Some pointers for the testbed implementation

Implementation of the Collision Avoidance task:

Goal : To compute 'throttle' and 'brake' parameters

Implementation for Collision Avoidance involves two important steps. They are

1. Step 1: Apply emergency brakes if distance between vehicles less than threshold value (say 10m)
2. Step 2: Provide emergency warning if distance between vehicles is in a specific range (say less than 40m but greater than 10m)

The value of 'brake' and 'throttle' parameter can vary in the range [0,1]. If distance between vehicles less than 40m but greater than 10m you can issue an emergency warning (blinking LEDs) with the following code

```
gioSetPort(hetPORT1, gioGetPort(hetPORT1) ^ 0x80000021);
```

Implementation of Steer Task:

Goal: To compute 'steer' in the range -0.5<= 0 <=0.5

To compute steer value, we need to first compute the error. The value of error can be based on three parameters

Parameter 1 - heading error (or its derivative) of the follower vehicle (headingError)

Parameter 2 - lateral error (or its derivative) of the follower vehicle (lateralError)

Parameter 3 - angular velocity (or its derivative) of the follower vehicle (angular_vel)

Variables involved in the computation of the steer value:

steer, angular_vel, headingError, lateralError

The value of 'steer' depends on the error value. If error is negative, steer should be positive. Similarly if error is positive, steer should be negative. Also, while computing error, consider the possibility of having different weights for each of the three parameters. Optimal weights can be arrived after testing with different combinations.

Implementation of Cruise Control Task:

Goal: To compute brake, throttle

Brake and throttle values are computed to achieve three purposes:

1. P1: to maintain a specific distance between the follower and the lead vehicle.
2. P2: to maintain a specific velocity if the lead vehicle is far apart from the follower vehicle.
3. P3: to detect the curve ahead and maintain the permissible velocity of the follower vehicle while manoeuvring through a curve.

We are trying to compute brake based on the acceleration demand (accelDemand) of the follower vehicle. The way in which acceleration demand is computed depends on the purpose.

Generally accelDemand is based on the distance between lead and the follower vehicle, velocity and acceleration of both vehicles. In case of intelligent speed adaptation for curves, acceleration demand can be computed based on the difference between current velocity (fol_vel) and curveVelocity.

1. A naïve algorithm with no vehicle ahead → accelDemand = Function(constant velocity)
2. A naïve algorithm with vehicle ahead → accelDemand = Function(inter vehicular distance)
3. Improved algorithm (with vehicle ahead) → accelDemand = Function(inter vehicular distance, velocity, acceleration)
4. Improved algorithm + Intelligent Speed adaptation (with vehicle ahead) → accelDemand = Function(inter.vehi.distance, velocity, acceleration, curveVelocity, Distance to the curve)

There are 4 curves in the track. For each curve, hard coded values denote distance between track's starting point and the curve. P3 can be achieved by knowing permissible curve velocity (curveVelocity), the current distance between the follower vehicle (foltravelledDistance) and track's starting point.

Variables involved in the computation of the brake, throttle value:

distApartRadar, fol_vel, lead_vel, accelDemand, fol_acc_x, vehicleAhead, foltravelledDistance, curvePassedIndex, curveAheadIndex, curveCurrentIndex, maintainCurveSpeed, curveVelocity