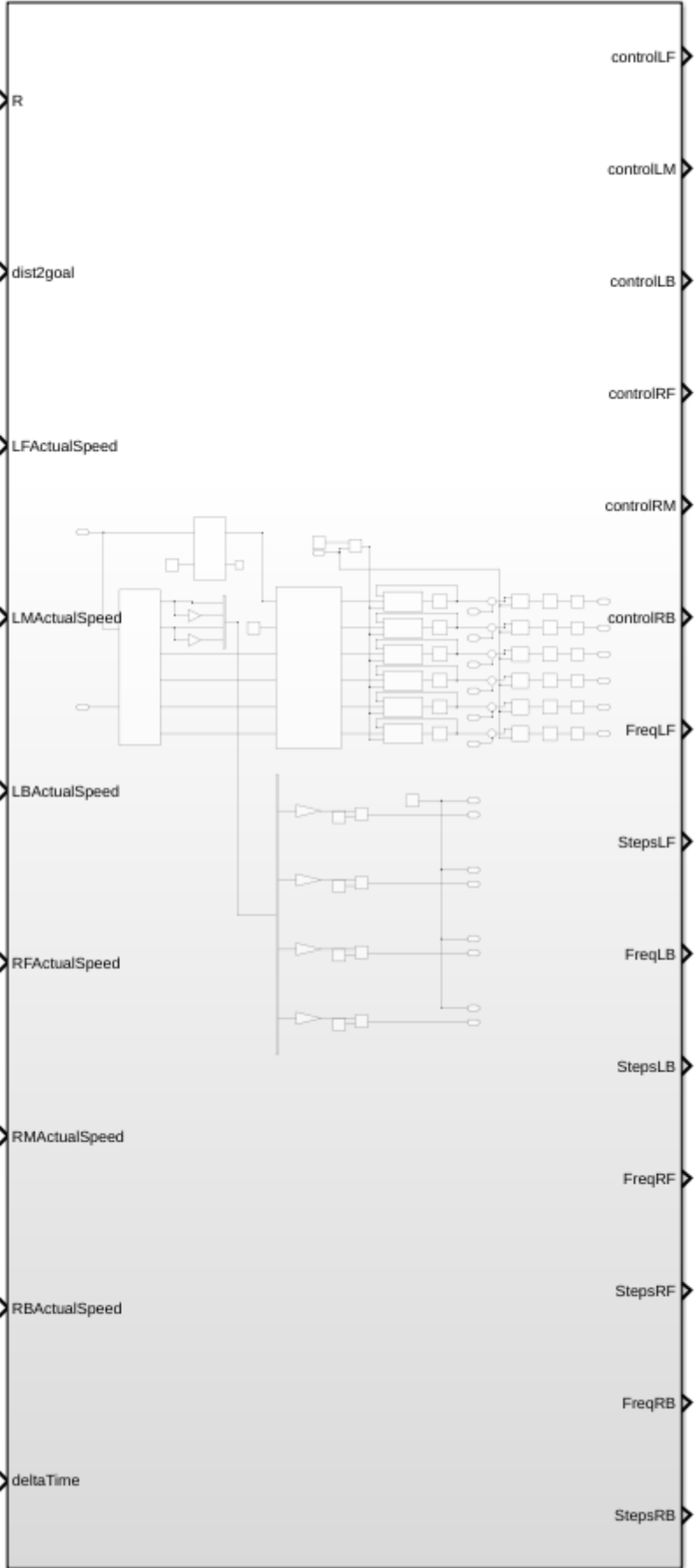


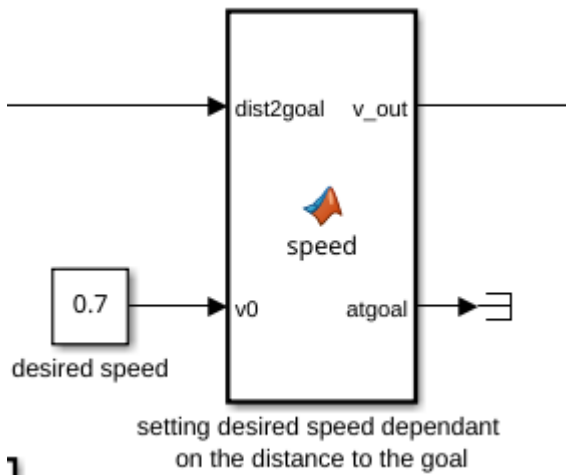
Automatic control

Overview

In general, the job of the control system is to turn high level instructions from the software system, and turn them into instructions for the hardware. This is done by having a distance to a goal, a turning radius, and a `deltaTime` as an input, and turning those into control signals for the motors:



