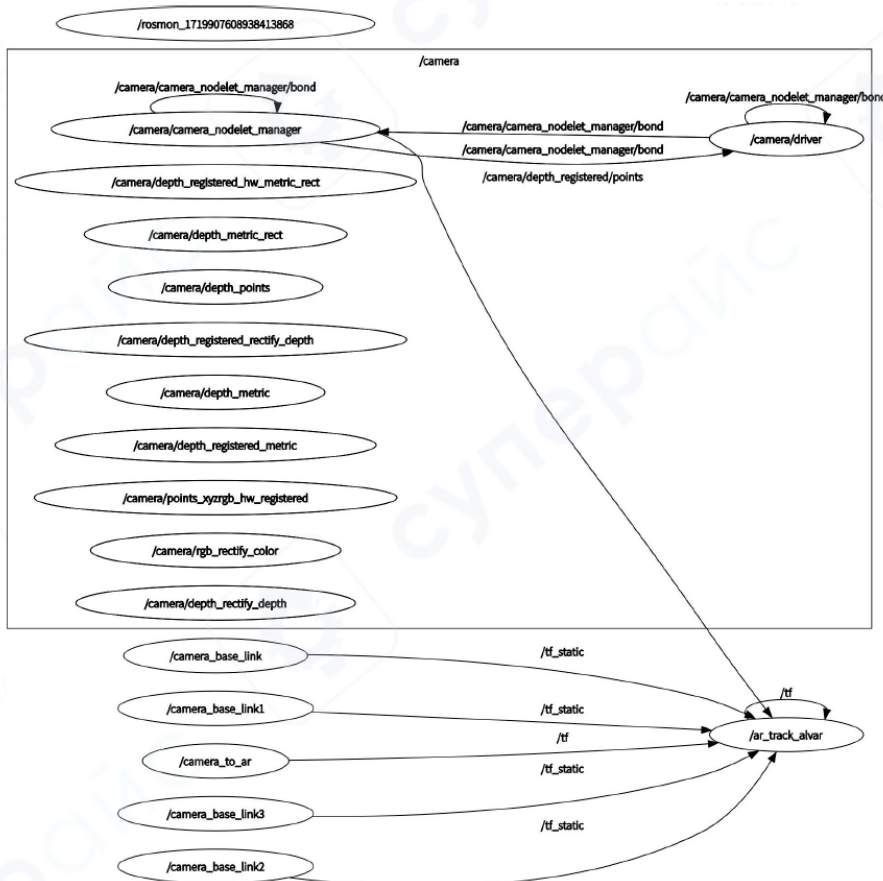
```
roslaunch turn_on_wheeltec_robot ar_label.launch
```



2. Граф узлов распознавания AR-меток

Для просмотра узлов и их отношений используйте команду:

