Отчет по лабораторной работе N5.

Группа:
1) Алхименков Леонид
2) Казаков Максимилиан
3) Хуан Бинкунь

simple_map.cpp

```
#include <ros/ros.h>
#include <sensor_msgs/                                    LaserScan.h>
#include <nav_msgs/OccupancyGrid.h>
#include <tf/transform_listener.h>

#include "simple_map/scan_to_map.h"

//глобальная переменная - публикатор сообщения карты
ros::Publisher mapPub;

//глоабльный указатель на tfListener, который будет проинициализирован в main
tf::TransformListener *tfListener;

//имя для СК карты
std::string map_frame;

//разрешение карты
double map_resolution = 0.1;
//размер карты в клетках
int map_width = 100;
int map_height = 100;

void drawLine(int x1, int y1, int x2, int y2, nav_msgs::OccupancyGrid& map_msg) {
    const int deltaX = abs(x2 - x1);
    const int deltaY = abs(y2 - y1);
    const int signX = x1 < x2 ? 1 : -1;
    const int signY = y1 < y2 ? 1 : -1;
    int error = deltaX - deltaY;
    while(x1 != x2 || y1 != y2)
   {
        map_msg.data[ y1* map_width + x1] = 0;
        int error2 = error * 2;
        if(error2 > -deltaY)
        {
            error -= deltaY;
            x1 += signX;
        }
        if(error2 < deltaX)
        {
            error += deltaX;
            y1 += signY;
        }
    }

}



void prepareMapMessage(nav_msgs::OccupancyGrid& map_msg, const ros::Time& stamp)
{
    map_msg.header.frame_id = map_frame;
    map_msg.header.stamp = stamp;
```

```
    map_msg.info.height = map_height;
    map_msg.info.width = map_width;
    map_msg.info.resolution = map_resolution;

    // изменяем размер вектора, который является хранилищем данных карты, и заполняем
его значением (-1) - неизвестное значение
    map_msg.data.resize(map_height*map_width, -1);
}

bool determineScanTransform(tf::StampedTransform& scanTransform,
                            const ros::Time& stamp,
                            const std::string& laser_frame)
{
    try
    {
        if ( ! tfListener->waitForTransform(map_frame,
                                            laser_frame,
                                            stamp,
                                            ros::Duration(0.1)) )
        {
          ROS_WARN_STREAM("no transform to scan "<<laser_frame);
          return false;
        }
        tfListener->lookupTransform(map_frame,
                                    laser_frame,
                                    stamp,
                                    scanTransform);

    }
    catch (tf::TransformException& e)
    {
        ROS_ERROR_STREAM("got tf exception "<<e.what());
        return false;
    }
    return true;
}


/**
 * Функция, которая будет вызвана
 * при получении данных от лазерного дальномера
 */
void laserCallback(const sensor_msgs::LaserScan& scan)
{
    tf::StampedTransform scanTransform;
    const std::string& laser_frame = scan.header.frame_id;
    const ros::Time& laser_stamp = scan.header.stamp;
    if (!determineScanTransform(scanTransform, laser_stamp, laser_frame)) {
        return;
    }

    //создаем сообщение карты
    nav_msgs::OccupancyGrid map_msg;
    //заполняем информацию о карте - готовим сообщение
    prepareMapMessage(map_msg, laser_stamp);

    //положение центра дальномера в СК дальномера
    tf::Vector3 zero_pose(0,0,0);
    //положение дальномера в СК карты
    tf::Vector3 scan_pose = scanTransform(zero_pose);
    ROS_DEBUG_STREAM("scan pose "<<scan_pose.x()<<" "<<scan_pose.y());

    //задаем начало карты так, чтобы сканнер находился в центре карты
    map_msg.info.origin.position.x = scan_pose.x() - map_width *
map_resolution /2.0;
```

```
    map_msg.info.origin.position.y = scan_pose.y() - map_height * map_resolution
/2.0;

    //индексы карты, соответствующие положению центра лазера
    int y1 = (scan_pose.y() - map_msg.info.origin.position.y ) / map_resolution;
    int x1 = (scan_pose.x() - map_msg.info.origin.position.x ) / map_resolution;
    ROS_DEBUG_STREAM("publish map "<<x1<<" "<<y1);
    // в клетку карты соотвтествующую центру лазера - записываем значение 0
    map_msg.data[ y1* map_width + x1] = 0;
    bool full = false;
    int x2, y2;
    for (int i = 0; i < scan.ranges.size(); ++i) {
        if (scan.ranges[i] == scan.range_max)
            full = true;
        double angle = scan.angle_min + scan.angle_increment * i;
        tf::Vector3 v = scanTransform(tf::Vector3(scan.ranges[i]*cos(angle),
scan.ranges[i]*sin(angle), 0));
        v -= tf::Vector3(map_msg.info.origin.position.x,
map_msg.info.origin.position.y, 0);
        if (v.x() >= 0 && v.y() >= 0 &&
                v.x() < map_msg.info.resolution * map_msg.info.width && v.y() <
map_msg.info.resolution * map_msg.info.height) {
            x2 = v.x() / map_msg.info.resolution;
            y2 = v.y() / map_msg.info.resolution;
            map_msg.data[ y2* map_width + x2] = 100;
            drawLine (x1, y1, x2, y2, map_msg);
        }
        full = false;
    }

    // публикуем сообщение с построенной картой
    mapPub.publish(map_msg);
}


int main(int argc, char **argv)
{
  /**
   * 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 node("~");

  //читаем параметры
  map_frame = node.param<std::string>("map_frame", "odom");
  map_resolution = node.param("map_resolution", map_resolution);
  map_width = node.param("map_width", map_width);
  map_height = node.param("map_height", map_height);

  //создание объекта tf Listener
  tfListener = new tf::TransformListener;

  // Подписываемся на данные дальномера
  ros::Subscriber laser_sub = node.subscribe("/scan", 100, laserCallback);

  //объявляем публикацию сообщений карты
  //Используем глобальную переменную, так как она понядобится нам внутр функции -
обработчика данных лазера
```

```
  mapPub = node.advertise<nav_msgs::OccupancyGrid>("/simple_map", 10);


   /**
   * 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(), то есть основного в данном случае)
   * Функция будет завершена, когда подьзователь прервет выполнение процесса с Ctrl-C
   *
   */
  ros::spin();

  return 0;
}
```