```
//------------------------------------------------------------------------

// INCLUDE LIBRARIES

//------------------------------------------------------------------------



#include <ros/ros.h>

#include <geometry_msgs/PoseStamped.h>

#include <geometry_msgs/Twist.h>



#include <mavros_msgs/State.h>



#include <unistd.h>

#include <mavros_msgs/CommandBool.h>

#include <mavros_msgs/CommandTOL.h>

#include <mavros_msgs/SetMode.h>
```

```
#include <mavros_msgs/State.h>
#include <sensor_msgs/Range.h>
//------------------------------------------------------------------------

// CONSTANTS AND VARIABLES

//------------------------------------------------------------------------




// Current position

float   x;

float   y;

float   z;

// Position error

float   e_wx;

float   e_wy;

float   e_wz;

//landing posit

// Position error
```

```
float   e_x;

float   e_y;

float   e_z;

//landing position error

float   e_ax;

float   e_ay;

float   e_az;

// P controller gain

float   K=1;
// parameters for altitude
float altezza;

bool atterraggio=false;

float Quota;

ros::Publisher                  local_vel_pub;
ros::Subscriber                 state_sub;
ros::Subscriber                 pos_sub;
geometry_msgs::PoseStamped  local_position;
geometry_msgs::Twist       vel;
mavros_msgs::State        current_state;
mavros_msgs::State        previous_state;
```

```
//------------------------------------------------------------------------

// FUNCTIONS

//------------------------------------------------------------------------

// ------------------------------------------------STATO-----------------------------------------------------------

void state_cb(const mavros_msgs::State::ConstPtr& msg){

    current_state = *msg;

    if (previous_state.mode != "OFFBOARD" && current_state.mode == "OFFBOARD")

        ROS_INFO("OFFBOARD enabled");

    if (previous_state.mode == "OFFBOARD" && current_state.mode != "OFFBOARD")

        ROS_INFO("OFFBOARD disabled");

    previous_state = *msg;

}

// ------------------------------------------------LOCAL POSITION--------------------------------------------------------

void position_cb(const geometry_msgs::PoseStamped msg){

    local_position = msg;
```

```
}
// ----------------------------------------------------Quota-----------------------------------------------------------

void range_leddar( const sensor_msgs::Range::ConstPtr &msg)
{

Quota=msg->range;;



ROS_INFO("altezza raggiunta=%f",Quota);
}









//---------------------------------------------- FUNZIONE DI TAKE OFF-------------------------------------------------------



void setTakeoff( float z_ref)
{

   x = local_position.pose.position.x;
   y = local_position.pose.position.y;
   z = Quota;
          e_x=0; // inserisco 0 perche evito che si sposti lungo x
          e_y=0; //inserisco 0 perche evito che si sposti lungo y
// errore differenza tra obiettivo e posizione attuale
  // e_x = x_ref - x;
```

```
    //e_y = y_ref - y;
    e_z = z_ref -z;
    ROS_INFO("x error is %f ", e_x);
    ROS_INFO("y error is %f ", e_y);
    ROS_INFO("y error is %f ", e_z);

}
  //---------------------------------------------- FUNZIONE LANDING----------------------------------------------------------
 void setLanding(float ax_ref, float ay_ref, float az_ref)
{

// errore differenza tra obiettivo e posizione attuale

    x = local_position.pose.position.x;
    y = local_position.pose.position.y;
    z = Quota;
    e_ax = ax_ref - x;
    e_ay = ay_ref - y;
    e_az = az_ref -z;
    ROS_INFO("x error is %f ", e_ax);
    ROS_INFO("y error is %f ", e_ay);
    ROS_INFO("y error is %f ", e_az);

}
   //---------------------------------------------- FUNZIONE DESTINATION----------------------------------------------------------
 void setDestination(float x_ref, float y_ref, float z_ref)
{
    x = local_position.pose.position.x;
    y = local_position.pose.position.y;
    z = Quota;
// errore differenza tra obiettivo e posizione attuale
```

```
    e_wx = x_ref - x;
    e_wy = y_ref - y;
    e_wz = z_ref -z;
            ROS_INFO("x error is %f ", e_wx);
    ROS_INFO("y error is %f ", e_wy);
    ROS_INFO("y error is %f ", e_wz);

}

//------------------------------------------------------------------------------

// MAIN:

//------------------------------------------------------------------------------


int main(int argc, char **argv)

{

    ros::init(argc, argv, "offb_node4");

    ros::NodeHandle nh;


    state_sub = nh.subscribe<mavros_msgs::State> ("mavros/state", 10, state_cb);

    pos_sub = nh.subscribe<geometry_msgs::PoseStamped> ("mavros/local_position/pose", 10,
position_cb);
```

```
        ros::Subscriber dist_sub = nh.subscribe<sensor_msgs::Range>("/mavros/rangefinder/rangefinder",
10, range_leddar);


   local_vel_pub = nh.advertise<geometry_msgs::Twist> ("mavros/setpoint_velocity/cmd_vel_unstamped",
10);


   ros::ServiceClient arming_client =
nh.serviceClient<mavros_msgs::CommandBool>("mavros/cmd/arming");


   ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>("mavros/set_mode");



 // the setpoint publishing rate MUST be faster than 2Hz


   ros::Rate rate(20.0);



   // wait for FCU connection


   while(ros::ok() && !current_state.connected)
        {


      ros::spinOnce();


      rate.sleep();


   }



        float z_partenza=Quota;


                altezza=0.5+z_partenza;
```

```
        setTakeoff(altezza);
        while(ros::ok()&& (e_z>0.1))
        {
                setTakeoff(altezza);
                ROS_INFO("take off");
                vel.linear.x = K*e_x;
                vel.linear.y = K*e_y;
                vel.linear.z = K*e_z;
                ROS_INFO("x error is %f ", e_x);
                ROS_INFO("y error is %f ", e_y);
                ROS_INFO("z error is %f ", e_z);
                ROS_INFO("altezza %f ", altezza);
                ROS_INFO("quota %f ", Quota);
           local_vel_pub.publish(vel);
                ros::spinOnce();
                rate.sleep();
         }


ROS_INFO("Attesa di 5 secondi");
        for(int i = 0; ros::ok() && i < 2*20; ++i)
        {
                 ros::spinOnce();
                 rate.sleep();
        }


        float x_partenza=local_position.pose.position.x;
        float y_partenza=local_position.pose.position.y;

         setLanding(x_partenza,y_partenza,0);
```

```
 while(ros::ok()&& (Quota>0.18))
{
        setLanding(x_partenza,y_partenza,0);
        ROS_INFO("landing");
        ROS_INFO("x local %f ", x_partenza);
        ROS_INFO("y local is %f ", y_partenza);
   vel.linear.x = K*e_ax;
        vel.linear.y = K*e_ay;
        vel.linear.z = K*e_az;
        local_vel_pub.publish(vel);
        ros::spinOnce();
        rate.sleep();
}

 //disarmo drone a fine atterraggio
        mavros_msgs::CommandBool arm_cmd;

        arm_cmd.request.value = false;
        if( arming_client.call(arm_cmd) && arm_cmd.response.success)
        {
                ROS_INFO("Vehicle disarmed");

        }

        return 0;

}
```