#include "dji\_sdk\_demo/demo\_flight\_control.h"

#include "dji\_sdk/dji\_sdk.h"

#include <time.h>

#include <iostream>

#include <fstream>

#include <stdlib.h>

#include <ctime>

const float deg2rad = C\_PI/180.0;

const float rad2deg = 180.0/C\_PI;

const int takeoff\_wait = 0; // sek

float search\_height = 15, pickup\_height = 5;

int somefile\_state = 0;

int Print\_counter = 0;

int search\_counter = 0, square\_counter = 1;

int flight\_state = 0, state = 1;

float xCmd, yCmd, zCmd, yaw;

float x, y, z;

bool found = false;

float square\_size = 5;

float homex = 0, homey = 0;

float dropx = 0, dropy = 1;

float s\_x = 100, s\_y = 100; // recure point

float targetx = 90;//camera\_cordinatex; //(subscribe to get this coordiante)

float targety = 70;//camera\_cordinatey; //(subscribe to get this coordiante)

float parameterx = targetx - 1;

float parametery = targety;

ros::ServiceClient set\_local\_pos\_reference;

ros::ServiceClient sdk\_ctrl\_authority\_service;

ros::ServiceClient drone\_task\_service;

ros::ServiceClient query\_version\_service;

ros::Publisher ctrlPosYawPub;

ros::Publisher ctrlBrakePub;

uint8\_t flight\_status = 255;

uint8\_t display\_mode = 255;

sensor\_msgs::NavSatFix current\_gps;

geometry\_msgs::Quaternion current\_atti;

geometry\_msgs::Point current\_local\_pos;

std::ofstream somefile;

Mission square\_mission;

void flight\_pub();

void flight\_plan();

void rand\_camera();

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

{

ros::init(argc, argv, "demo\_flight\_control\_node");

ros::NodeHandle nh;

ros::Subscriber attitudeSub = nh.subscribe("dji\_sdk/attitude", 10, &attitude\_callback);

ros::Subscriber gpsSub = nh.subscribe("dji\_sdk/gps\_position", 10, &gps\_callback);

ros::Subscriber flightStatusSub = nh.subscribe("dji\_sdk/flight\_status", 10, &flight\_status\_callback);

ros::Subscriber displayModeSub = nh.subscribe("dji\_sdk/display\_mode", 10, &display\_mode\_callback);

ros::Subscriber localPosition = nh.subscribe("dji\_sdk/local\_position", 10, &local\_position\_callback);

ROS\_INFO("Update");

ctrlPosYawPub = nh.advertise<sensor\_msgs::Joy>("dji\_sdk/flight\_control\_setpoint\_ENUposition\_yaw", 10);

ctrlBrakePub = nh.advertise<sensor\_msgs::Joy>("dji\_sdk/flight\_control\_setpoint\_generic", 10);

// Basic services

sdk\_ctrl\_authority\_service = nh.serviceClient<dji\_sdk::SDKControlAuthority> ("dji\_sdk/sdk\_control\_authority");

drone\_task\_service = nh.serviceClient<dji\_sdk::DroneTaskControl>("dji\_sdk/drone\_task\_control");

query\_version\_service = nh.serviceClient<dji\_sdk::QueryDroneVersion>("dji\_sdk/query\_drone\_version");

set\_local\_pos\_reference = nh.serviceClient<dji\_sdk::SetLocalPosRef> ("dji\_sdk/set\_local\_pos\_ref");

static ros::Time start\_time = ros::Time::now();

ros::Duration elapsed\_time = ros::Time::now() - start\_time;

start\_time = ros::Time::now();

bool obtain\_control\_result = obtain\_control();

bool takeoff\_result;

if (!set\_local\_position()) // We need this for height

{

ROS\_ERROR("GPS health insufficient - No local frame reference for height. Exiting.");

return 1;

}

if(is\_M100())

{

ROS\_INFO("takeoff in %d",takeoff\_wait);

ros::Duration(takeoff\_wait).sleep();

ROS\_INFO("Ready");

ROS\_INFO("M100 taking off!");

takeoff\_result = M100monitoredTakeoff();

}

if(takeoff\_result)

{

x = s\_x;

y = s\_y;

z = search\_height;

yaw = 0;

flight\_state = 2;

ROS\_INFO("flight state: %d",flight\_state);

}

ros::spin();

return 0;