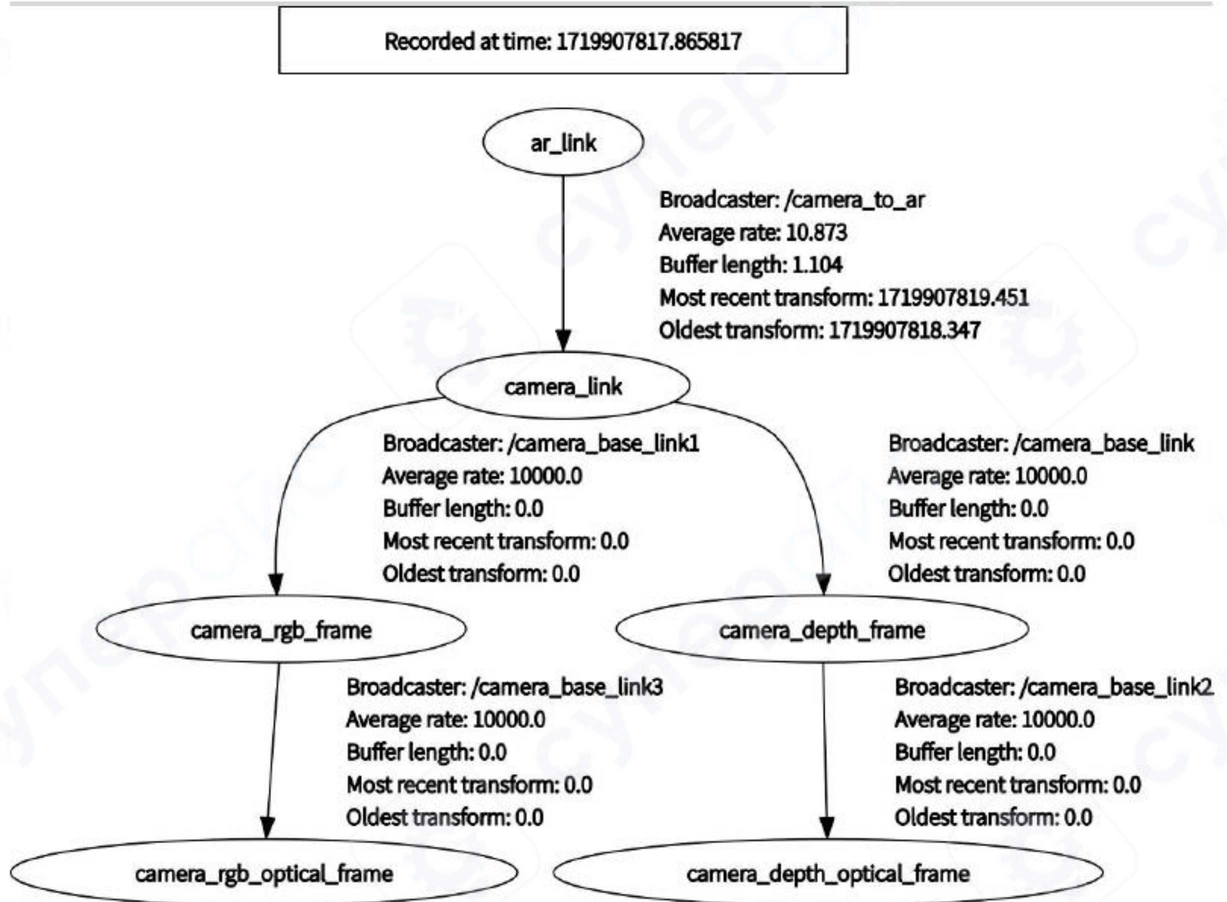
```
rqt_graph
```



3. TF-дерево распознавания AR-меток

Для отображения дерева преобразований используйте команду:

```
rosrun rqt_tf_tree rqt_tf_tree
```



4. Описание файла запуска ar_label.launch

Файл **ar_label.launch** настраивает параметры и запускает **узел распознавания AR-меток**. Основные параметры можно найти на **ROS Wiki**. Вот их значения и назначение:

```
ar_label.launch
1  <launch>
2      <!-- 开启摄像头 -->
3      <include file="$(find astra_camera)/launch/astrapro.launch" />
4      <arg name="marker_size" default="4.4" />
5      <arg name="max_new_marker_error" default="0.08" />
6      <arg name="max_track_error" default="0.2" />
7
8      <arg name="cam_image_topic" default="/camera/depth_registered/points" />
9      <arg name="cam_info_topic" default="/camera/rgb/camera_info" />
10     <arg name="output_frame" default="ar_link" />
11
12     <!-- 连接camera_link和ar_link, 连接tf树 -->
13     <node pkg="tf" type="static_transform_publisher" name="camera_to_ar" args="0 0 0 0 0 0  ar_link camera_link 100" />
14
15     <node name="ar_track_alvar" pkg="ar_track_alvar" type="individualMarkers" respawn="false" output="screen">
16         <param name="marker_size"          type="double" value="$(arg marker_size)" />
17         <param name="max_new_marker_error" type="double" value="$(arg max_new_marker_error)" />
18         <param name="max_track_error"      type="double" value="$(arg max_track_error)" />
19         <param name="output_frame"         type="string" value="$(arg output_frame)" />
20
21         <remap from="camera_image"  to="$(arg cam_image_topic)" />
22         <remap from="camera_info"   to="$(arg cam_info_topic)" />
23     </node>
24
25 </launch>
26
```

