МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

КУРСОВАЯ РАБОТА

По дисциплине «Системы реального времени на основе Linux»

Тема: Коррекция прицела

Студенты гр. 3304, 3303 _____ Быков А.Ю

_____ Фабер В.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2018

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студенты: Быков А.Ю., Фабер В.В.

Группы: 3303, 3304

Тема работы: Коррекция прицела

Исходная постановка задачи:

Установить прицел на цели, вести цель, поразить цель.

Исходные данные:

Оперативник с снайперской винтовкой уничтожает цели на поле боя. Цели появляются хаотически. Задача поразить как можно больше целей за время работы программы.

Ограничения на исходные данные:

- Цели передвигаются в трёхмерном мире из стороны в сторону (слева направо, справа налево, снизу вверх, по диагонали и прочее). До каждой из целей есть своё расстояние. Пуля летит с какой-то скоростью и ей потребуется время, на преодоление этого расстояния. Стрелять необходимо с упреждением.
- Цели движутся равномерно прямолинейно.
- Открывать огонь можно только при получении соответствующей команды от человека-оператора. Если команды нет какое-то время, переключиться на другую цель.
- На стрельбище присутствуют препятствия, попадание в которых не приводит к поражению цели.

Студенты гр. 3304, 3303 _____ Быков А.Ю

_____ Фабер В.В.

Преподаватель _____ Филатов А.Ю.

АННОТАЦИЯ

В данном отчете приведено описание реализация программы «Корректировка прицела», в которой стрелок ведет огонь по постоянно движущимся мишеням. Мишени передвигаются в трехмерном мире из стороны в сторону, стрелок, в свою очередь, ведет огонь на упреждение.

SUMMARY

This report describes the implementation of the “Sight Adjustment” program, in which the shooter fires on constantly moving targets. Targets move from side to side in the three-dimensional world, the shooter, in turn, fires on the lead.

ВВЕДЕНИЕ

Необходимо реализовать программу, в которой роботу-стрелку требуется поражать мишени. Причем выстрел производится по команде от человека.

Ниже приведен список задач, которые потребуется решить в ходе реализации программы:

- Создать механизм для появления на поле мишеней
- Добавить мишеням возможность движения в случайном направлении
- Реализовать стрелка, способного поражать эти мишени
- Реализовать механизм для отдачи стрелку приказа на выстрел

Ограничения на исходные данные:

- Цели передвигаются в трёхмерном мире из стороны в сторону (слева направо, справа налево, снизу вверх, по диагонали и прочее). До каждой из целей есть своё расстояние. Пуля летит с какой-то скоростью и ей потребуется время, на преодоление этого расстояния. Стрелять необходимо с упреждением.
- Цели движутся равномерно прямолинейно.
- Открывать огонь можно только при получении соответствующей команды от человека-оператора. Если команды нет какое-то время, переключиться на другую цель.
- На стрельбище присутствуют препятствия, попадание в которых не приводит к поражению цели.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

ROS (Robot Operating System) — Операционная система для роботов — это фреймворк для программирования роботов, предоставляющий функциональность для распределённой работы. ROS был первоначально разработан в 2007 году под названием switchyard в Лаборатории Искусственного Интеллекта тэнфордского Университета для проекта (STAIR). В 2008 году развитие продолжается в Willow Garage, научноисследовательском институте/инкубаторе робототехники, совместно с более чем двадцатью сотрудничающими институтами.

ROS обеспечивает стандартные службы операционной системы, такие как: аппаратную абстракцию, низкоуровневый контроль устройств, реализацию часто используемых функций, передачу сообщений между процессами, и управление пакетами. ROS основан на архитектуре графов, где обработка данных происходит в узлах, которые могут получать и передавать сообщения между собой. Библиотека ориентирована на Unix-подобные системы (Ubuntu Linux включен в список «поддерживаемых», в то время как другие варианты, такие как Fedora и Mac OS X, считаются «экспериментальными»).

ROS имеет две основные «стороны»: стороны операционной системы ros, как описано выше и ros-pkg, набор поддерживаемых пользователями пакетов (организованных в наборы, которые называются стек), которые реализуют различные функции робототехники: SLAM, планирование, восприятие, моделирование и др.

ROS выпускается в соответствии с условиями BSD-лицензии и с открытым исходным кодом. ROS бесплатен для использования, как в исследовательских, так и в коммерческих целях. мощности стали использоваться более эффективно, тем самым уменьшая сопутствующие затраты.