control drive



The function simply works like this:

```
function [v_out, atgoal] = speed(dist2goal, v0)

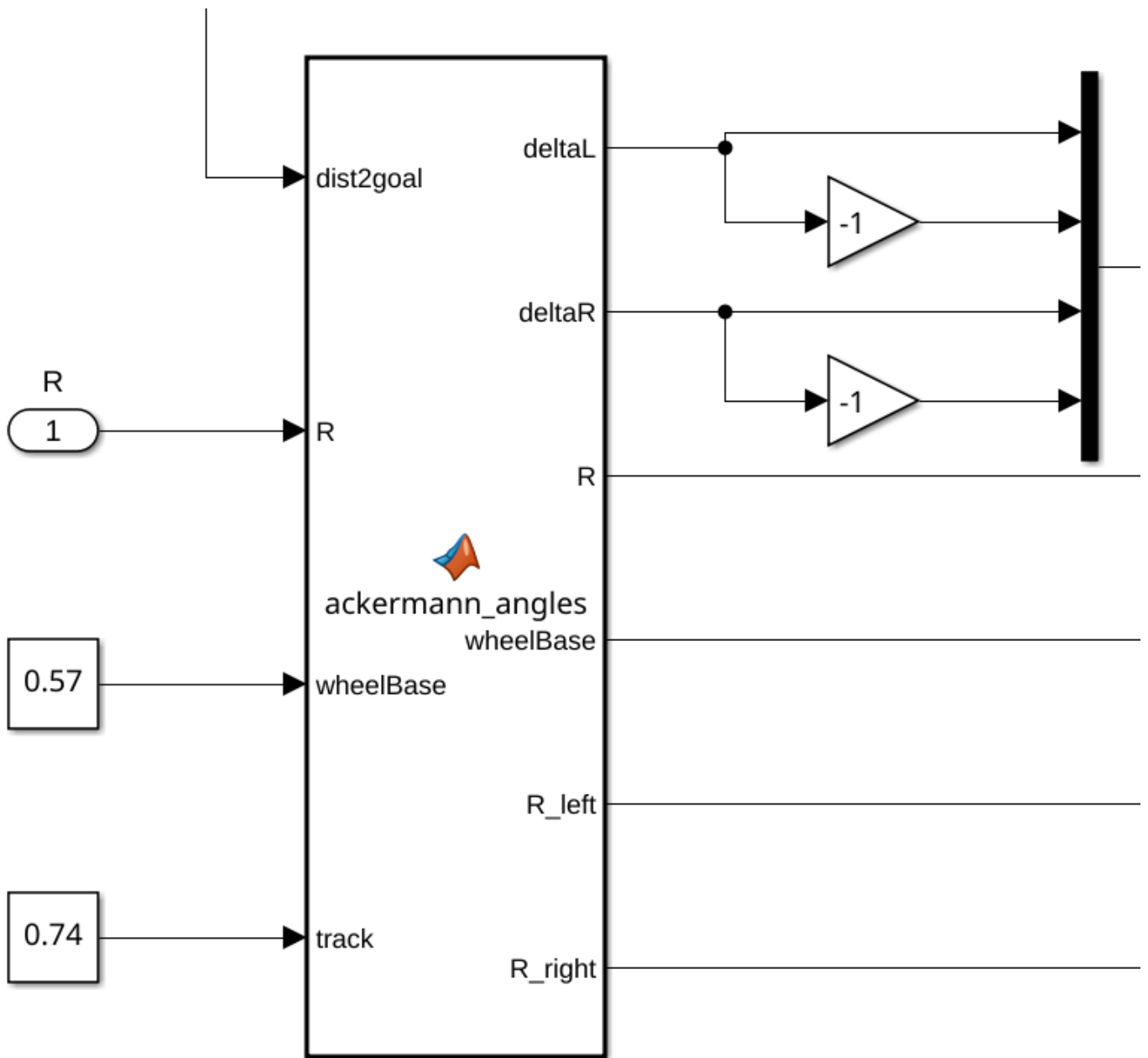
%slow down if close to goal
if dist2goal <= 1
    v_out = 0;
    atgoal = 1;

%stopping if the rover is close to the target
elseif dist2goal <= 2
    v_out = v0 * 0.5;
    atgoal = 0;

%keeping cruising speed if not close to goal
else
    v_out = v0;
    atgoal = 0;
end
```