}

bool

M100monitoredTakeoff()

{

ros::Time start\_time = ros::Time::now();

float home\_altitude = current\_gps.altitude;

if(!takeoff\_land(dji\_sdk::DroneTaskControl::Request::TASK\_TAKEOFF))

{

return false;

}

ros::Duration(0.01).sleep();

ros::spinOnce();

// Step 1: If M100 is not in the air after 10 seconds, fail.

while (ros::Time::now() - start\_time < ros::Duration(10))

{

ros::Duration(0.01).sleep();

ros::spinOnce();

}

if(flight\_status != DJISDK::M100FlightStatus::M100\_STATUS\_IN\_AIR ||

current\_gps.altitude - home\_altitude < 1.0)

{

ROS\_ERROR("Takeoff failed.");

return false;

}

else

{

start\_time = ros::Time::now();

ROS\_INFO("Successful takeoff!");

ros::spinOnce();

}

return true;

}

void attitude\_callback(const geometry\_msgs::QuaternionStamped::ConstPtr& msg)

{

current\_atti = msg->quaternion;

}

void local\_position\_callback(const geometry\_msgs::PointStamped::ConstPtr& msg)

{

current\_local\_pos = msg->point;

}

void gps\_callback(const sensor\_msgs::NavSatFix::ConstPtr& msg)

{

current\_gps = \*msg;

flight\_plan();

}

void flight\_status\_callback(const std\_msgs::UInt8::ConstPtr& msg)

{

flight\_status = msg->data;

}

void display\_mode\_callback(const std\_msgs::UInt8::ConstPtr& msg)

{

display\_mode = msg->data;

}

bool is\_M100()

{

dji\_sdk::QueryDroneVersion query;

query\_version\_service.call(query);

if(query.response.version == DJISDK::DroneFirmwareVersion::M100\_31)

{

return true;

}

return false;

}

bool obtain\_control()

{

dji\_sdk::SDKControlAuthority authority;

authority.request.control\_enable=1;

sdk\_ctrl\_authority\_service.call(authority);

if(!authority.response.result)

{

ROS\_ERROR("obtain control failed!");

return false;

}

return true;

}

bool takeoff\_land(int task)

{

dji\_sdk::DroneTaskControl droneTaskControl;

droneTaskControl.request.task = task;

drone\_task\_service.call(droneTaskControl);

if(!droneTaskControl.response.result)

{

ROS\_ERROR("takeoff\_land fail");

return false;

}

return true;

}

bool set\_local\_position()

{

dji\_sdk::SetLocalPosRef localPosReferenceSetter;

set\_local\_pos\_reference.call(localPosReferenceSetter);

return localPosReferenceSetter.response.result;

}

void

localOffsetFromGpsOffset(geometry\_msgs::Vector3& deltaNed,

sensor\_msgs::NavSatFix& target,

sensor\_msgs::NavSatFix& origin)

{

double deltaLon = target.longitude - origin.longitude;

double deltaLat = target.latitude - origin.latitude;

deltaNed.y = deltaLat \* deg2rad \* C\_EARTH;

deltaNed.x = deltaLon \* deg2rad \* C\_EARTH \* cos(deg2rad\*target.latitude);

deltaNed.z = target.altitude - origin.altitude;

}

geometry\_msgs::Vector3 toEulerAngle(geometry\_msgs::Quaternion quat)

{

geometry\_msgs::Vector3 ans;

tf::Matrix3x3 R\_FLU2ENU(tf::Quaternion(quat.x, quat.y, quat.z, quat.w));

R\_FLU2ENU.getRPY(ans.x, ans.y, ans.z);

return ans;

}

void flight\_pub()

{

if (Print\_counter == 50){

if(somefile\_state == 0){

somefile.open ("somefile.txt");

somefile\_state = 1;}

somefile << "current local pos x "; somefile << current\_local\_pos.x; somefile << " and latitude "; somefile << current\_gps.latitude; somefile << "\n";

somefile << "current local pos y "; somefile << current\_local\_pos.y; somefile << " and longitude "; somefile << current\_gps.longitude; somefile << "\n";

somefile << "current local pos z "; somefile << current\_local\_pos.z; somefile << " and altitude "; somefile << current\_gps.altitude; somefile << "\n \n";

if(somefile\_state == 2){

somefile.close();}

ROS\_INFO("x %f and current local pos x %f and latitude %f",x,current\_local\_pos.x,current\_gps.latitude);