1. **marker_size (double)**
   - Размер стороны чёрной квадратной границы AR-метки (в сантиметрах).
   - По умолчанию **4.4 см** (стандарт ROS Wiki).
   - Значение можно изменить в соответствии с реальным размером метки для повышения точности.
2. **max_new_marker_error (double)**
   - Порог неопределённости для обнаружения новой метки.
3. **max_track_error (double)**
   - Порог ошибки, при котором метка считается **утраченной**.
4. **camera_image (string)**
   - Название темы с изображениями, используемой для распознавания AR-меток.
   - Требует данные с **глубинного типа точек**. В данном случае используется **тема с глубинным облаком точек**.
5. **camera_info (string)**
   - Тема с параметрами **калибровки камеры** для коррекции изображения.
   - Используется тема **RGB info**.
6. **output_frame (string)**
   - Название координатной системы, относительно которой публикуется позиция AR-метки.
   - Можно установить в **ar_link** или использовать **camera_link**.
   - Если указана несуществующая система координат, потребуется выполнить **TF-преобразование** для корректировки.

Суммарное описание файла ar_label.launch

Файл запуска **ar_label.launch** выполняет следующие задачи:

1. Настраивает параметры для распознавания AR-меток.
2. Включает камеру для получения изображений.
3. Запускает **узел ar_track_alvar** для распознавания и публикации TF-данных.

Содержимое файла **простое и прямолинейное**, и его настройка позволяет адаптировать параметры для конкретных условий использования.

8 Функция слежения за AR-меткой: использование и пояснение

8.1 Краткое описание

Функция **слежения за AR-меткой** использует распознавание AR-метки для определения её позиции и привязки координат (TF). Затем робот движется, следуя за распознанной AR-меткой.

8.2 Способ использования

Перед началом работы:

- Подключите верхний компьютер к Wi-Fi робота.
- Выполните **SSH-подключение** к роботу.

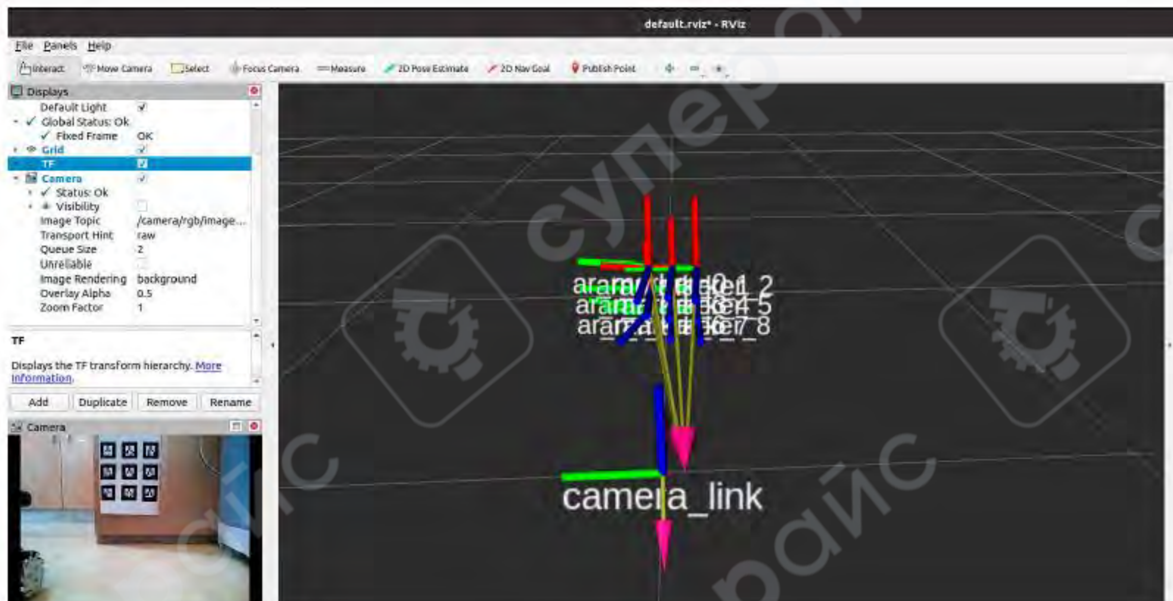
1) Запуск функции слежения за AR-меткой
Откройте терминал и введите команду:

```
roslaunch simple_follower ar_follower.launch
```

2) Работа функции:

- После запуска **launch-файла** узел AR-слежения активируется.
- Переместите AR-метку **перед камерой робота** – робот начнёт следовать за меткой.