2. ИСПОЛЬЗУЕМЫЕ КОМПОНЕНТЫ

2.1. Описание топиков

- **/set_model_state** – в данный топик публикуются координаты модели, которая будет помещена на «сцену»;
- **/spawn_sdf_model** – в данный топик публикуется описание модели, которая будет помещена на «сцену»;
- **/shooting_signal** – в данный топик посылается сообщение, обозначающее что была получена команда на выстрел;

2.2. Описание сообщений

- **ModelState** – с помощью данного сообщения задаются координаты модели, которое публикуется в топик set_model_state;
- **SpawnModel** – данное сообщение публикуется в топик spawn_sdf_model и содержит в себе всю необходимую информацию о модели, которую добавляют на «сцену»;
- **Empty –** данное сообщение публикуется в топик shooting_signal. Оно не несет в себе никакой информации, важным является сам факт получения такого сообщения.

2.3. Описание пакета tf

tf – это пакет, который позволяет пользователю отслеживать несколько координатных объектов с течением времени. tf поддерживает связь между координатами объектов, позволяет пользователю преобразовывать точки, векторы и т. д. между любыми двумя объектами в любой момент времени.

tf позволяет отвечать на такие вопросы, как:

- Где был объект относительно мировых координат, 5 секунд назад?
- Каковы координаты рассматриваемого объекта относительно другого
- объекта?
- Каково текущее положение объекта на карте местности?

Используемые классы

tf::Transform – предоставляет функционал для преобразования координат;

tf::TransformBroadcaster – позволяет публиковать преобразованные координаты;

tf::TransformListener – позволяет получать преобразованные координаты;

tf::StampedTransform – предоставляет тип сообщения, который публикуется с преобразованными координатами с использованием класса TransformBroadcaster

tf::Quaternion – данный класс реализует кватернион для выполнения линейных вращений (поворотов) объекта.

3. РАЗРАБОТКА ПРОГРАММЫ

В ходе реализации программы инициализируется сцена в Gazebo, на которую добавляются мишени в виде мисок (Bowl) и снаряды в виде шариков (Cricket Ball).

