#include "drone\_control\_fb\_slam/drone\_bebop\_control.h"

namespace drone\_bebop\_control

{

droneBebopControl::droneBebopControl(

const ros::NodeHandle& nh,

const ros::NodeHandle& nh\_private)

: nh\_(nh),

nh\_private\_(nh\_private),

name\_(nh\_private.getNamespace())

{

//ROS\_INFO("[%s]: Initializing droneBebopControl Node", name\_.c\_str());

ROS\_INFO("Initializing droneBebopControl Node");

loadParameters();

pose\_camera\_sub\_ = nh\_.subscribe("rs\_frame", 1, &droneBebopControl::poseCameraCallback, this);

state\_change\_sub\_ = nh\_.subscribe("state\_change", 1, &droneBebopControl::stateChangeCallback, this);

path\_change\_sub\_ = nh\_.subscribe("path\_change", 1, &droneBebopControl::pathChangeCallback, this);

desired\_velocity\_pub\_ = nh\_.advertise < geometry\_msgs::Twist > ("velocity", 1);

land\_pub\_ = nh\_.advertise < std\_msgs::Empty > ("/bebop/land", 1);

take\_off\_pub\_ = nh\_.advertise < std\_msgs::Empty > ("/bebop/takeoff", 1);

//steering\_angle\_ = 0.0;

//probability\_of\_collision\_ = 0.0;

// Aggressive initialization

desired\_forward\_velocity\_ = 0.0;

desired\_side\_velocity\_ = 0.0;

use\_slam\_ = false;

fly\_path\_ = false;

fly\_path\_state\_ = 1.0; // Initial position

count = 0;

round = 0;

pos\_command\_x = 0.0;

pos\_command\_y = 0.0;

prev\_err\_x\_ = 0.0;

prev\_err\_y\_ = 0.0;

f\_D\_err\_x\_ = 0.0;

f\_D\_err\_y\_ = 0.0;

no\_slam\_alert\_ = 1;

no\_slam\_counter\_ = 0;

}

void droneBebopControl::run()

{

ros::Duration(2.0).sleep();

ros::Rate rate(30.0);

while (ros::ok())

{

// Desired body frame velocity to world frame

//double desired\_forward\_velocity\_m = (1.0 - probability\_of\_collision\_) \* max\_forward\_index\_;

//if (desired\_forward\_velocity\_m <= 0.0)

//{

//ROS\_INFO("Detected negative forward velocity! Drone will now stop!");

//desired\_forward\_velocity\_m = 0;

//}

double Kp\_pos\_x = 0.1;

double Kd\_pos\_x = 0.5;

double Kp\_pos\_y = 0.1;

double Kd\_pos\_y = 0.5;

double Ki\_pos\_y = 0.000;

double D\_err\_x, err\_x;

double D\_err\_y, err\_y;

// filtered P+D navigation: //scale\_true\_

err\_x = (pos\_command\_x - (position\_x - position\_x\_0)\*1);

D\_err\_x = err\_x - prev\_err\_x\_;

f\_D\_err\_x\_ = 0.5648\*f\_D\_err\_x\_ + 12.75\*D\_err\_x;

cmd\_velocity\_.linear.x = Kp\_pos\_x\*err\_x + Kd\_pos\_x\*f\_D\_err\_x\_;

ROS\_INFO("ErrX: %.3f - F\_DErrX: %.3f ",err\_x, f\_D\_err\_x\_);

if (cmd\_velocity\_.linear.x >= 0.5)

cmd\_velocity\_.linear.x = 0.5;

if (cmd\_velocity\_.linear.x<= -0.5)

cmd\_velocity\_.linear.x = -0.5;

err\_y = (pos\_command\_y - (position\_y - position\_y\_0)\*1);

D\_err\_y = err\_y - prev\_err\_y\_;

f\_D\_err\_y\_ = 0.5648\*f\_D\_err\_y\_ + 12.75\*D\_err\_y;

cmd\_velocity\_.linear.y = Kp\_pos\_y\*err\_y + Kd\_pos\_y\*f\_D\_err\_y\_ ;

if (cmd\_velocity\_.linear.y >= 0.5)

cmd\_velocity\_.linear.y = 0.5;

if (cmd\_velocity\_.linear.y <= -0.5)

cmd\_velocity\_.linear.y = -0.5;

prev\_err\_x\_ = err\_x;

prev\_err\_y\_ = err\_y;

// Verify that slam is still working..

no\_slam\_counter\_ = (no\_slam\_counter\_ > 10) ? no\_slam\_counter\_ : no\_slam\_counter\_+1;

if (no\_slam\_counter\_ > 10)

{

ROS\_INFO("lost SLAM tracking, holding place.");

no\_slam\_alert\_ = 1;

}

// Publish desired state if slam is still running

if (use\_slam\_ && !no\_slam\_alert\_) //After we start and no\_slam\_counter\_<10 (we didnt lost it).

{

//cmd\_velocity\_.angular.z = 0.1;

desired\_velocity\_pub\_.publish(cmd\_velocity\_);

ROS\_INFO("PUBLISHING VELOCITY");

// ROS\_INFO("VEl: %.3f",cmd\_velocity\_);

}

else if (use\_slam\_ && no\_slam\_alert\_) //After we start and no\_slam\_counter\_>10 (we didnt lost it).