- Откройте инструмент **RViz** и выберите в качестве **базовой системы координат**: **camera_link**.
- В RViz можно будет наблюдать **TF-координаты** камеры и AR-метки.



TF-координаты камеры и AR-меток

8.3 Важные замечания

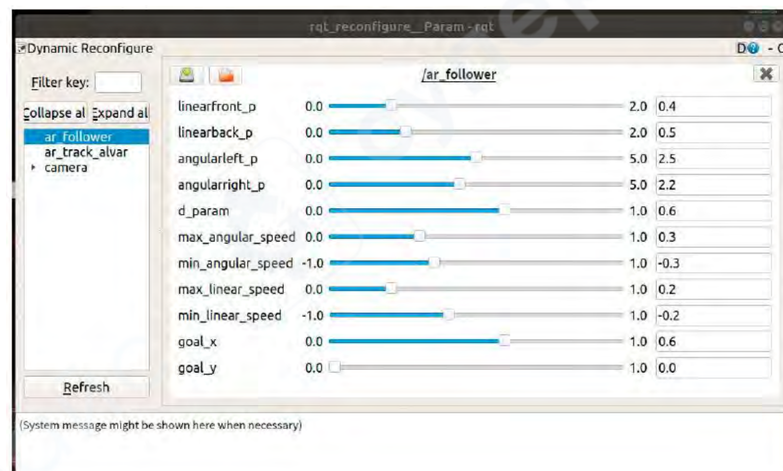
Динамическая настройка параметров:

В функции слежения за AR-меткой предусмотрена возможность **реального времени изменения параметров** с помощью **rqt_reconfigure**.

Команда для запуска:

```
rosrun rqt_reconfigure rqt_reconfigure
```

1. Выберите узел **ar_follower**.
2. Доступные параметры для настройки включают:
   - **Скорость движения робота** в различных направлениях.
   - **Максимальные и минимальные значения** линейной и угловой скорости.
   - **Расстояние и направление**, которые робот поддерживает относительно AR-метки.



Окно динамической настройки реального времени rqt

Примечание:

Функция слежения за AR-меткой также обладает возможностями **распознавания AR-меток**, аналогично функции распознавания. В **RViz** настройка и 操 作 **TF-координат** осуществляется так же. Пользователи могут выбрать подходящую функцию в зависимости от своих потребностей.

8.4 Объяснение функционала

Функция слежения за AR-меткой запускается с помощью файла **ar_follower.launch**.

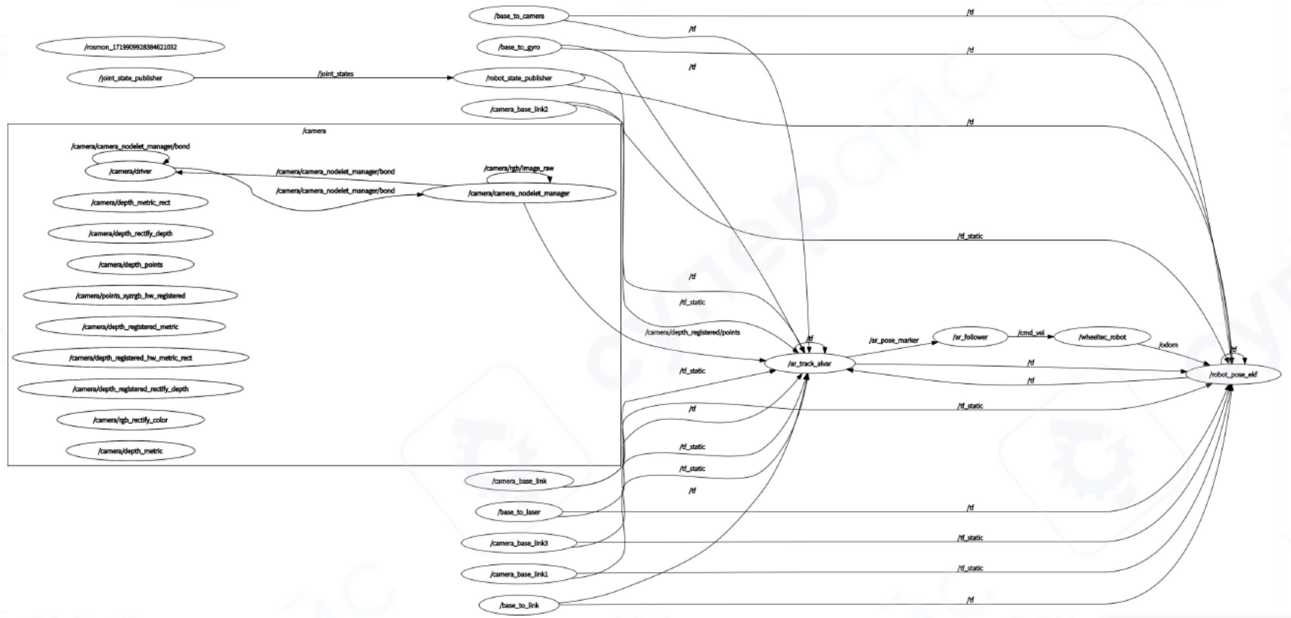
1. Запуск функции слежения за AR-меткой

```
roslaunch simple_follower ar_follower.launch
```