ROS\_INFO("y %f and current local pos y %f and longitude %f",y,current\_local\_pos.y,current\_gps.longitude);

ROS\_INFO("z %f and current local pos z %f and altitude %f",z,current\_local\_pos.z,current\_gps.altitude);

ROS\_INFO("yaw: %f xCmd: %f yCmd: %f zCmd: %f",yaw,xCmd,yCmd,zCmd);

Print\_counter = 0;

}

sensor\_msgs::Joy controlPosYaw;

controlPosYaw.axes.push\_back(xCmd);

controlPosYaw.axes.push\_back(yCmd);

controlPosYaw.axes.push\_back(zCmd);

controlPosYaw.axes.push\_back(yaw);

ctrlPosYawPub.publish(controlPosYaw);

}

void flight\_plan(){

if ( current\_local\_pos.x > 0 && current\_local\_pos.y > 0 ){

if ( abs(current\_local\_pos.x - targetx) < 5 && abs(current\_local\_pos.y - targety) < 5 ){

found = true;

ROS\_INFO("True");

}}

switch(flight\_state){

case 0:

if (square\_counter != 1){

somefile\_state = 2;

Print\_counter = 50;

flight\_pub();

ROS\_INFO("all done");

}

break;

case 1:

if (found){

rand\_camera();

/\*

flight\_state = 3;

ROS\_INFO("flight state: %d",flight\_state);

\*/

}else{

if(square\_counter<10){

z = search\_height;

switch(search\_counter){

case 0:

x = s\_x + square\_size\*square\_counter;

if (square\_counter == 1)

{

y = s\_y;

}else{

y = s\_y - square\_size\*(square\_counter-1);

}

yaw = 0;

search\_counter++;

flight\_state = 2;

ROS\_INFO("flight state: %d",flight\_state);

break;

case 1:

x = s\_x + square\_size\*square\_counter;

y = s\_y + square\_size\*square\_counter;

yaw = 90;

search\_counter++;

flight\_state = 2;

ROS\_INFO("flight state: %d",flight\_state);

break;

case 2:

x = s\_x - square\_size\*square\_counter;

y = s\_y + square\_size\*square\_counter;

yaw = 180;

search\_counter++;

flight\_state = 2;

ROS\_INFO("flight state: %d",flight\_state);

break;

case 3:

x = s\_x - square\_size\*square\_counter;

y = s\_y - square\_size\*square\_counter;

yaw = 270;

search\_counter = 0;

square\_counter++;

flight\_state = 2;

ROS\_INFO("counter: %d",square\_counter);

ROS\_INFO("flight state: %d",flight\_state);

break;

}

}else{

flight\_state = 0;

}

}

break;

case 2:

xCmd = x-current\_local\_pos.x;

yCmd = y-current\_local\_pos.y;

zCmd = z;

if (abs(xCmd) > 0.5 || abs(yCmd) > 0.5 || abs(zCmd-current\_local\_pos.z) > 0.5 ){

if (found){

rand\_camera();

//flight\_state = 3;

//ROS\_INFO("flight state: %d",flight\_state);

}else{

Print\_counter++;

flight\_pub();

}

}else{

ros::Duration(2).sleep();

flight\_state = 1;

ROS\_INFO("flight state: %d",flight\_state);

}

break;

case 3:

switch (state){

case 1: //Fly closer (1 meter from target)

x = parameterx;

y = parametery;

xCmd = x - current\_local\_pos.x;

yCmd = y - current\_local\_pos.y;

zCmd = search\_height;

ROS\_INFO("fly 1");

if (abs(xCmd) > 0.5 || abs(yCmd) > 0.5 || abs(zCmd-current\_local\_pos.z) > 0.5 ){

flight\_pub();

}else{

ros::Duration(2).sleep();

state = 2;

ROS\_INFO("state: %d",state);

}

break;

case 2:

x = parameterx;

y = parametery;

xCmd = x - current\_local\_pos.x;

yCmd = y - current\_local\_pos.y;

zCmd = pickup\_height;

ROS\_INFO("fly 2");

if (abs(xCmd) > 0.5 || abs(yCmd) > 0.5 || abs(zCmd-current\_local\_pos.z) > 0.5 ){

flight\_pub();