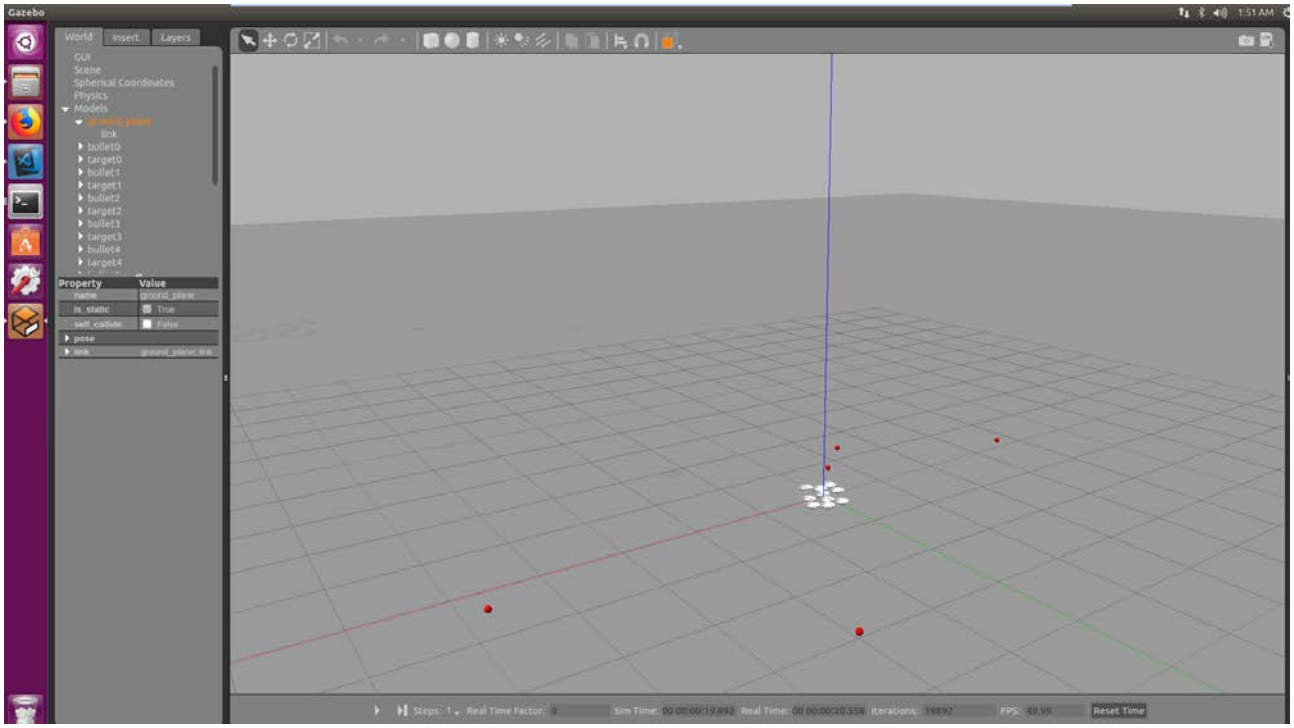

Рисунок 1 – Стрельбище

Для использования пользовательского ввода была релизована отдельная нода. Она принимает ввод с клавиаутуы, и если была введена буква «s», публикует пустое сообщение в топик **/shooting_signal**. Скриншот терминала с запущенной нодой приведен на рисунке 2.

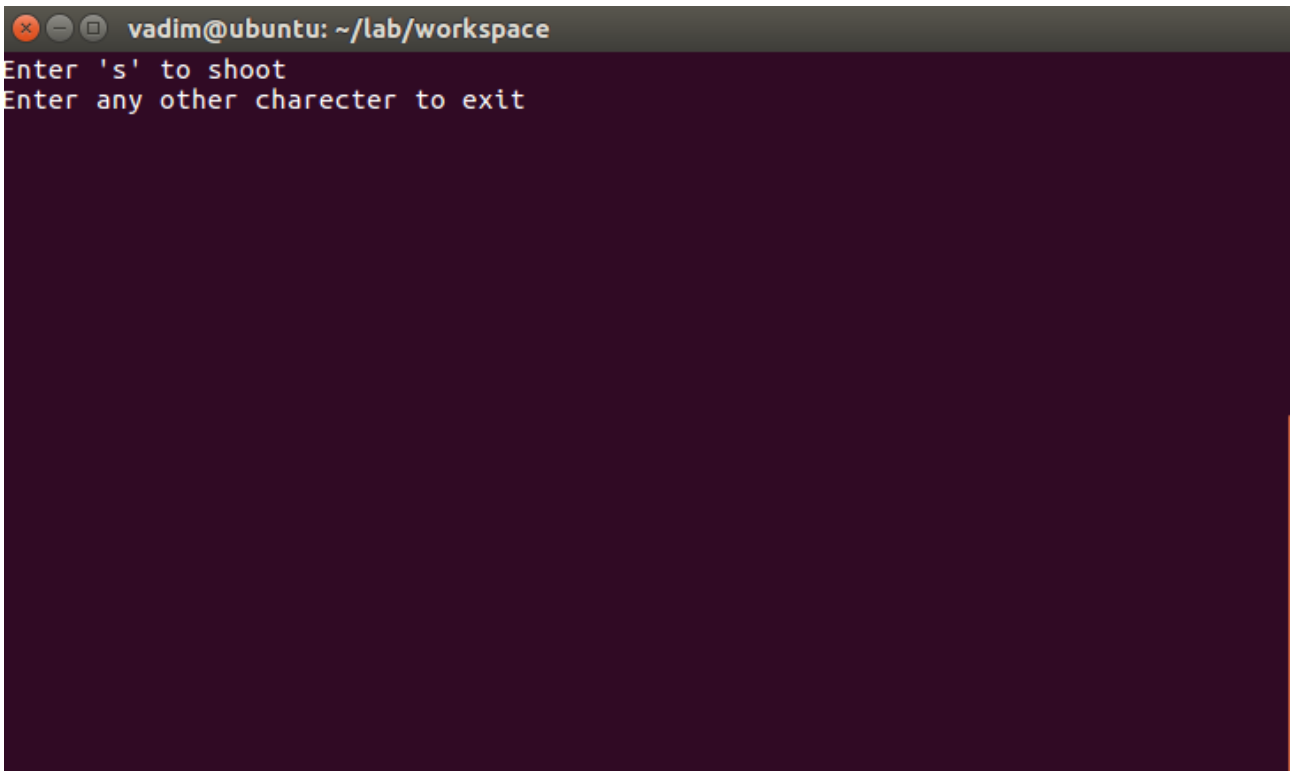

Рисунок 2 – Ввода команды для стрельбы

Вторая нода является ключевой. Внутри нее инициализируются модели для размещения на сцене и происходит непосредственно стрельба. Эта нода подписана на топик **/shooting_signal** и, при обнаружении нового сообщения в этом топике, производит выстрел. Скриншот терминала с запущенной нодой приведен на рисунке 3.