2. Граф узлов функции слежения за AR-меткой

Для просмотра узлов используйте команду:

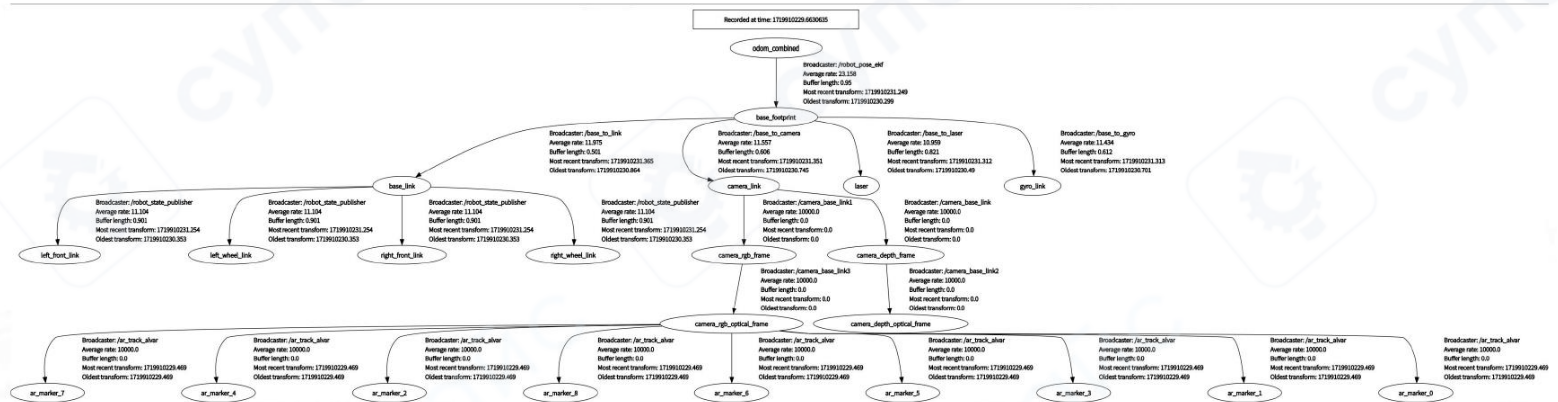
```
rqt_graph
```



3. TF-дерево функции слежения за AR-меткой

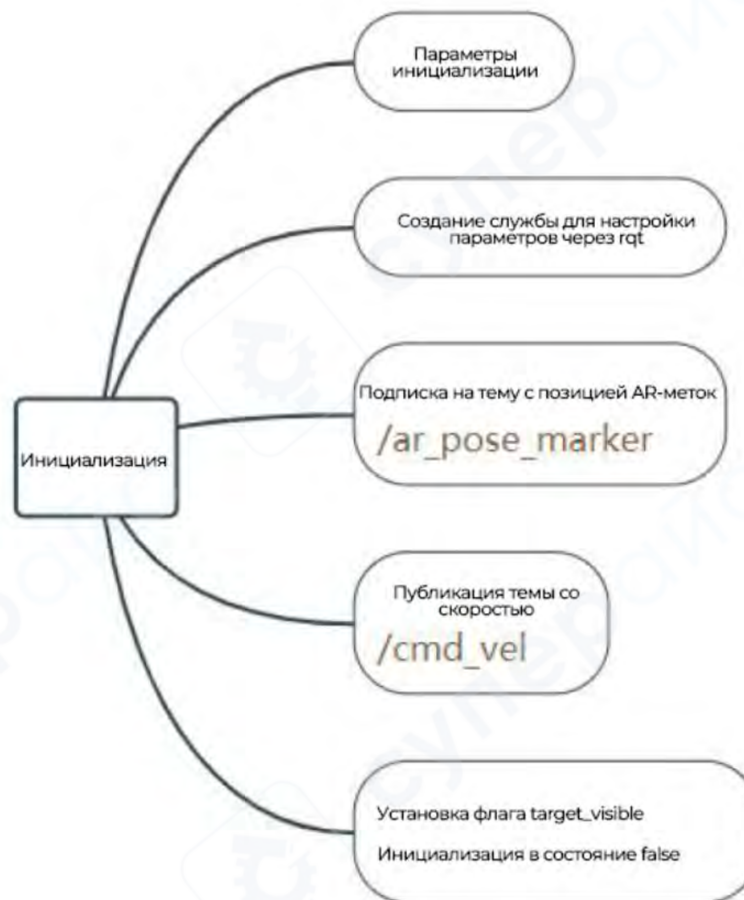
Для отображения дерева преобразований используйте:

```
rosrun rqt_tf_tree rqt_tf_tree
```



4. Описание файла запуска ar_follower.launch

Функция слежения реализована в файле **ar_follower.py**.



1. **Инициализация:**
   - Для слежения за AR-меткой подписывается на **тему /ar_pose_marker**.
   - Эта тема возвращает **позицию и смещение** AR-метки относительно заданной системы координат.
2. **Обработка распознавания метки и управления скоростью:**
   - Устанавливается **флаг** для определения состояния распознавания метки.
   - При обнаружении метки флаг активируется (True), а робот начинает движение.
   - Если метка потеряна, робот **останавливается**.

Основной код (фрагмент):

```
# Инициализация: метка не обнаружена
self.target_visible = False
self.move_cmd = Twist()
rospy.loginfo("ar_follow starting!")

# Публикация скорости, пока узел активен
while not rospy.is_shutdown():
   self.cmd_vel_pub.publish(self.move_cmd)
```

```
        rate.sleep()

    # Обновление скорости при распознавании метки
    def set_cmd_vel(self, msg):
        try:
            marker = msg.markers[0]  # Выбор первой обнаруженной метки
            if not self.target_visible:
                rospy.loginfo("The robot is Tracking Target!")
                self.target_visible = True  # Метка обнаружена

        except:
            if self.target_visible:
                rospy.loginfo("Robot Lost Target!")
                self.target_visible = False
            self.move_cmd.linear.x = 0  # Остановка робота
            self.move_cmd.angular.z = 0
            return

        # Корректировка смещения по осям
        offset_y = 0.06  # Смещение центра робота относительно центра метки
        target_offset_y = marker.pose.pose.position.y + offset_y  # Смещение по Y
        target_offset_x = marker.pose.pose.position.x  # Смещение по X
```

Описание работы кода:

1. **Инициализация:**
   - При запуске функция устанавливает флаг self.target_visible в **False** (метка не обнаружена).
2. **Публикация команд скорости:**
   - Пока узел активен, **публикуются команды скорости** с текущими значениями self.move_cmd.
3. **Обнаружение метки:**
   - При распознавании первой метки через **/ar_pose_marker** флаг self.target_visible устанавливается в **True**.
   - Позиции X и Y метки используются для вычисления и корректировки движения робота.
4. **Потеря метки:**
   - Если метка теряется, флаг возвращается в **False**, и робот **останавливается** (линейная и угловая скорости обнуляются).