OSS Erlang 開発演習用教材

ブレーキシミュレータ

－　機能仕様書　－

改版履歴

版数	日時			変更内容
0.1	2010.04.19	暫定	賀川	初版
0.2	2010.05.06	暫定	賀川	レビュー後の改版
0.3	2010.05.20	暫定	賀川	Output モジュール追加
1.0	2010.05.31	正式	賀川	Ver.1.0 初版ﾘﾘｰｽにあわせて発行
1.1	2010.06.03	正式	賀川	transmission 追加、図 1.1 変更、タイヤ円周を DB 化
2.0	2010.10.29	正式	賀川	入力レンジを 0..100000 へ拡張。路面環境を GUI で変更可能にし、摩擦係数を入力可能にする。空気抵抗の投影面積を変更可能にする。シミュレート結果のデータを保存可能にする。前方の車をシミュレート可能にする。前方距離センサー車間化
2.2	2010.11.18	正式	賀川	誤記修正・計算式修正

１．システム概要

１．１　基本アーキテクチャ

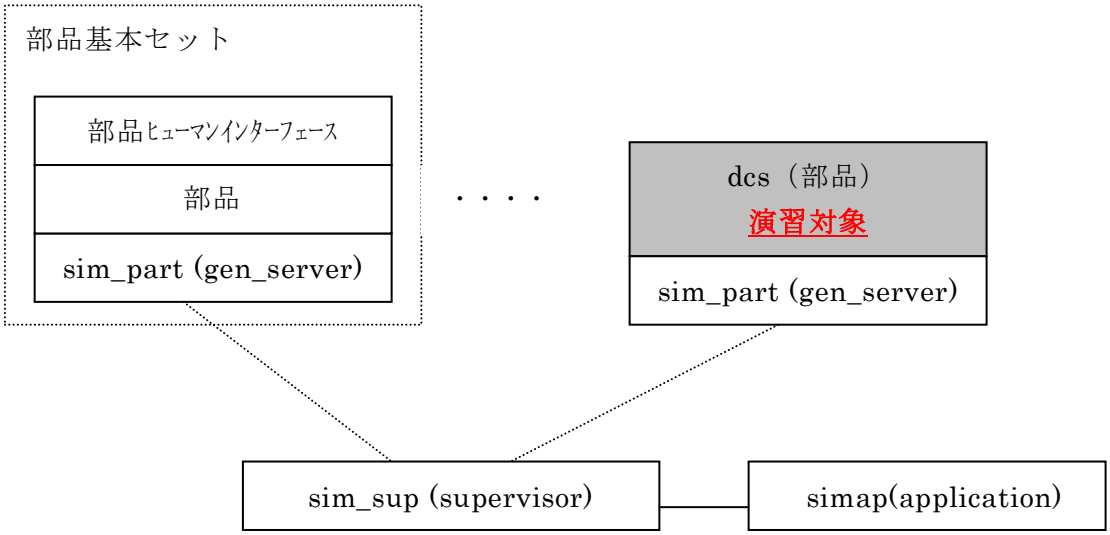


