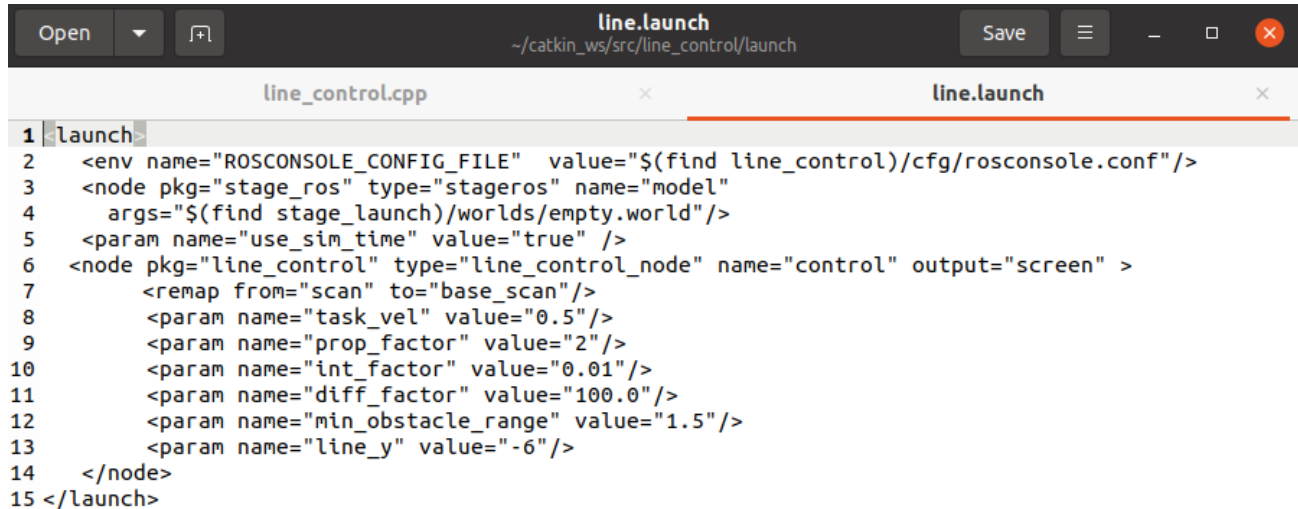


Отчет по лабораторной работе N3.

Группа:

- 1) Алхименков Леонид
- 2) Казаков Максимилиан
- 3) Хуан Бинкунь

launch :



```
1 <launch>
2   <env name="ROSCONSOLE_CONFIG_FILE" value="$(find line_control)/cfg/rosconsole.conf"/>
3   <node pkg="stage_ros" type="stageros" name="model"
4     args="$(find stage_launch)/worlds/empty.world"/>
5   <param name="use_sim_time" value="true" />
6   <node pkg="line_control" type="line_control_node" name="control" output="screen" >
7     <remap from="scan" to="base_scan"/>
8     <param name="task_vel" value="0.5"/>
9     <param name="prop_factor" value="2"/>
10    <param name="int_factor" value="0.01"/>
11    <param name="diff_factor" value="100.0"/>
12    <param name="min_obstacle_range" value="1.5"/>
13    <param name="line_y" value="-6"/>
14  </node>
15 </launch>
```

Коды:

"line control.cpp"

```
#include "line_control.h"
#include "math.h"
#include <std_msgs/Float64.h>

void LineControl::laserCallback(const sensor_msgs::LaserScan& msg)
{
    // проверим нет ли вблизи робота препятствия
    const double kMinObstacleDistance = 0.3;
    for (size_t i = 0; i<msg.ranges.size(); i++)
    {
        if ( msg.ranges[i] < kMinObstacleDistance )
        {
            obstacle = true;
            ROS_WARN_STREAM("OBSTACLE!!!");
            break;
        }
    }
}

void LineControl::poseCallback(const nav_msgs::Odometry& msg)
{
    ROS_DEBUG_STREAM("Pose msg: x = "<<msg.pose.pose.position.x<<
        " y = "<<msg.pose.pose.position.y<<
        " theta = "<<2*atan2(msg.pose.pose.orientation.z,
            msg.pose.pose.orientation.w) );
```

```

// обновляем переменные класса, отвечающие за положение робота
x = msg.pose.pose.position.x;
y = msg.pose.pose.position.y;
theta = 2*atan2(msg.pose.pose.orientation.z,
                msg.pose.pose.orientation.w);
}

double LineControl::cross_track_err_line()
{
    return line_y - y;
}

double LineControl::cross_track_err_line1()
{
    return line_y1 - y;
}

double LineControl::cross_track_err_circle()
{
    double dx = cx - x;
    double dy = cy - y;
    double e = sqrt(dx*dx + dy*dy) - R;
    return e;
}

double LineControl::cross_track_err_circle1()
{
    double dx = cx1 - x;
    double dy = cy1 - y;
    double e = sqrt(dx*dx + dy*dy) - R;
    return e;
}

void LineControl::publish_error(double e)
{
    std_msgs::Float64 err;
    err.data = e;
    err_pub.publish(err);
}

void LineControl::timerCallback(const ros::TimerEvent&)
{
    ROS_DEBUG_STREAM("on timer ");
    // сообщение с помощью которого задается
    // управление угловой и линейной скоростью
    geometry_msgs::Twist cmd;
    // при создании структура сообщения заполнена нулевыми значениями

    // если вблизи нет препятствия то задаем команды
    if ( !obstacle )
    {
        // вычислим текущую ошибку управления
        double err;
        int znak = 1;
        if(x > 0)
        {
            err = cross_track_err_circle();
        }

        if(x <= 0 && y < 0 && x > -12)
        {
            err = cross_track_err_line();
        }
        if(x <= 0 && y > 0 && x > -12)