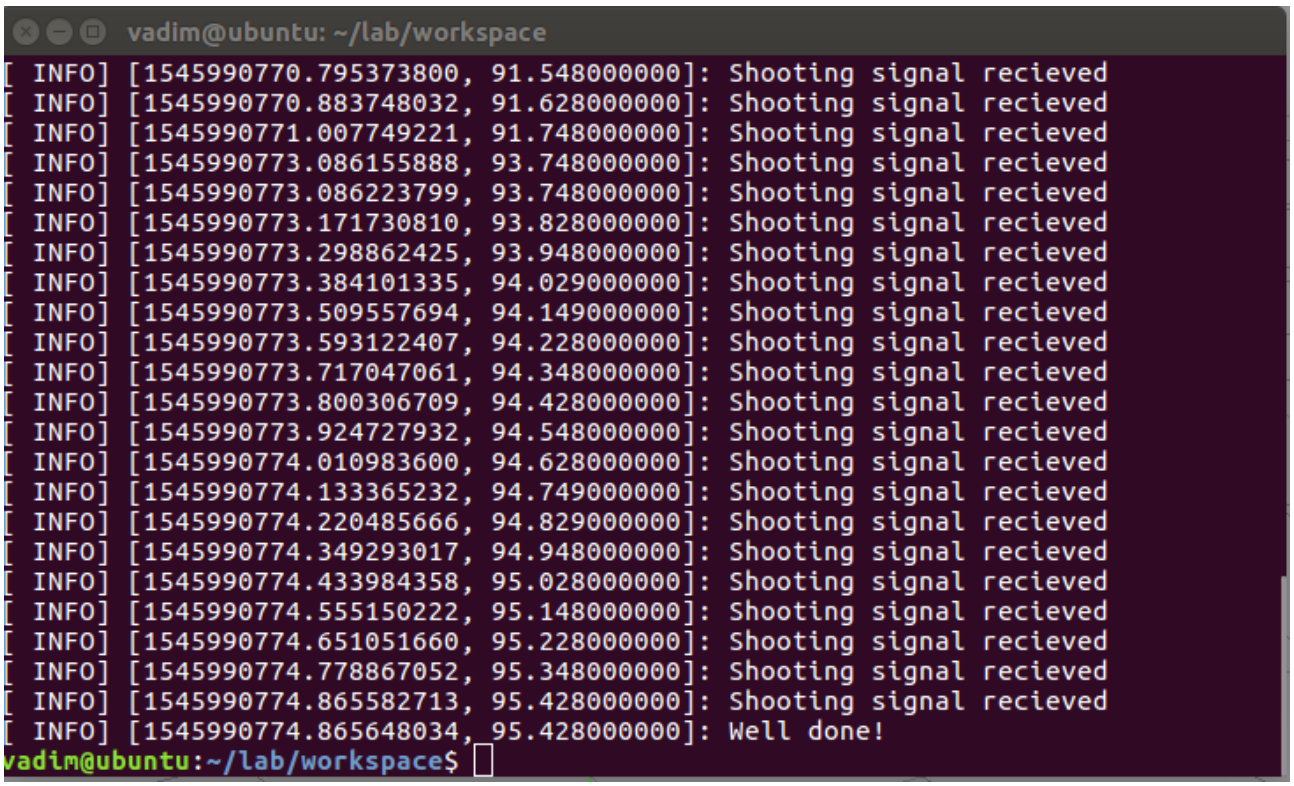

Рисунок 3 – Окно вывода стрелка

На рисунке 4 представлен график rqt_graph со всеми нодами и топиками, используемыми в программе.

Рисунок 4 – Ноды и топики программы

ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта была разработана программа, в которой решается проблема стрельбы по мишеням с упреждением.

