Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа компьютерных технологий и информационных систем

О Т Ч Ё Т П О Л А Б О Р А Т О Р Н О Й Р А Б О Т Е № 4

Дисциплина: Программное обеспечение встраиваемых систем

Тема: Разработка узла автоматического управления роботом

Выполнил
студент гр. 5140901/21501 <*подпись*> А.М. Кобыжев

Преподаватель <*подпись*> Г.С. Васильянов

«___» _____ 2023 г.

Санкт-Петербург

2023

СОДЕРЖАНИЕ

1. ЦЕЛИ РАБОТЫ

Разработка узла, управляющего движением робота в автоматическом режиме при помощи лидара.

2. ЗАДАНИЕ

- Разработать узел, управляющий движением робота в автоматическом режиме.
  - Расположение ноды для автоматического режима – *myrobot/myrobot_teleop/nodes/lidar_auto_teleop*;
  - запуск – *teleop_auto.launch*
- Робот должен избегать препятствий используя лидар
- Используемая карта – testwalls; запуск – gazebo_testwalls.launch

3. ХОД РАБОТЫ

Репозиторий с исходным кодом для данной лабораторной работы можно посмотреть по ссылке:

https://github.com/alexnevskiy/urdf_labs/tree/lab4

3.1. Описание алгоритма автоматического управления

Для разработки узла, управляющего движением робота в автоматическом режиме при помощи лидара, предлагается следующий алгоритм работы. Видимость лидара разбивается на 4 равных сектора – передний, два боковых и задний, как показано на рис. 3.1. Каждый сектор имеет область видимости в 90 градусов и отвечает за определённую сторону робота. Для определения движения робота используется передний сектор, который в свою очередь делится на три равных подсектора. Крайние подсекторы отвечают за поворот робота. Если расстояние до препятствия ниже определённого порогового значения, то робот поворачивает в противоположную сторону. То есть если правый подсектор зафиксировал расстояние меньше порога, то робот будет поворачивать влево до тех пор, пока расстояние до препятствия не станет выше порога. Центральный подсектор отвечает за движение вперёд или назад. Если расстояние до препятствия выше

определённого порога, то робот продолжает движение вперёд, иначе останавливается. При остановке по достижении определённого количества времени робот сдаёт назад либо в случайном направлении, если ни один из боковых подсекторов не зафиксировал препятствие, либо в противоположном направлении от засёкшего препятствие подсектора.