Ackermann steering

Getting the necessary angles that the steering motors need to make is done by calculating the Ackermann steering angles that equate from the given turning radius and distance to the goal.



getting the steering angles and turning radii

deltaL and deltaR are equal to the angles that the left and right steering motors need to make, which are sent to the steering control.

This function also outputs the turning radius of the right and left wheels of the rover.

The function works like this:

```
function [deltaL,deltaR,R,wheelBase,R_left,R_right] =
ackermann_angles(dist2goal,R,wheelBase,track)
    % Function to compute ackerman steering angles from a bicycle steer angle.
    if R == 0
        alpha = 0;
    else
```

```

        alpha = dist2goal/R;
    end

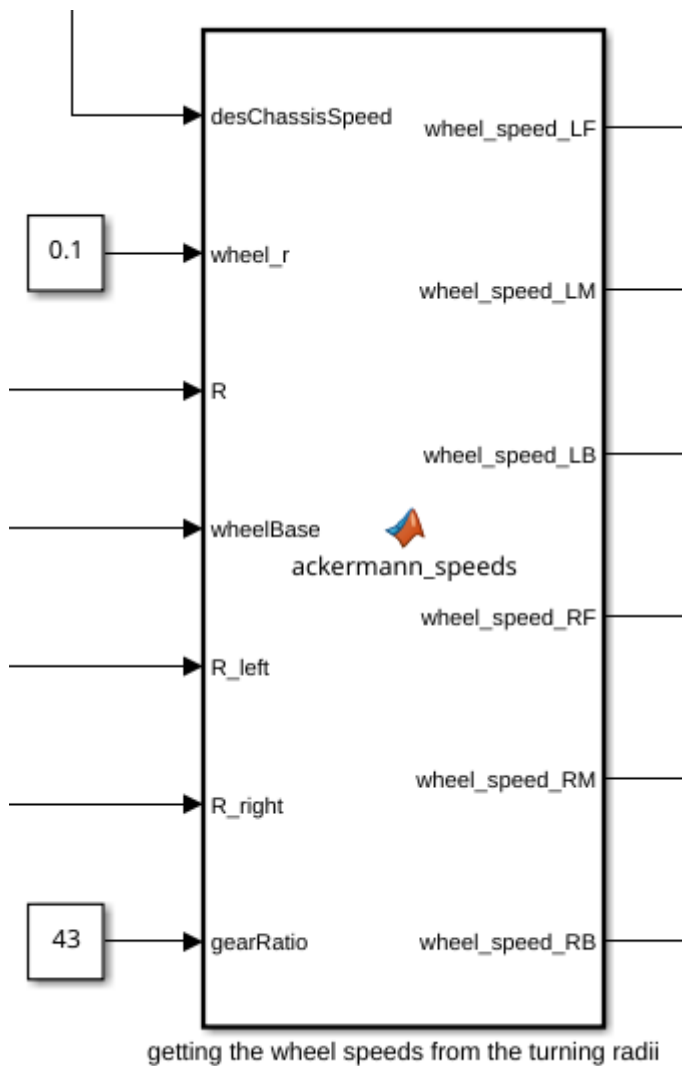
    R = abs(R);

    if alpha < 0
        R_left = (R+track/2);
        R_right = (R-track/2);
        deltaL = atan(wheelBase/(R_left));
        deltaR = atan(wheelBase/(R_right));
    elseif alpha > 0
        R_left = (R-track/2);
        R_right = (R+track/2);
        deltaL = -atan(wheelBase/(R_left));
        deltaR = -atan(wheelBase/(R_right));
    else
        deltaL = 0;
        deltaR = 0;
        R_left = R;
        R_right = R;
    end
end
end