```

```

    {
        err = cross_track_err_line1();
        znak = -1;
    }
    if(x <= -12)
    {
        err = cross_track_err_circle1();
        znak = 1;
    }

    // публикация текущей ошибки
    publish_error(err);
    // интегрируем ошибку
    int_error += err;
    // дифференцируем ошибку
    double diff_error = err - old_error;
    // запоминаем значение ошибки для следующего момента времени
    old_error = err;
    cmd.linear.x = task_vel;
    // ПИД регулятор угловой скорости  $w = k \cdot err + k_i \cdot \text{int\_err} + k_d \cdot \text{diff\_err}$ 
    cmd.angular.z = znak*(prop_factor * err + int_factor*int_error +
diff_error * diff_factor);
    ROS_DEBUG_STREAM("error = "<<err<<" cmd v="<<cmd.linear.x<<" w =
"<<cmd.angular.z);
    }
    // отправляем (публикуем) команду
    cmd_pub.publish(cmd);
}

LineControl::LineControl():
    int_error(0.0),
    old_error(0.0),
    obstacle(false),
    node("~")
{
    ROS_INFO_STREAM("LineControl initialisation");
    // читаем параметры
    line_y = node.param("line_y", -6.0);
    line_y1 = node.param("line_y1", 6.0);
    cx = node.param("cx", 0);
    cy = node.param("cy", 0);
    cx1 = node.param("cx1", -12);
    cy1 = node.param("cy1", 0);
    R = node.param("R", 6);
    task_vel = node.param("task_vel", 1.0);
    prop_factor = node.param("prop_factor", 0.1);
    int_factor = node.param("int_factor", 0.0);
    diff_factor = node.param("diff_factor", 0.0);
    min_obstacle_range = node.param("min_obstacle_range", 1.0);
    double dt = node.param("dt", 0.1);

    // подписываемся на необходимые данные
    laser_sub = node.subscribe("/scan", 100, &LineControl::laserCallback, this);
    pose_sub = node.subscribe("/base_pose_ground_truth", 100,
&LineControl::poseCallback, this);
    timer1 = node.createTimer(ros::Duration(dt), &LineControl::timerCallback,
this);
    cmd_pub = node.advertise<geometry_msgs::Twist>("/cmd_vel", 100);
    err_pub = node.advertise<std_msgs::Float64> ("/err", 100);
}

```

"line_control.h"

```
#ifndef LINE_CONTROL_H
#define LINE_CONTROL_H
#include <ros/ros.h>
#include "sensor_msgs/LaserScan.h"
#include "nav_msgs/Odometry.h"

class LineControl
{
public:
    LineControl();
    // секция приватных функций
private:
    // функция вычисления ошибки управления для движения вдоль прямой
    double cross_track_err_line();
    // функция вычисления ошибки управления для движения вдоль окружности
    double cross_track_err_circle();
    double cross_track_err_oval();
    double cross_track_err_line1();
    double cross_track_err_circle1();
    /**
     * Функция, которая будет вызвана
     * при получении данных от лазерного дальномера
     */
    void laserCallback(const sensor_msgs::LaserScan& msg);
    /**
     * Функция, которая будет вызвана при
     * получении сообщения с текущим положением робота
     */
    void poseCallback(const nav_msgs::Odometry& msg);
    /**
     * функция обработчик таймера
     */
    void timerCallback(const ros::TimerEvent&);
    // функция публикации ошибки
    void publish_error(double e);
    // секция приватных членов
private:
    // заданная координата линии, вдоль которой должен двигаться робот
    double line_y;
    double line_y1;
    double cx, cy, R, cx1, cy1;
    // заданная скорость движения
    double task_vel;
    // пропорциональный коэффициент регулятора обратной связи
    double prop_factor;
    // интегральный коэффициент регулятора
    double int_factor;
    // дифференциальный коэффициент регулятора
    double diff_factor;
    // интеграл ошибки
    double int_error;
    // старое значение ошибки
    double old_error;
    // минимально допустимое значение расстояния до препятствия
    double min_obstacle_range;
    // флаг наличия препятствия
    bool obstacle;
    // положение робота
    double x, y, theta;

    // объекты, обеспечивающие связь с ros - должны существовать все время жизни
    приложения
    // объект NodeHandle, через который создаются подписки и публикаторы
    ros::NodeHandle node;
```

```
// публикатор команд управления
ros::Publisher cmd_pub;
// публикатор текущей ошибки управления
ros::Publisher err_pub;
// подписчики
ros::Subscriber laser_sub;
ros::Subscriber pose_sub;
ros::Timer timer1;

};

#endif // LINE_CONTROL_H
```