EECS376 PS6

This assignment was all about learning how to create more sophisticated (and realistic) physics into our assumptions about robotic movement. We used to assume that when commands were issued, they were carried out instantaneously, which is not only a false assumption for real world dynamics, but can potentially be a dangerous assumption for the longevity of motors and servos on robots. We learned the dynamics appropriate for computing movement in which the velocity commands are linearly increased until reaching max velocity (at a constant acceleration), both in angular and translational cases. Using this knowledge, we were to design code to that, given code that computes acceleration and deceleration for well defined trajectories, would cause an arbitrary robot to brake given a LIDAR alarm (code from the past assignments). The algorithm I came up with is as follows:

```
procedure break_trajectory(stamped_pose current_pose, vector<odometry_msg> vec_of_states,
                                integer current_pos_in_vec) returns nothing:

    // if this case occurs, then there is no trajectory being executed so the robot must be
    // at rest...keep it that way if we are actually intending to brake
    if(no elements in vec_of_states OR current_pos_in_vec >= length(vec_of_states)) then:
        vec_of_states.add(empty odometry msg)
    end if

    // get the current velocities that the robot is moving at...want to ramp these down
    double current_vel = vec_of_states[current_pos_in_vec].twist.twist.linear.x;
    double current_omega = vec_of_states[current_pos_in_vec].twist.twist.angular.z;

    // abandon old trajectory
    clear(vec_of_states);

    // compute max distance the robot will travel for each velocity
    double ramp_down_phi_des = 0.5 * (current_omega ^ 2) / max_alpha;
    double ramp_down_linear_des = 0.5 * (current_vel ^ 2) / max_alpha;

    // compute time necessary to come to a stop..since we are ramping both down
    // simultaneously, this should be the max of the time to do each individually
    double t_ramp = max(sqrt(ramp_down_linear_dist/a_max), sqrt(ramp_down_phi_des/a_max));

    // compute the number of iterations this will take given time step dt
    double npts_ramp = round(t_ramp / dt);

    // compute rates at which we slow down
    double d_omega = alpha_max * dt, d_vel = alpha_max * dt;

    //start off desired velocities to be how fast the robot is moving now
    double omega_des = current_omega, speed_des = current_vel;

    // need to account for rotation direction when slowing down angular velocity
    double omega_accel = sign(current_omega) * a_max;
```

```
// start breaking
double t = 0.0;
double new_x = 0.0, new_y = 0.0;
for integer index = 0; index < npts_ramp; ++index do:

        // need to cap velocity at minimum of 0.0
        speed_des -= (speed_des > 0.0) ? a_max * dt : 0.0;

        // compute new x and y coordinates for odometry pose
        new_x += speed_des * dt * cos(ramp_down_phi_des);
        new_y += speed_des * dt * sin(ramp_down_phi_des);

        // cannot let omega_des change sign
        if sign(current_omega) == -1.0 then:
                omega_des -= (omega_des < 0.0) ? omega_accel * dt : 0.0;
        else:
                omega_des -= (omega_des > 0.0) ? omega_accel * dt : 0.0;
        end if

        // update angle the robot is at
        ramp_down_phi_des += omega_des * dt;

        // append the new odometry state to the list
        vec_of_states.add(convert_params_to_odom(new_x, new_y, speed_des,
                                                 omega_des, ramp_down_phi_des));

end for

// at the end of this, assume we made it and publish an at rest state
vec_of_states.add(create_rest_odom());
```

What was interesting in this assignment was that I never got a chance to test this algorithm out. I can prove to you that it works using the dynamics of motion as well as the steps taken in the algorithm, but I do not have a video showcasing its performance. I am not sure why, but when I ran the mobot launch files, my LIDAR would not work (even with the updated launch file). The LIDAR scans would not return anything OTHER than an array full of “-inf” elements, which not only doesn’t make sense (a negative LIDAR value??), but would not change even when I paused gazebo, moved the mobot next to a wall and then resumed the simulation. Anyways, I did find an error in my lidar_alarm code; I did not anticipate getting negative LIDAR scan values, so my lidar_alarm was always True until I fixed the issue.