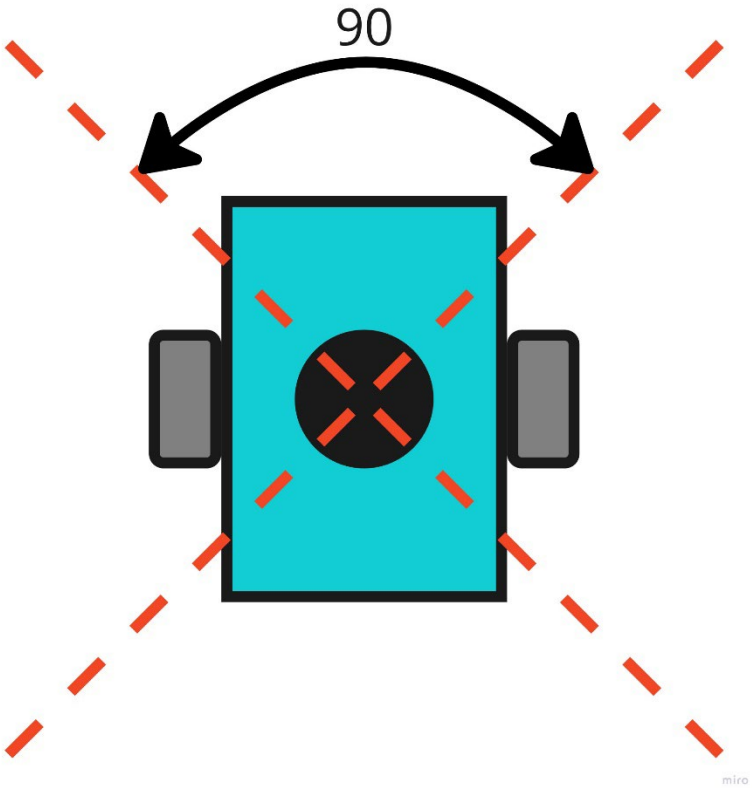

Рис. 3.1. Схема робота с лидаром

Помимо этого робот может поворачивать на различного рода перекрёстках. Для определения возможности поворота на 90 градусов необходимо, чтобы сектор в 15 градусов из расстояний с боковых секторов лидара превышал заданный порог, как показано на рис. 3.2. То есть, если в заданном секторе все значения расстояний превышают заданный порог, то робот может повернуть в эту сторону (на 90 градусов). Выбор стороны, куда направится робот (прямо, налево или направо) определяется следующим образом. Если хотя бы один из боковых секторов определил, что возможен поворот, то с вероятностью 50% определяется, будет робот поворачивать или продолжит движение прямо. Если робот всё же будет поворачивать, то

определяется направление его будущего движения на основе того, какие из боковых секторов имеют возможность для манёвра. Если оба сектора имеют такую возможность, то случайным образом определяется сторона поворота, иначе выбирается та сторона, откуда поступила готовность. После определения стороны поворота робот на определённое время перестаёт принимать решения о поворотах на перекрёстках, чтобы не крутиться на одном месте при достижении любого из перекрёстков.

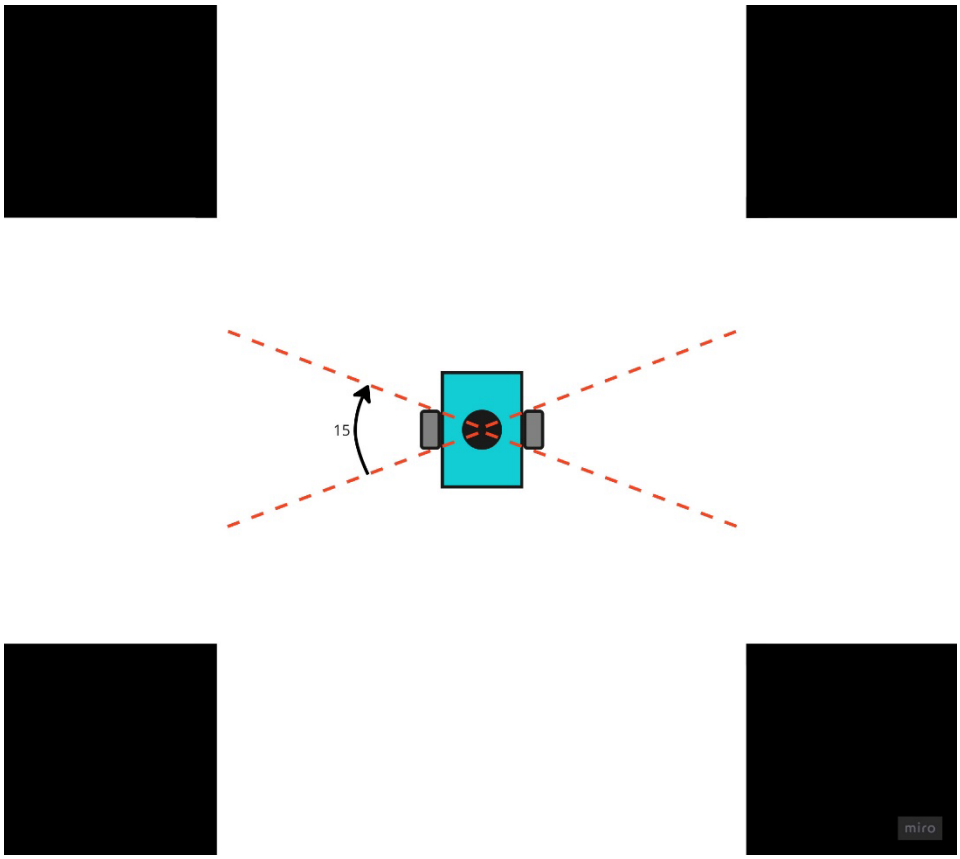

Рис. 3.2. Схема поворота робота на перекрёстке

3.2. Описание узла управления роботом в автоматическом режиме (лидар)

Для описания узла управления роботом в автоматическом режиме при помощи лидара модифицирован описанный скрипт в прошлой лабораторной работе *ir_auto_teleop*. Описание узла произведено в файле *lidar_auto_teleop*, код которого представлен в листинг 3.1. Также изменено количество лучей в *macro.gazebo.xacro*, исходящих из лидара, с 400 до 360, чтобы было удобно производить вычисления в алгоритме.