{

cmd\_velocity\_.linear.x = 0.0;

cmd\_velocity\_.linear.y = 0.0;

cmd\_velocity\_.linear.z = 0.0;

cmd\_velocity\_.angular.z = 0.0;

desired\_velocity\_pub\_.publish(cmd\_velocity\_);

ROS\_INFO("publishing verlocity with problem");

}

else //At starting ()

{

cmd\_velocity\_.linear.x = 0.0; //desired\_forward\_velocity\_;

cmd\_velocity\_.linear.y = 0.0;

cmd\_velocity\_.linear.z = 0.0;

cmd\_velocity\_.angular.z = 0.0;

desired\_velocity\_pub\_.publish(cmd\_velocity\_);

ROS\_INFO("NOT PUBLISHING VELOCITY");

position\_x\_0 = position\_x;

position\_y\_0 = position\_y;

}

//ROS\_INFO("POSX: %.3f - POSY: %.3f - POSZ: %5.3f",

//position\_x - position\_x\_0, position\_y - position\_y\_0, position\_z);

ROS\_INFO("POSX: %.3f - POSY: %.3f - POSZ: %5.3f",err\_x, err\_y, position\_z);

if ((fly\_path\_) && (round<2))

{

bool is\_inplace = false;

//ROS\_INFO("test1");

switch (fly\_path\_state\_)

{

case 1: // forward

pos\_command\_x = 1.0;

pos\_command\_y = 0.0; //0.5

if (((err\_y <= 0.15) && (err\_y >= -0.15)) && ((err\_x <= 0.15) && (err\_x >= -0.15)))

{

is\_inplace = true;

count = count + 1;

if (count > 100)

{

ROS\_INFO("DONE 10 seconds of mode LEFT BACK.");

fly\_path\_state\_ = 2;

count = 0;

}

}

break;

case 2: // backword

pos\_command\_x = 0.0;

pos\_command\_y = 0.0; //0.5

if (((err\_y <= 0.15) && (err\_y >= -0.15)) && ((err\_x <= 0.15) && (err\_x >= -0.15)))

{

is\_inplace = true;

count = count + 1;

if (count > 100)

{

ROS\_INFO("DONE 10 seconds of mode LEFT FRONT.");

fly\_path\_state\_ = 1;

count = 0;

}

}

break;

case 3: // right front

pos\_command\_x = 0.8;

pos\_command\_y = 0.0;

if (((err\_y <= 0.15) && (err\_y >= -0.15)) && ((err\_x <= 0.15) && (err\_x >= -0.15)))

{

is\_inplace = true;

count = count + 1;

if (count > 100)

{

ROS\_INFO("DONE 10 seconds of mode RIGHT FRONT.");

ROS\_INFO("DONE 10 seconds of mode RIGHT FRONT.");

ROS\_INFO("DONE 10 seconds of mode RIGHT FRONT.");

fly\_path\_state\_ = 4;

count = 0;

}

}

ROS\_INFO("mode RIGHT FRONT : %s", is\_inplace ? "IN PLACE" : "ON THE WAY");

break;

case 4: // right back

pos\_command\_x = 0.0;

pos\_command\_y = 0.0;

if (((err\_y <= 0.15) && (err\_y >= -0.15)) && ((err\_x <= 0.15) && (err\_x >= -0.15)))

{

is\_inplace = true;

count = count + 1;

if (count > 100)

{

ROS\_INFO("DONE 10 seconds of mode RIGHT BACK.");

ROS\_INFO("DONE 10 seconds of mode RIGHT BACK.");

ROS\_INFO("DONE 10 seconds of mode RIGHT BACK.");

fly\_path\_state\_ = 1;

count = 0;

//round++;

}

}

ROS\_INFO("mode RIGHT BACK : %s", is\_inplace ? "IN PLACE" : "ON THE WAY");

break;

}

}

if (round==0)

{

std\_msgs::Empty takeoff\_empty;

take\_off\_pub\_.publish(takeoff\_empty);

round = round + 1;

}

if (round==2)

{

std\_msgs::Empty land\_empty;

use\_slam\_ = 0;

land\_pub\_.publish(land\_empty);

}

rate.sleep();

ros::spinOnce();

}

}

void droneBebopControl::poseCameraCallback(const geometry\_msgs::PoseStamped::ConstPtr& msg)

{

//probability\_of\_collision\_ = msg->collision\_prob;

//steering\_angle\_ = msg->steering\_angle;

position\_x = msg->pose.position.x;

position\_y = msg->pose.position.y;

position\_z = msg->pose.position.z;

//position\_z = msg.pose.pose.position.z;

no\_slam\_counter\_ = 0;

no\_slam\_alert\_ = 0;

}

void droneBebopControl::stateChangeCallback(const std\_msgs::Bool& msg)

{

//change current state

use\_slam\_ = msg.data;

ROS\_INFO("change state");

ROS\_INFO("VEl: %.3f",cmd\_velocity\_);

prev\_err\_x\_ = 0;

prev\_err\_y\_ = 0;

f\_D\_err\_x\_ = 0;

f\_D\_err\_y\_ = 0;

}

void droneBebopControl::pathChangeCallback(const std\_msgs::Bool& msg)

{

//change current state

fly\_path\_ = msg.data;

ROS\_INFO("fly path");

}

void droneBebopControl::loadParameters()

{

ROS\_INFO("Reading parameters");

//ROS\_INFO("[%s]: Reading parameters", name\_.c\_str());

//nh\_private\_.param<double>("alpha\_velocity", alpha\_velocity\_, 0.3);

//nh\_private\_.param<double>("scale\_true", scale\_true\_, 1.0);

//nh\_private\_.param<double>("max\_forward\_index", max\_forward\_index\_, 0.2);

//nh\_private\_.param<double>("critical\_prob", critical\_prob\_coll\_, 0.7);

}

} // namespace drone\_bebop\_control

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

{

ros::init(argc, argv, "drone\_bebop\_control\_");

drone\_bebop\_control::droneBebopControl dn;

dn.run();

return 0;

}