```

Ackermann speeds

Using the data coming from the previous function, as well as a wheel radius and motor gear ratio, the speeds needed by the motors are calculated.



Internally the function works like this:

```
function
[wheel_speed_LF, wheel_speed_LM, wheel_speed_LB, wheel_speed_RF, wheel_speed_RM, wheel_speed_RB] =
...
    ackermann_speeds(desChassisSpeed, wheel_r, R, wheelBase, R_left, R_right, gearRatio)
if R > 0
    wheel_speed_LF = (desChassisSpeed/wheel_r)*sqrt(R_left^2 + (wheelBase/2)^2)/R;
    wheel_speed_LM = desChassisSpeed * R_left/(R*wheel_r);
    wheel_speed_LB = (desChassisSpeed/wheel_r)*sqrt(R_left^2 + (wheelBase/2)^2)/R;
    wheel_speed_RF = (desChassisSpeed/wheel_r)*sqrt(R_right^2 + (wheelBase/2)^2)/R;
    wheel_speed_RM = desChassisSpeed * R_right/(R*wheel_r);
    wheel_speed_RB = (desChassisSpeed/wheel_r)*sqrt(R_right^2 + (wheelBase/2)^2)/R;
else
    wheel_speed_LF = desChassisSpeed / wheel_r;
    wheel_speed_LM = desChassisSpeed / wheel_r;
    wheel_speed_LB = desChassisSpeed / wheel_r;
```

```

wheel_speed_RF = desChassisSpeed / wheel_r;
wheel_speed_RM = desChassisSpeed / wheel_r;
wheel_speed_RB = desChassisSpeed / wheel_r;
end

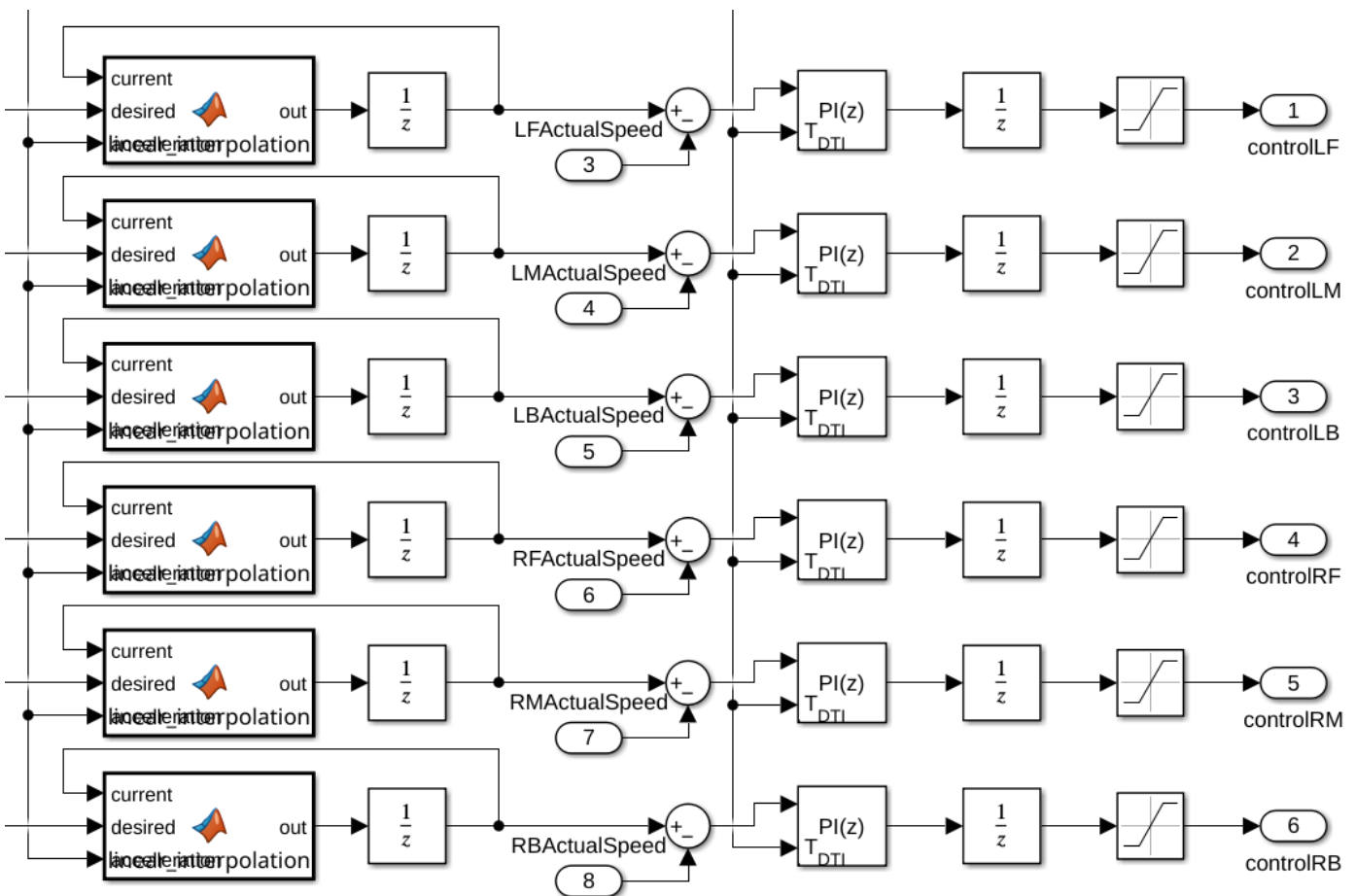
wheel_speed_LF = wheel_speed_LF * gearRatio;
wheel_speed_LM = wheel_speed_LM * gearRatio;
wheel_speed_LB = wheel_speed_LB * gearRatio;
wheel_speed_RF = wheel_speed_RF * gearRatio;
wheel_speed_RM = wheel_speed_RM * gearRatio;
wheel_speed_RB = wheel_speed_RB * gearRatio;
end