Представленный скрипт имеет схожую структуру с описанным ранее. При инициализации класса *LidarAutoTeleop* происходит инициализация ноды *lidar_auto_teleop*, регистрация паблишера для отправки сообщений топику, отвечающему за управление роботом, подписчика на получение данных о расстояниях с лидара, а также задание начальных значений полей для линейной и угловой скоростей, всех расстояний с лидара, а также расстояний для передней, левой, правой и задней частей робота.

Для запуска управления роботом вызывается функция *run()*, в которой сначала задаётся частота опроса (10 Гц), счётчики для подсчёта тактов срабатывания угловых и центрального датчика, счётчик тактов поворота робота при переключении на другую дорогу, счётчик игнорирования готовности поворота робота, а также флаги, отвечающие за готовность поворота робота на другую дорогу, за поворот робота в левую или правую сторону, за игнорирование готовности поворота, а также за движение робота в обратном направлении. Далее в бесконечном цикле происходит обновление состояния робота – функция *update_state()*, после чего выводится различного рода информация с лидара и о решениях робота, а также описывается логика работы управления роботом с помощью лидара.

Сначала происходит получение данных о расстояниях с лидара по четырём сторонам робота. Расстояние под самым первым индексом в полученном массиве рассчитано относительно луча, исходящего из лидара перпендикулярно задней части робота, то есть по факту нормаль к задней вертикальной поверхности тела робота. Так как всего исходящих лучей из лидара 360, то 0, 89, 179 и 269 расстояния в массиве получены от лучей, являющихся нормалями к четырём сторонам робота. Поделим весь массив на 4 равные части, каждая из которых будет отвечать за определённую часть робота. Данные о расстояниях хранятся в массиве против часовой стрелки относительно передней части робота. За переднюю часть отвечает подмассив, начиная с индекса 135 по 225 не включительно. По аналогии слева с 225 по 315, справа с 45 по 135, с -45 по 45. Из-за того, что значения хранятся против

часовой стрелки, то необходимо их отзеркалить, чтобы начальные значения отвечали за левую часть промежутка расстояний стороны робота, а конечные за правую.

Далее идёт блок кода, отвечающий за определение стороны, куда повернёт робот. В начале производится проверка того, что центральный пучок расстояний для левой или правой стороны, состоящий из 15 элементов, имеет расстояния больше заданного порога (7 метров). Если данное условие соблюдено, то флаг, отвечающий за готовность поворота в определённую сторону, устанавливается в истину, иначе в ложь. Далее производится проверка на полученные значения флагов. Если хотя бы один из двух является истиной, то флаг игнорирования готовности поворота ставится в истину и случайным образом генерируется логическое значение для поворота. Если оно равно истине или робот не двигается вперёд (обработка случая, когда робот упёрся в стену), то определяется, в какую сторону повернёт робот. Если оба флага готовности поворота равны истине, то выбирается случайное направление, иначе то, у которого флаг равен истине.

Затем идёт блок кода, отвечающий за выбор направления робота, на основе значений флагов, рассмотренных ранее. Если хотя бы один из флагов готовности поворота равен истине и при этом робот не движется назад (обработка случая, когда при движении назад робот может резко сменить курс из-за срабатывания алгоритма поворота на перекрёстке), то происходит обработка стороны поворота робота. Сначала повышается значение счётчика поворота (поворот робота на перекрёстке на 90 градусов занимает 10 тактов), после чего делается проверка на достижение его предела. Если предел достигнут, то значения флагов готовности поворота сбрасывается в ложь. Далее идёт проверка на флаги готовности поворота, и в зависимости от значения флагов выставляются нужные значения переменных для поворота в определённую сторону.

После этого идёт блок кода, отвечающий за движение робота во все стороны, основываясь на расстояниях из передней части. Сначала

производится проверка на флаг игнорирования поворота и счётчик тактов игнорирования. Если значение флага равно истине, а значение счётчика игнорирования готовности поворота меньше 50 (50 тактов = 5 секунд при 10 Гц), то значение счётчика увеличивается, иначе обнуляется и флаг игнорирования готовности поворота выставляется в ложь.