Список задач, которые были решены:

- Создан механизм для появления на поле мишеней
- Мишеням добавлена возможность движения в случайном направлении
- Реализован стрелок, способный поражать эти мишени
- Реализован механизм для отдачи стрелку приказа на выстрел

ПРИЛОЖЕНИЕ А

ЛИСТИНГ ПРОГРАММЫ

course_work_key.cpp

```
#include "ros/ros.h"
#include "std_msgs/Empty.h"

int main(int argc, char **argv)
{

    ros::init(argc, argv, "course_work_key");

    ros::NodeHandle n;

    ros::Publisher pub = n.advertise<std_msgs::Empty>("shooting_signal", 1000);

    ros::Rate rate(10);

    while (ros::ok())
    {
        system("clear");

        std::cout << "Enter 's' to shoot" << std::endl;
        std::cout << "Enter any other charecter to exit" << std::endl;

        char value;
        std::cin.clear();
        std::cin >> value;
        if(value == 's' || value == 'S') {
            ROS_INFO("Send shooting signal");

            std_msgs::Empty msg;

            pub.publish(msg);
        } else {
            ros::shutdown();
        }

        rate.sleep();
    }

    return 0;
}
```

course_work_node.cpp

```
#include "ros/ros.h"
#include <tf/transform_broadcaster.h>
#include <tf/transform_listener.h>
#include <std_msgs/String.h>
#include "gazebo_msgs/SpawnModel.h"
#include "gazebo_msgs/ModelState.h"
#include <fstream>
#include "string.h"
#include "std_msgs/Empty.h"
#include <cstdlib>
#include <vector>

using namespace std;

int shoot = 0;

struct TargetInit {
    double speed;
    double y0;
    double alpha;
    long time0;

    void init(long _time0) {
        time0 = _time0;
        generateInitValues();
    }

    void generateInitValues() {
        double f = (double)rand() / RAND_MAX;
        alpha = f * M_PI;

        y0 = rand() % 20 + 10;

        f = (double)rand() / RAND_MAX;
        speed = 0.01 + f * 0.05;
    }

    void print() {
        std::cout << "Bullet init: speed = " << speed << " ; y0 = " << y0 << " ;
alpha = " << alpha << " ; time0 = " << time0 << ";"  << std::endl;
    }
};

struct BulletInit {
    double speed;
    double theta;
    long time0;

    void init(TargetInit &target_init, double _speed, long _time0) {
        speed = _speed;
        time0 = _time0;

        double beta = atan(target_init.speed * (time0 - target_init.time0) /
target_init.y0);
        theta = asin(target_init.speed / speed * cos(beta)) + beta;
    }

    void print() {
        std::cout << "Bullet init: speed = " << speed << " ; theta = " << theta
<< " ; time0 = " << time0 << ";" << std::endl;
```

```
    }
};

struct Coordinates {
    double x;
    double y;
    double z;

    void init(double _x, double _y, double _z) {
        x = _x;
        y = _y;
        z = _z;
    }

    void print(std::string beginning) {
        std::cout << beginning << " : ( " << x << " ; " << y << " ; " << z << "
)" << std::endl;
    }
};

Coordinates calculateTarget(TargetInit target, long time) {
    Coordinates result;
    result.x = target.speed * (time - target.time0) * cos(target.alpha);
    result.y = target.y0;
    result.z = target.speed * (time - target.time0) * sin(target.alpha);
    return result;
}

Coordinates calculateBullet(BulletInit bullet, TargetInit target, long time) {
    Coordinates result;
    result.x = bullet.speed * ((double)time - bullet.time0) * sin(bullet.theta)
* cos(target.alpha);
    result.y = bullet.speed * ((double)time - bullet.time0) * cos(bullet.theta);
    result.z = bullet.speed * ((double)time - bullet.time0) * sin(bullet.theta)
* sin(target.alpha);
    return result;
}

void updatePose(geometry_msgs::Pose &pose, float x, float y, float z) {
    pose.position.x = x;
    pose.position.y = y;
    pose.position.z = z;

    pose.orientation.x = 1 * sin(M_PI / 2.0 / 2);
    pose.orientation.y = 0;
    pose.orientation.z = 0;
    pose.orientation.w = cos(M_PI / 2.0 / 2);
}

gazebo_msgs::SpawnModel createModel(string model, string model_name) {
    gazebo_msgs::SpawnModel srv_target;
    ifstream fin("/home/andrey/.gazebo/models/" + model + "/model.sdf");

    string model_target;
    string buf;
    while (!fin.eof()) {
        getline(fin, buf);
        model_target += buf + "\n";
    }
    srv_target.request.model_xml = model_target;
    srv_target.request.model_name = model_name;
    return srv_target;
```

```
}

gazebo_msgs::ModelState createModelState(string model_name, Coordinates coord) {
    gazebo_msgs::ModelState msg;
    msg.model_name = model_name;
    updatePose(msg.pose, coord.x, coord.y, coord.z);
    return msg;
}

void chatterCallback(const std_msgs::Empty::ConstPtr& msg)
{
    ROS_INFO("Shooting signal recieved");
    shoot++;
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "course_work");

    srand(time(0));

    ros::NodeHandle nh;

    ros::Subscriber sub = nh.subscribe("shooting_signal", 1000,
chatterCallback);
    ros::Publisher pub =
nh.advertise<gazebo_msgs::ModelState>("gazebo/set_model_state", 10);

    ros::service::waitForService("gazebo/spawn_sdf_model");
    ros::ServiceClient add_robot =
nh.serviceClient<gazebo_msgs::SpawnModel>("gazebo/spawn_sdf_model");

    gazebo_msgs::SpawnModel srv;

    int n = 10;

    for(int i = 0; i < n; i++) {
        srv = createModel("cricket_ball", "bullet" + to_string(i));
        add_robot.call(srv);

        srv = createModel("bowl", "target" + to_string(i));
        add_robot.call(srv);
    }

    ros::Rate rate(25);

    double bullet_speed = 0.4;
    long time = 0;

    std::vector<TargetInit> target_inits;
    std::vector<BulletInit> bullet_inits;
    std::vector<Coordinates> targets;
    std::vector<Coordinates> bullets;


    for(int i = 0; i < n; i++) {
        TargetInit target_init;
        BulletInit bullet_init;
        Coordinates target;
        Coordinates bullet;

        target_init.init(time);
        bullet_init.init(target_init, bullet_speed, time);
```

```
        target.init(0, target_init.y0, 0);
        bullet.init(0, 0, 0);

        target_inits.push_back(target_init);
        bullet_inits.push_back(bullet_init);
        targets.push_back(target);
        bullets.push_back(bullet);
    }

    while (ros::ok()) {
      int count = 0;
        for(int i = 0; i < n; i++) {
            TargetInit target_init = target_inits[i];
            BulletInit bullet_init = bullet_inits[i];
            Coordinates target = targets[i];
            Coordinates bullet = bullets[i];

            if(shoot <= i) {
                bullet_init.init(target_init, bullet_speed, time);
                bullet_inits[i] = bullet_init;
            }

            if(bullet.y < target.y) {
                target = calculateTarget(target_init, time);
                targets[i] = target;
                pub.publish(createModelState("target" + to_string(i), target));

                if(shoot > i) {
                    bullet = calculateBullet(bullet_init, target_init, time);
                    bullets[i] = bullet;
                    pub.publish(createModelState("bullet" + to_string(i),
bullet));
                }
            } else {
                count++;
            }
        }

        if(count == n) {
            ROS_INFO("Well done!");
            break;
        }

        time++;
        rate.sleep();

        ros::spinOnce();
    }
    return 0;
}
```