ROS\_INFO("Lowering net pickup");

ROS\_INFO("xCmd: %f yCmd: %f zCmd: %f",xCmd,yCmd,zCmd);

}else{

ros::Duration(2).sleep();

state = 3;

ROS\_INFO("state: %d",state);

}

break;

case 3:

x = targetx;

y = targety;

xCmd = x - current\_local\_pos.x;

yCmd = y - current\_local\_pos.y;

zCmd = pickup\_height;

ROS\_INFO("fly 3");

if (abs(xCmd) > 0.5 || abs(yCmd) > 0.5 || abs(zCmd-current\_local\_pos.z) > 0.5 ){

flight\_pub();

}else{

ros::Duration(2).sleep();

state = 4;

ROS\_INFO("state: %d",state);

}

break;

case 4:

x = targetx;

y = targety;

xCmd = x - current\_local\_pos.x;

yCmd = y - current\_local\_pos.y;

zCmd = search\_height;

ROS\_INFO("fly 4");

if (abs(xCmd) > 0.5 || abs(yCmd) > 0.5 || abs(zCmd-current\_local\_pos.z) > 0.5 ){

ROS\_INFO("Reasing net pickup");

flight\_pub();

}else{

ros::Duration(2).sleep();

flight\_state = 4;

state = 1;

ROS\_INFO("state: %d",state);

}

break;

}

break;

case 4:

x = dropx;

y = dropy;

xCmd = x - current\_local\_pos.x;

yCmd = y - current\_local\_pos.y;

zCmd = search\_height;

if (abs(xCmd) > 0.5 || abs(yCmd) > 0.5 || abs(zCmd-current\_local\_pos.z) > 0.5 ){

flight\_pub();

}else{

ros::Duration(2).sleep();

flight\_state = 5;

ROS\_INFO("state: %d",state);

}

break;

case 5:

x = dropx;

y = dropy;

xCmd = x - current\_local\_pos.x;

yCmd = y - current\_local\_pos.y;

zCmd = 2;

if (abs(xCmd) > 0.5 || abs(yCmd) > 0.5 || abs(zCmd-current\_local\_pos.z) > 0.5 ){

flight\_pub();

}else{

ROS\_INFO("Lowering net");

ros::Duration(2).sleep();

ROS\_INFO("Release net");

ros::Duration(2).sleep();

flight\_state = 6;

ROS\_INFO("state: %d",state);

}

break;

case 6:

x = dropx;

y = dropy;

xCmd = x - current\_local\_pos.x;

yCmd = y - current\_local\_pos.y;

zCmd = search\_height;

if (abs(xCmd) > 0.5 || abs(yCmd) > 0.5 || abs(zCmd-current\_local\_pos.z) > 0.5 ){

flight\_pub();

}else{

ros::Duration(2).sleep();

flight\_state = 7;

ROS\_INFO("state: %d",state);

}

break;

case 7:

x = homex;

y = homey;

xCmd = x - current\_local\_pos.x;

yCmd = y - current\_local\_pos.y;

zCmd = search\_height;

if (abs(xCmd) > 0.5 || abs(yCmd) > 0.5 || abs(zCmd-current\_local\_pos.z) > 0.5 ){

flight\_pub();

}else{

//land();

ros::Duration(2).sleep();

flight\_state = 0;

ROS\_INFO("state: %d",state);

}

break;

break;

}

}

void rand\_camera(){

//bool rand\_found = false;

int i=0;

srand (time(NULL));

while(i<10){

int rand\_nr;

rand\_nr = rand() % 4;

ROS\_INFO("random number %d",rand\_nr);

switch(rand\_nr){

case 0 :

ROS\_INFO("v");

xCmd = current\_local\_pos.x;

yCmd = current\_local\_pos.y;

zCmd = search\_height;

yaw = yaw - 5;

flight\_pub();

ROS\_INFO("yaw: %f", yaw);

break;

case 1 :

ROS\_INFO("o");

break;

case 2 :

ROS\_INFO("h");

xCmd = current\_local\_pos.x;

yCmd = current\_local\_pos.y;

zCmd = search\_height;

yaw = yaw + 5;

ROS\_INFO("yaw: %f", yaw);

break;

case 3 :

ROS\_INFO("n");

break;

}

i++;

}

flight\_state = 3;

ROS\_INFO("flight state: %d",flight\_state);

}