1 Acceleration Protocols

Four acceleration protocols are currently supported by DeepNNcar and executed by the client. This adds a level of security to ensure constant connectivity between the RPi and the client. These protocols can be selected by setting the appropriate boolean flag *True* in Controller.py. All protocols are subject to the following constraint.

$$ACC \in [ACC_0 - R, ACC_0 + F] \tag{1}$$

$$|error| < threshold \tag{2}$$

Bounds for IR frequency

$$\lambda \in [\frac{1}{T_{zd}}, \frac{1}{T_{bt}}] \tag{3}$$

$$speed = \lambda \times \pi \times d \tag{4}$$

$$r \in [0, r_{max}] \tag{5}$$

Where ACC_0 is the duty cycle when DeepNNCar is idle and R and F are respectively the user-specified reverse and forward range for the acceleration duty cycle.

1.1 Default acceleration protocol

By default, the user can control the speed indirectly by manipulating the duty cycle of the PWM signal. To adjust the speed, the user presses the right or left trigger on the xbox controller.

$$ACC = ACC_0 + RT * F + LT * R \tag{6}$$

Where RT is the right trigger signal $\in [0,1]$ and LT is the left trigger signal $\in [0,1]$.

1.2 Constant duty cycle acceleration protocol

This protocol is simple to implement and useful when collecting data on a track with a consistent surface when a constant speed, not necessarily a specific speed in m/s, is required. Such a mode can be used to determine the relationship between duty cycle and true speed in m/s (fig. 4).

$$ACC = ACC_0 + F \tag{7}$$

1.3 Steering-based constant duty cycle acceleration protocol

This protocol is based off the realization that when turning, it is natural to slow down a vehicle. This mode is useful when collecting data on a uniform track with tight turns. This mode will capture more data on the turns than the constant duty cycle acceleration protocol alone.

$$ACC = ACC_0 + F - C * |S - S_0| \tag{8}$$

Where C is a constant, S is the current steering value $\in [10,20]$, and S_0 is the idle duty cycle for acceleration.

1.4 CRUISE CONTROL ACCELERATION PROTOCOL

The proportional-integral-derivative (PID) controller is often used in control systems to regulate a continuously modulated control using a feedback loop. We use a PID controller to regulate the speed of DeepNNCar and to emulate the cruise control functionality present on many cars. To modify the speed of the car, we calculate an error value $e(t)$ as:

$$e(t) = r(t) - y(t) \tag{9}$$

where $r(t)$ and $y(t)$ are the set speed and measured speed respectively at time t in m/s. We measure the current speed of DeepNNCar as described previously by using the IR coupler to calculate RPM.

Using this error signal, we calculate a control signal $u(t)$

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{10}$$

Where K_p (proportional gain), K_i (integral gain), and K_d (derivative gain) are non-negative hyper parameters used to control the shape and response of the feedback loop. Using a trial-and-error method, the control coefficients were selected to be 0.013, 0.0001, 0.0002 respectively.

Once calculated, the control signal $u(t)$ can then be used to update the acceleration duty cycle with the following equation

$$ACC(t+1) = ACC(t) + u(t) \tag{11}$$

Cruise control has several useful applications. For our purposes, it was used in data collection and to test the limits of the autonomous driving accuracy in a controller manner.

Using "UP" and "DOWN" on the directional pad, the user can change the reference speed. This protocol is restricted to only working in the forward direction.