Далее по аналогии с алгоритмом, основанном на ик-сенсорах, производится определение направления движения робота. Однако, чтобы использовать возможности лидара, берётся не центральное значение одного из трёх секторов расстояний в передней части робота (по аналогии с тремя ик-сенсорами), а минимальное, что позволяет роботу раньше среагировать на приближающееся препятствие. То есть из 30 значений (90 расстояний делим на 3 равные части) выбирается минимальное расстояние до препятствия. Теперь перейдём к описанию самого алгоритма выбора направления движения робота. Сначала происходит проверка на то, что одно из минимальных боковых значений имеет расстояние до препятствия меньше заданного расстояния (1,25 метра). Если расстояние меньше, то увеличивается счётчик подсчёта тактов для угловых секторов и определяется угловая скорость в зависимости от того, какой из секторов имеет наименьшее расстояние до препятствия, иначе угловая скорость и счётчик обнуляются. Далее происходит проверка расстояния для центрального сектора. Если расстояние меньше, то линейная скорость устанавливается в 0 и повышается счётчик подсчёта тактов для центрального сектора, иначе линейная скорость устанавливается в 1 и счётчик обнуляется. В конце происходит проверка на достижение одного из счётчиков заданного предела (50 тактов = 5 секунд при 10 Гц). При положительной проверке флаг обратного движения ставится в истину, иначе в ложь. Если флаг положительный, то линейная скорость устанавливается в -1 (задний ход) и в зависимости от конкретного значения угловой скорости в данный момент высчитывается будущее направление робота. Если угловая скорость не равна нулю, то устанавливается обратное значение, чтобы робот при движении назад поворачивал в сторону сектора, имеющего меньшее

расстояние до препятствия, иначе выбирается случайная угловая скорость между 1 и -1 для случаев, когда порог до препятствия зафиксирован только в центральном секторе, например при столкновении с препятствием острой формы.

Листинг 3.1. Описание узла управления роботом в автоматическом режиме

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import random
import numpy as np

MINICAR_MAX_LIN_VEL = 1
MINICAR_MAX_ANG_VEL = 2
MAX_LIN_VEL_MULTIPLIER = 1
MIN_LIN_VEL_MULTIPLIER = -1
DEFAULT_LIN_VEL_MULTIPLIER = 0
VEL_STEP = 0.05

LIDAR_CENTER_TARGET_RANGE = 0.75
LIDAR_SIDE_TARGET_RANGE = 1.25

READY_TO_TURN_RANGE = 10
READY_TO_TURN_COUNT = 15
COUNT_TO_TURN = 10

CLOSE_DIST_TARGET = 50
IGNORE_READY_TARGET = 50

LEFT_RANGE_LEFT_BORDER = 315
LEFT_RANGE_RIGHT_BORDER = 225
FRONT_RANGE_LEFT_BORDER = 225
FRONT_RANGE_RIGHT_BORDER = 135
RIGHT_RANGE_LEFT_BORDER = 135
RIGHT_RANGE_RIGHT_BORDER = 45
BACK_RANGE_LEFT_BORDER = 45
BACK_RANGE_RIGHT_BORDER = -45

LIDAR_TOPIC = '/myrobot/rplidar/scan'

class LidarAutoTeleop:
    def __init__(self):
        rospy.init_node('lidar_auto_teleop')

        cmd_vel_topic = rospy.get_param('/lidar_auto_teleop/cmd_vel', '/cmd_vel')
        rospy.loginfo('Cmd Vel topic: ' + cmd_vel_topic)

        self.pub = rospy.Publisher(cmd_vel_topic, Twist, queue_size=10)

        self.ranges = np.zeros(360)
        self.front_ranges = np.zeros(90)
        self.left_ranges = np.zeros(90)
        self.right_ranges = np.zeros(90)
        self.back_ranges = np.zeros(90)

        self.sub = rospy.Subscriber(LIDAR_TOPIC, LaserScan, callback=self.callback)

        self.state_vel = 0
        self.state_ang = 0
        self.target_linear_vel = 0.0
        self.target_angular_vel = 0.0
```