```

Control signals

DC motors

After getting all the angles and speeds needed for the rover to make a certain movement, the signals get sent to the motor drivers/controllers in a way that they can use them.



The DC motors are controlled by first sending the desired speed into a linear interpolation function block, which just makes the desired speed change like a linear function, instead of as a stepping function gradually changing the speed to limit the amount of amps being pulled at startup.

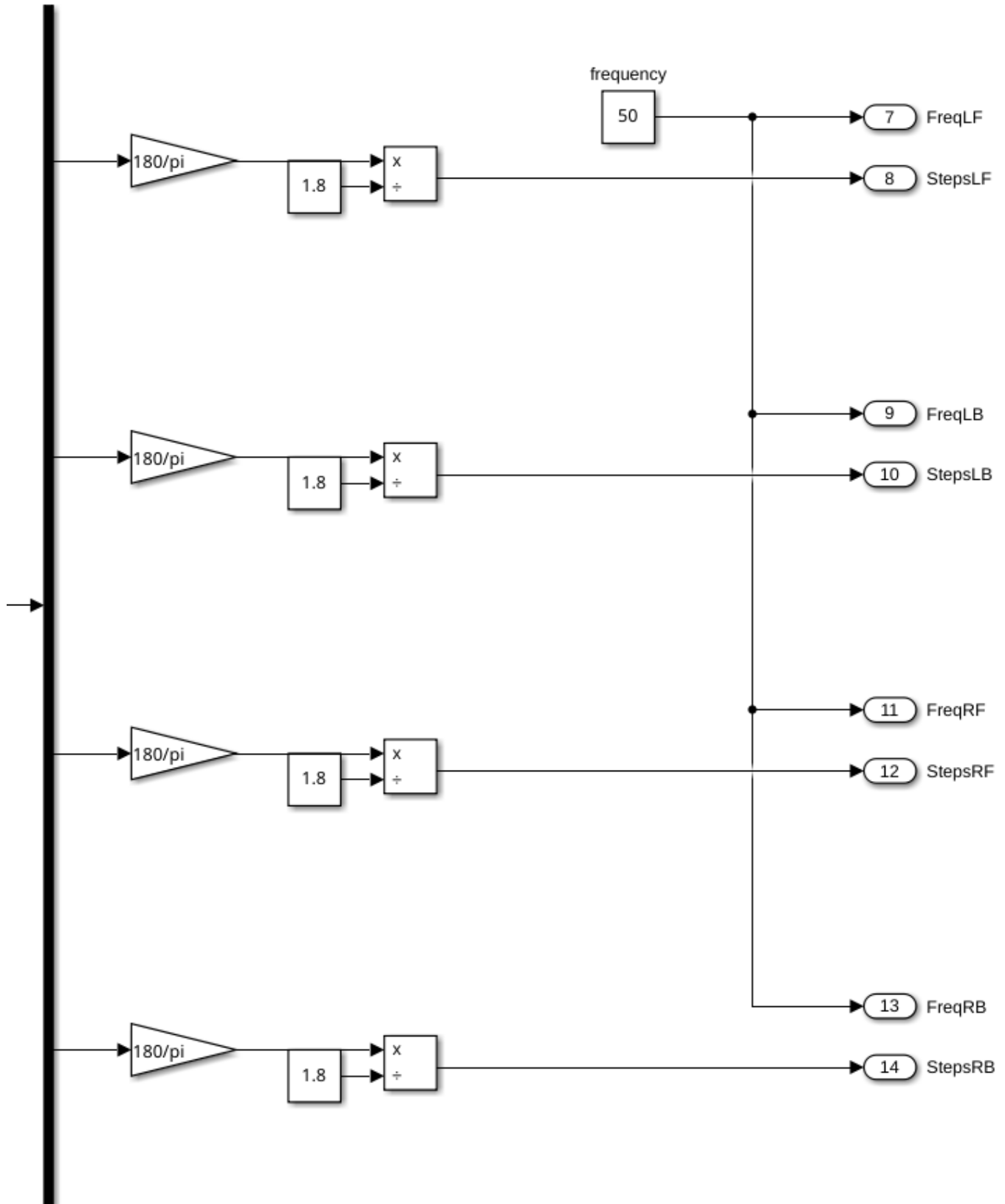
```
function out = linear_interpolation(current, desired, acceleration)
if current < desired
    current = current + acceleration;
end
if current > desired
    current = current - acceleration;
end

out = current;
```

After this, the linearly changing desired speed will be compared with the actual speed of the motors and the final speed will be controlled using PI controllers which are saturated to stay between -5V and 5V.

Stepper motors

Due to the limitations of Simulink, making the stepper motors move is done using PWM signals coming from the STM32 microcontrollers. The control system only gives the desired position of the stepper motors in terms of steps, and a frequency at which the PWM signal pulses, which in this case is just a constant value.



The sending of the PWM signals is handled by the embedded team.

Revision #6

Created 2026-04-14 08:46:22 UTC by Rowan Ali

Updated 2026-04-16 14:04:39 UTC by Rowan Ali