図１.1　アプリケーションモジュール関係図

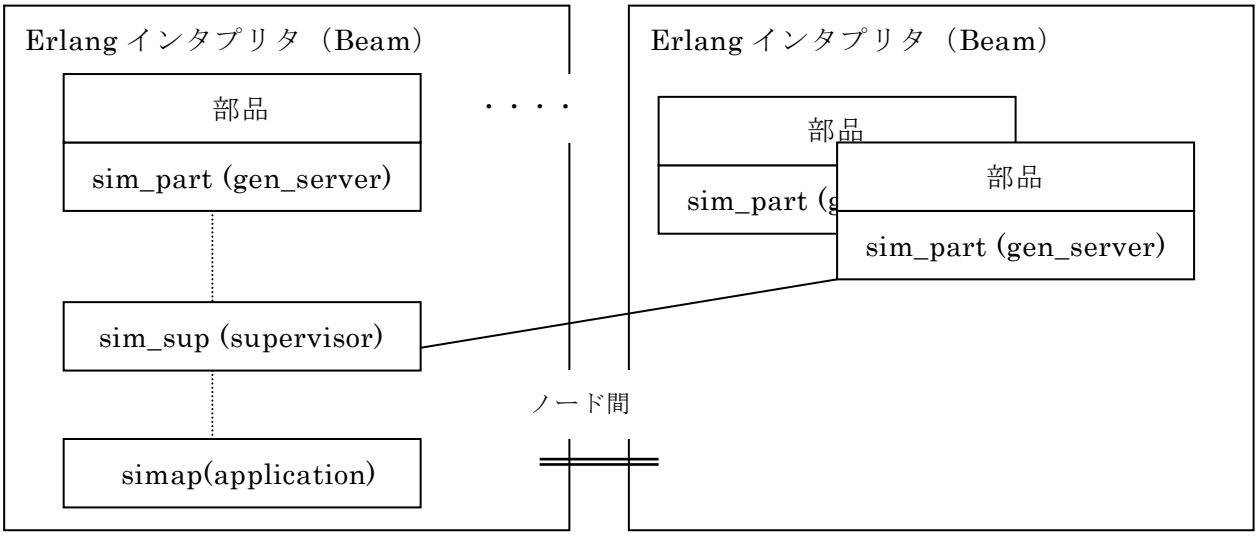


図１.2　実行イメージ

１．２　部品間通信方法

使用する OTP ライブラリ：gen_server,関数:cast
を使用し、メッセージ通信を行う。

記述例

```
gen_server:cast(宛先, {メッセージフォーマットは 3 章参照})
```

・第一引数に宛先を指定し、値は gen_server 名となる。
　3 章インターフェース仕様書のアドレス一覧を参照のこと。
・第二引数はメッセージを指定する。
　※メッセージフォーマットは、3 章インターフェース仕様書を参照のこと。

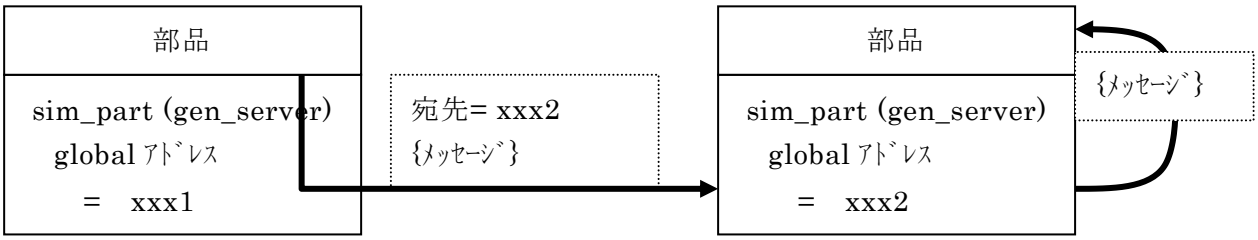


図１.2　部品間通信イメージ

【送信側の特徴】

Erlnag OTP の gen_server を利用し、それぞれの部品にグローバル（Erlang Beam 分散ネットワーク内でユニークとなる）を付与することで、アドレス管理の自作は不要となる。