```
        self.forward = False
        self.back = False
        self.left = False
        self.right = False

    def run(self):
        rate = rospy.Rate(10)
        close_side_dist_counter = 0
        close_center_dist_counter = 0
        turn_counter = 0
        ignore_ready_counter = 0
        left_ready_flag = False
        right_ready_flag = False
        turn_left_flag = False
        turn_right_flag = False
        ignore_ready_flag = False
        reverse_flag = False
        try:
            while not rospy.is_shutdown():
                self.update_state()
                rate.sleep()
                # logs
                print('================================================')
                print('front %06.3f | left %06.3f | right %06.3f | back %06.3f' %
(self.ranges[179], self.ranges[269], self.ranges[89], self.ranges[0]))
                print('ready_left ' + str(left_ready_flag) + ' | ready_right ' +
str(right_ready_flag))
                print('ignore ' + str(ignore_ready_flag) + ' | turn_left ' +
str(turn_left_flag) + ' | turn_right ' + str(turn_right_flag))
                print('center_counter ' + str(close_center_dist_counter) + ' | side_counter
' + str(close_side_dist_counter))
                print('================================================')
                self.front_ranges =
np.flipud(self.ranges[FRONT_RANGE_RIGHT_BORDER:FRONT_RANGE_LEFT_BORDER])
                self.left_ranges =
np.flipud(self.ranges[LEFT_RANGE_RIGHT_BORDER:LEFT_RANGE_LEFT_BORDER])
                self.right_ranges =
np.flipud(self.ranges[RIGHT_RANGE_RIGHT_BORDER:RIGHT_RANGE_LEFT_BORDER])
                self.back_ranges =
np.flipud(np.concatenate((self.ranges[BACK_RANGE_RIGHT_BORDER:],
self.ranges[0:BACK_RANGE_LEFT_BORDER])))

                if not ignore_ready_flag:
                    if len(np.argwhere(self.left_ranges[37:52] > READY_TO_TURN_RANGE)) >=
READY_TO_TURN_COUNT:
                        left_ready_flag = True
                    else:
                        left_ready_flag = False
                    if len(np.argwhere(self.right_ranges[37:52] > READY_TO_TURN_RANGE)) >=
READY_TO_TURN_COUNT:
                        right_ready_flag = True
                    else:
                        right_ready_flag = False

                    if left_ready_flag or right_ready_flag:
                        ignore_ready_flag = True
                        turn = random.choice([True, False])
                        if turn or not self.forward:
                            if left_ready_flag and right_ready_flag:
                                turn_left_flag = random.choice([True, False])
                                turn_right_flag = not turn_left_flag
                            elif left_ready_flag:
                                turn_left_flag = True
                            else:
                                turn_right_flag = True

                if (turn_left_flag or turn_right_flag) and not reverse_flag:
                    turn_counter += 1
                    if turn_counter >= COUNT_TO_TURN:
```

```
                        turn_left_flag = False
                        turn_right_flag = False
                        turn_counter = 0
                    if turn_left_flag:
                        self.forward = False
                        self.left = True
                        self.right = False
                    if turn_right_flag:
                        self.forward = False
                        self.left = False
                        self.right = True
                else:
                    if ignore_ready_flag and ignore_ready_counter < IGNORE_READY_TARGET:
                        ignore_ready_counter += 1
                    else:
                        ignore_ready_counter = 0
                        ignore_ready_flag = False

                    min_range_left = np.min(self.front_ranges[0:30])
                    min_range_center = np.min(self.front_ranges[30:60])
                    min_range_right = np.min(self.front_ranges[60:90])
                    if min_range_left < LIDAR_SIDE_TARGET_RANGE or min_range_right <
LIDAR_SIDE_TARGET_RANGE:
                        close_side_dist_counter += 1
                        if min_range_left < min_range_right:
                            self.right = True
                        else:
                            self.left = True
                    else:
                        self.left = False
                        self.right = False
                        close_side_dist_counter = 0

                    if min_range_center < LIDAR_CENTER_TARGET_RANGE:
                        close_center_dist_counter += 1
                        self.forward = False
                    else:
                        self.forward = True
                        close_center_dist_counter = 0

                    if close_side_dist_counter >= CLOSE_DIST_TARGET or
close_center_dist_counter >= CLOSE_DIST_TARGET:
                        reverse_flag = True
                    else:
                        reverse_flag = False

                    if reverse_flag:
                        self.forward = False
                        self.back = True
                        if not self.left and not self.right:
                            self.left = random.choice([True, False])
                            self.right = not self.left
                        else:
                            self.left = not self.left
                            self.right = not self.right
                    else:
                        self.back = False
            except rospy.ROSInterruptException:
                pass