package.xml

```
<?xml version="1.0"?>
<package>
  <name>course_work</name>
  <version>0.0.0</version>
  <description>The course_work package</description>

  <maintainer email="andrey@todo.todo">andrey</maintainer>
  <license>TODO</license>

    <buildtool_depend>catkin</buildtool_depend>

    <build_depend>roscpp</build_depend>
    <build_depend>gazebo_ros</build_depend>

    <run_depend>roscpp</run_depend>
    <run_depend>gazebo_ros</run_depend>

</package>
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(course_work)

find_package(catkin REQUIRED COMPONENTS
        roscpp
        tf
        geometry_msgs
        gazebo_ros
        )

catkin_package(
        DEPENDS gazebo_ros
        )

include_directories(
        ${catkin_INCLUDE_DIRS}
        ${gazebo_INCLUDE_DIRS}
        ${SDFormat_INCLUDE_DIRS}
        )

add_executable(${PROJECT_NAME}_node src/${PROJECT_NAME}_node.cpp)
add_executable(${PROJECT_NAME}_key src/${PROJECT_NAME}_key.cpp)

target_link_libraries(${PROJECT_NAME}_node ${catkin_LIBRARIES})
target_link_libraries(${PROJECT_NAME}_key ${catkin_LIBRARIES})
```