Отчет по лабораторной работе N2.

Группа:
СМ7-63Б
1) Алхименков Леонид
2) Казаков Максимилиан
3) Хуан Бинкунь

1. Модифицировать алгоритм, чтобы робот двигался прямолинейно до сближения с препятствием, а при достижении препятствия поворачивался на месте до тех пор пока препятствие не "исчезнет" из поля зрения дальномера, а затем продолжал прямолинейное движение в получившемся направлении (алгоритм жука)
2. Модифицировать алгоритм, чтобы робот двигался вдоль правой/левой стены на заданном расстоянии (можно инициализировать робот стоящим около стены)

Код, который может одновременно реализовать вопрос 1.2

robot_control.cpp

```
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"
#include "nav_msgs/Odometry.h"

bool obstacle = false;
ros::Publisher pub;
bool angle;
int error_rate = 25;
/**
 * Функция, которая будет вызвана
 * при получении данных от лазерного дальномера
 * параметр функции msg - ссылка на полученное сообщение
 */
void laserCallback(const sensor_msgs::LaserScan& msg)
{
  ROS_DEBUG_STREAM("Laser msg: "<<msg.scan_time);

  const double kMinRange = 0.4;
  obstacle = false;
  //проверим нет ли вблизи робота препятствия
  for (size_t i = 0; i<msg.ranges.size(); i++)
  {
      if (msg.ranges[i] < kMinRange)
      {
      obstacle = true;
      if ( i < msg.ranges.size()/2 ){
        angle = true;
      } else {
        angle = false;
      }
      break;
      }
  }

}
```

```
/**
 * Функция, которая будет вызвана при
 * получении сообщения с текущем положением робота
 * параметр функции msg - ссылка на полученное сообщение
 */

void 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) );
}
/*
 * функция обработчик таймера
 * параметр функции - структура, описывающая событие таймера, здесь не используется
 */
void timerCallback(const ros::TimerEvent&)
{
    static int counter = 0;
    counter++;
    ROS_DEBUG_STREAM("on timer "<<counter);
    //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)
    {
        ROS_INFO_STREAM("go forward");
        cmd.linear.x = 0.5;
      if(error_rate<15 && angle){
        cmd.angular.z = -0.5;
        error_rate++;
      } else if(error_rate<15 && !angle){
        cmd.angular.z = 0.5;
        error_rate++;
      }
    }
  else
    {
      error_rate = 0;
      if (angle == true)
      {
      ROS_INFO_STREAM("go left");
        cmd.linear.x = -0.1;
        cmd.angular.z = 0.5;
      }
      else
      {
        ROS_INFO_STREAM("go right");
        cmd.linear.x = -0.1;
        cmd.angular.z = -0.5;
        }
    }
    //отправляем (публикуем) команду
    pub.publish(cmd);
}

int main(int argc, char **argv)
{
  obstacle = false;
  /**
   * Инициализация системы сообщений ros
   * Регистрация node с определенным именем (третий аргумент функции)
   * Эта функция должна быть вызвана в первую очередь
```

```
   */
  ros::init(argc, argv, "control_node");

  /**
   * NodeHandle  - объект через который осуществляется взаимодействие с ROS:
   * передача сообщений
   * регистрация коллбаков (функций обработки сообщений)
   */
  ros::NodeHandle n;

  /**
   * subscribe() функция подписки на сообщения определенного типа,
   * передаваемое по заданному топику
   * В качестве параметров указываются
   * -  топик - на сообщения которого происходит подписка
   * -  длина очереди сообщений хранящихся до обработки (если очередь заполняется,
   *  то самые старые сообщения будут автоматически удаляться )
   *  -  функция обработки сообщений
   *
   *
   *  Подписываемся на данные дальномера

   */
  ros::Subscriber laser_sub = n.subscribe("base_scan", 1, laserCallback);

  /*
   * Подписываемся на данные о положении робота
   */
  ros::Subscriber pose_sub = n.subscribe("base_pose_ground_truth", 1,
poseCallback);


  /*
   * Регистрируем функцию обработчик таймера 10Hz
   */
  ros::Timer timer1 = n.createTimer(ros::Duration(0.001), timerCallback);

  /*
   * Сообщаем, что мы будем публиковать сообщения типа Twist по топику cmd_vel
   * второй параметр - длина очереди
   */
  pub = n.advertise<geometry_msgs::Twist>("cmd_vel", 1);

   /**
   * ros::spin() функция внутри которой происходит вся работа по приему сообщений
   * и вызову соответствующих обработчиков . Вся обработка происходит из основного потока
   * (того, который вызвал ros::spin())
   * Функция будет завершена, когда подьзователь прервет выполнение процесса с Ctrl-C
   *
   */
  ros::spin();

  return 0;
}
```