    def update_state(self):
        self.update_speed()
        self.target_linear_vel = MINICAR_MAX_LIN_VEL * self.state_vel
        self.target_angular_vel = MINICAR_MAX_ANG_VEL * self.state_ang
        twist = Twist()
        twist.linear.x = self.target_linear_vel; twist.linear.y = 0.0; twist.linear.z = 0.0
        twist.angular.x = 0.0; twist.angular.y = 0.0; twist.angular.z =
self.target_angular_vel
        self.pub.publish(twist)
```

```
    def update_speed(self):
        if self.forward and self.back:
            self.state_vel = 0
        elif self.forward:
            self.state_vel = max(self.state_vel, DEFAULT_LIN_VEL_MULTIPLIER)
            self.state_vel += VEL_STEP
            self.state_vel = min(self.state_vel, MAX_LIN_VEL_MULTIPLIER)
        elif self.back:
            self.state_vel = min(self.state_vel, DEFAULT_LIN_VEL_MULTIPLIER)
            self.state_vel -= VEL_STEP
            self.state_vel = max(self.state_vel, MIN_LIN_VEL_MULTIPLIER)
        else:
            self.state_vel = 0

        if self.left and self.right:
            self.state_ang = 0
        elif self.left:
            self.state_ang = 1
        elif self.right:
            self.state_ang = -1
        else:
            self.state_ang = 0

    def callback(self, msg):
        self.ranges = np.array(msg.ranges)

