Trent Ziemer

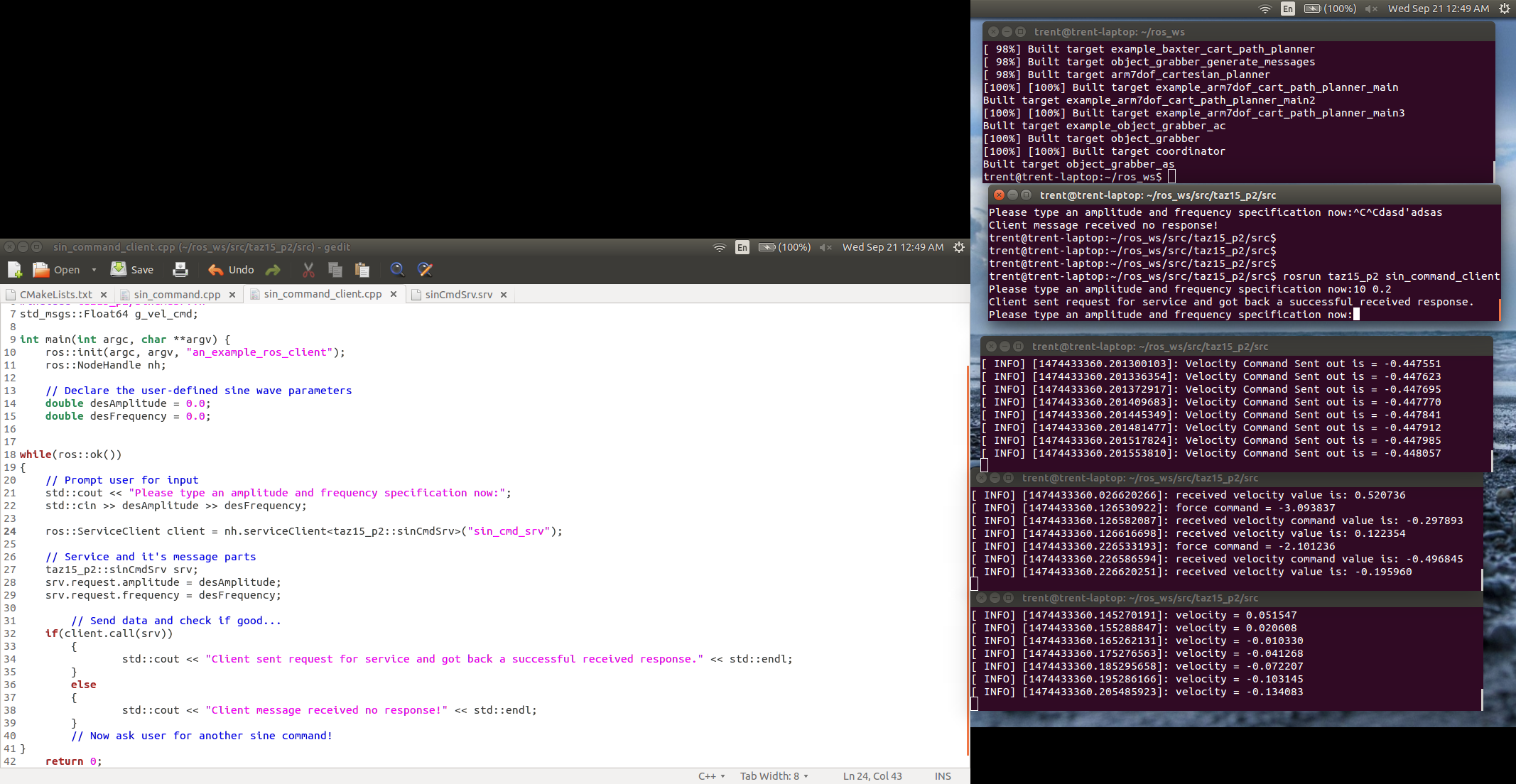
Dr. Newman

EECS 397

September 20, 2016

Homework 2: Sine Controller Services

On the left screen is some of my source code. On the right we see, from top to bottom: my catkin\_make terminal, the user inputting a sine command into the sin\_command\_client (which is then sent out to sin\_command), the sin\_command node sending out vel\_cmd data based on the client data through a client/service interaction, and then minimal controller node terminal, and the minimal simulator node terminal. The user can repeatedly send out new commands to the sin\_command node which sends out the most up-to-date commanded sine wave data to the controller.



Here is a copy of my source code for the server node:

// SIN COMMANDER NODE MODIFIED TO BE A SERVER FOR HW2 - TRENT ZIEMER 9/20/2016

#include<ros/ros.h>

#include<std\_msgs/Float64.h>

#include<taz15\_p2/sinCmdSrv.h>

//global variables for callback functions to populate for use in main program

std\_msgs::Float64 g\_vel\_cmd;

// Global pointers so that the callback and int main() can both access the publisher

ros::NodeHandle \* g\_nh;

ros::Publisher \* g\_pub;

double g\_client\_command\_amplitude;

double g\_client\_command\_frequency;

bool callback(taz15\_p2::sinCmdSrvRequest& request, taz15\_p2::sinCmdSrvResponse& response)

{

ROS\_INFO("Server received a client request command.");

// Set global variables here based on received request data

g\_client\_command\_amplitude = request.amplitude;

g\_client\_command\_frequency = request.frequency;

// Junk ---->

response.received = true;

return true;

}

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

ros::init(argc, argv, "sin\_command");

ros::NodeHandle nh;

// Declare variables, with global pointers for use by callbacks

g\_nh = &nh;

ros::Publisher pub = g\_nh->advertise<std\_msgs::Float64>("vel\_cmd", 1);

g\_pub = &pub;

ros::start();

ros::ServiceServer service = nh.advertiseService("sin\_cmd\_srv", callback);

// Continually update vel\_cmd, spinning once to potentially update from callback action

while(ros::ok())

{

double vel\_cmd = g\_client\_command\_amplitude\*sin(g\_client\_command\_frequency\*ros::Time::now().toSec());

// Assign vel command to publishable object

g\_vel\_cmd.data = vel\_cmd;

// Now publish

g\_pub->publish(g\_vel\_cmd);

// Log info for user to cout

ROS\_INFO("Velocity Command Sent out is = %f", g\_vel\_cmd.data);

ros::spinOnce();

}

return 0; // Should never get here, unless roscore dies!

}

And here is a copy of my source code for the client node:

// SIN COMMANDER CLIENT NODE - TRENT ZIEMER 9/20/2016

#include<ros/ros.h>

#include<std\_msgs/Float64.h>

#include<taz15\_p2/sinCmdSrv.h>

std\_msgs::Float64 g\_vel\_cmd;

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

ros::init(argc, argv, "an\_example\_ros\_client");

ros::NodeHandle nh;

// Declare the user-defined sine wave parameters

double desAmplitude = 0.0;

double desFrequency = 0.0;

while(ros::ok())

{

// Prompt user for input

std::cout << "Please type an amplitude and frequency specification now:";

std::cin >> desAmplitude >> desFrequency;

ros::ServiceClient client = nh.serviceClient<taz15\_p2::sinCmdSrv>("sin\_cmd\_srv");

// Service and it's message parts

taz15\_p2::sinCmdSrv srv;

srv.request.amplitude = desAmplitude;

srv.request.frequency = desFrequency;

// Send data and check if good...

if(client.call(srv))

{

std::cout << "Client sent request for service and got back a successful received response." << std::endl;

}

else

{

std::cout << "Client message received no response!" << std::endl;

}

// Now ask user for another sine command!

}

return 0;

}