Lab 4
Zeta Group:

Justin Evaristo
Aditya Malik
Ben Poreh
Andrew Wood

The purpose of this lab was to develop and test software that would have a robot travel along a path sent over the network, pausing when an obstacle moved into its path, and resuming execution when the obstacle was removed using ROS. This package was to provide an action server that would accept connections with arbitrary clients, accept paths from these clients, compute the control commands necessary to move the robot from its current position (initial position assumed to be (0, 0) and (0, 0, 0, 1)) heading to the next position along the given path. In addition, our system listened to LIDAR scans, determined if an obstacle was in the path of the robot, and published an alarm if so. The effect an alarm had upon the execution of the path is as follows: we assumed all obstacles are "temporary," or in other words they were not static, and would move given enough time. Therefore, the robot does not need to compute a sub-path to get around the obstacle, it just needs to pause execution and wait until the obstacle moved out of the path. Our code was designed with several components in mind.

1) We created a ROS node that listened to LIDAR scans, and detemined if an obstacle was present, and if present, was in the path of the robot. This code was implemented in ps2, and lab2.

2) We created a PathExecutor class whose responsibility was to accept a path (list of pose objects), compute the commands necessary to "blindly" (no feedback) follow that path, and then to execute them. This code was implemented in ps3 and lab3. This code was modified to account for the LIDAR alarm to implement the "pausing" behavior.

3) We created a ROS action server whose responsibility was to instantiate, clean up, and forward path requests to the PathExecutor class as well as provide callbacks.

4) We created a ROS action client whose responsibility was to create a simple path and send it over the network.

For component 2, we had to change the "move" algorithm as follows (the "move" algorithm was responsible for accepting a command and making the robot move for the specified amount in the command):

```
procedure move(double dt, double total_time, geometry_msgs::Twist& cmd, bool override)
        returns double:
    double elapsed_time = 0.0;
    geometry_msgs::Twist zero_cmd;
    while(elapsed_time < total_time AND ros::ok()) do:

        // POINT A
        ros::spinOnce();        // lets lidar alarm callback execute if necessary
        while(NOT override AND lidar_alarm_present() AND ros::ok()) do:
            publish(zero_cmd);

            // POINT B
            ros::spinOnce();        // let lidar alarm callback execute again...otherwise
                                    // endless loop
        publish(cmd);
        elapsed_time += dt;
        sleep(dt);
    return elapsed_time;
```

There are two points in this algorithm (Point A and Point B) that are worth mentioning, and without both, the algorithm will not only be incorrect, but will hang. They are important for the following reasons:

Point A: The algorithm is designed to pause execution of the path when an obstacle is present. Therefore, we need to let the action server respond to the LIDAR alarm callback. However, since ROS is single threaded by default, we need to call ros::spinOnce() to allow the server to execute a single callback that is present, which would be the LIDAR alarm in our implementation. Without this line, the robot would not pause when an obstacle is present.

Point B: When the inner while loop is present (an obstacle is detected), we are publishing no movement commands. For this loop to ever end, ROS needs to execute a callback again to allow the server to be informed that the obstacle has moved. Without this line, the inner while loop would never end once an obstacle is detected, regardless if the obstacle moves or not.

We ran our code on a turtlebot, and had it move in a simple “L” shape. To view our code running, here is a link to a youtube video of it: https://m.youtube.com/watch?v=GFEk9SfJC5M&feature=youtu.be and to view our source code, here is pointer to our github repo: https://github.com/aew61/eecs376

One last thing we should mention about our algorithm above, is the “override” variable. It is used to “ignore” obstacle commands, and we used it to prevent robots from pausing execution when turning. Obviously when the robot is moving forward, if there is an obstacle in the way, it should stop, but if the robot is spinning, then there is no need to pause execution.