if __name__=="__main__":
    teleop = LidarAutoTeleop()
    teleop.run()
```

3.2.1. Запуск робота

Для запуска робота необходимо снова запустить контроллер, симулятор и rviz. После инициализации всех компонентов необходимо запустить узел для управления роботом в автоматическом режиме при помощи *roslaunch myrobot_teleop lidar_auto_teleop.launch*, предварительно создав launch-файл по аналогии с *ir_auto_keyboard.launch* из предыдущей лабораторной работы. Запуск узла в терминале представлен на рис. 3.3. В терминале также отображается различная информация о расстояниях с лидара, а также о принятых решениях робота при поворотах на перекрёстках.

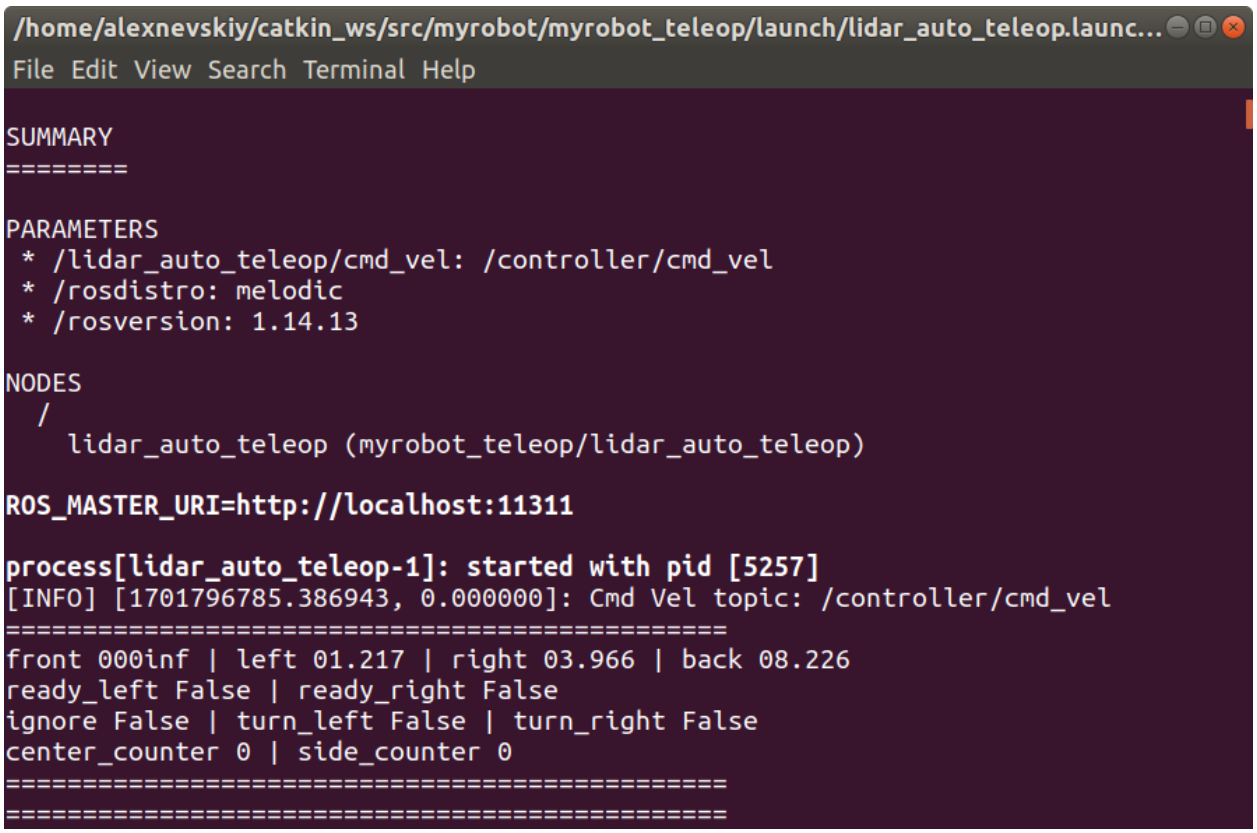

Рис. 3.3. Окно терминала с узлом управления в автоматическом режиме

В rviz мы можем наблюдать траекторию робота, после некоторого времени работы автоматического управления, как показано на рис. 3.4. Видно, что робот заехал в одну из двух комнат, а также при приближении к стене направляется в противоположную сторону от неё.

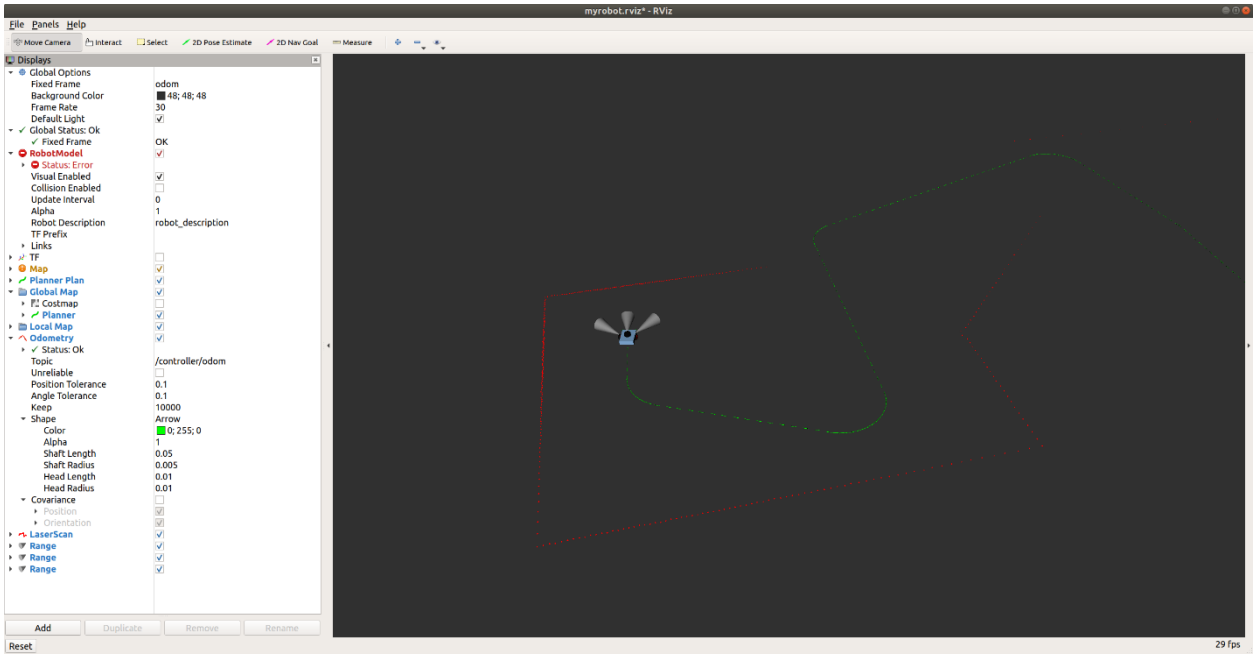

Рис. 3.4. Окно rviz с траекторией движения робота в автоматическом режиме

Пример управления роботом в автоматическом режиме в gazebo представлен на рис. 3.5. Видно, что робот выезжает из части карты, где расположены две комнаты.

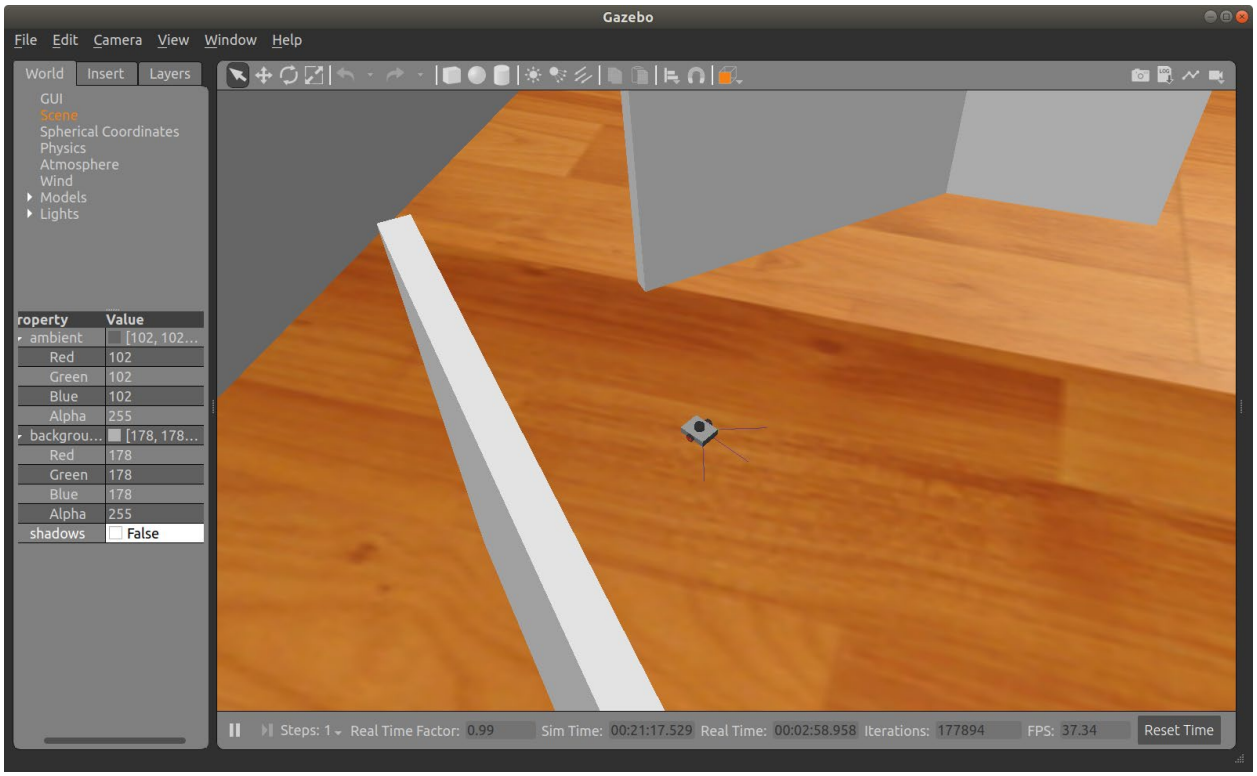

Рис. 3.5. Окно gazebo с автоматическим управлением роботом

4. ВЫВОДЫ

В ходе выполнения данной лабораторной работы разработан алгоритм автоматического управления и произведено описание узла управления роботом в автоматическом режиме при помощи лидара. Для этого написан скрипт на языке Python, в котором производится считывание расстояний с лидара (максимальное расстояние, которое улавливает лидар, равняется 12 метрам). На основе полученных данных определяется в каком направлении будет двигаться робот. Помимо этого робот может определять перекрёстки и поворачивать на одну из его дорог. Предложенный алгоритм автоматического управления предусматривает выезд из любых тупиковых ситуаций, а также въезд хотя бы в одну из двух комнат на предложенной карте *testwalls* за счёт намеренного поворота на перекрёстках.