メッセージの宛先は、目的に応じて３章インターフェースで取り決めた、アドレス一覧を指定する。

メッセージ内の発アドレスは、自部品のアドレスを３章インターフェースで取り決めた値で指定する。

記述例

```
gen_server:cast({global,wheelFright},{state_inf, brake,[10]}),
```

【受信側の特徴】

sim_part (gen_server)で受信した、メッセージを部品プロセスへ中継する。

部品プロセスにおいて、パターンマッチを利用することで、メッセージの宛先により処理を分けることを容易にする。

記述例

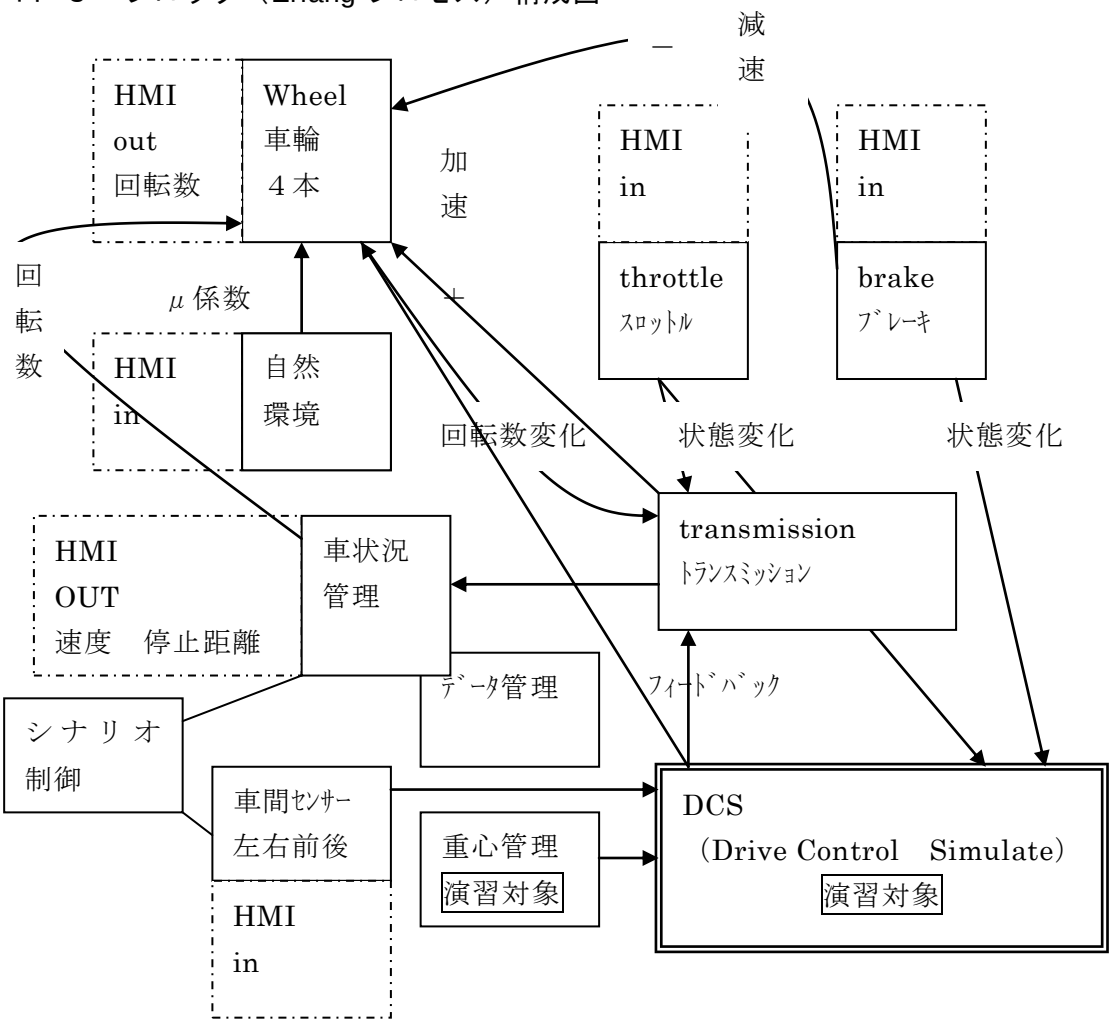
receive

```
{state_inf, brake, Parameter} ->
{state_inf, _From, Parameter} ->
```

ブレーキからの受信を意識した例

送信者を意識しない例

１．３　ブロック（Erlang プロセス）構成図



・演習対象 DCS（Drive Control　Simulate）に対して、各部品からの状態変化を通知する。
・DCS は、各情報を基に計算を行い、フィードバック情報指示をする。

図１.3　部品間ブロック図

２．モジュール詳細

Erlang モジュールとプロセス一覧を表 2-1 に示す。2.1 項以降で各モジュールの仕様を示す。

表２-１　Erlang モジュールとプロセス一覧

	behaviour	ﾓｼﾞｭｰﾙ名	プロセス名	概要
1	application	sim_app	brakesim	シミュレーションアプリケーション層
2	supervisor	sim_sup	sim_sup	シミュレーションプロセス監視
3	gen_server	sim_part	sim_part	部品プロセス管理サーバ
4		sim_db	sim_db	ｼﾐｭﾚｰﾀﾃﾞｰﾀﾍﾞｰｽ
5		wheel	wheelFright	前右　車輪回転数管理
6		wheel_gui	wheelFrigh_gui	前右　車輪回転数表示
		wheel	wheelFleft	前左　車輪回転数管理
		wheel_gui	wheelFleft_gui	前左　車輪回転数表示
		wheel	wheelRright	後右　車輪回転数管理
		wheel_gui	wheelRright_gui	後右　車輪回転数表示
		wheel	wheelRleft	後左　車輪回転数管理
		wheel_gui	wheelRleft_gui	後左　車輪回転数表示
7		brake	brake	ブレーキ状態管理
8		brake_gui	brake_gui	ブレーキ入力 HMI
9		throttle	throttle	スロットル（アクセル）
10		throttle_gui	throttle_gui	スロットル入力 HMI
11		distance	frsensor	前方センサー
12		distance_gui	frsensor_gui	前方センサー入力 HMI
		distance	bksensor	後方センサー
		distance_gui	bksensor_gui	後方センサー入力 HMI
		distance	risensor	右センサー
		distance_gui	risensor_gui	右センサー入力 HMI
		distance	lfsensor	左センサー
		distance_gui	lfsensor_gui	左センサー入力 HMI
13		loadenv	loadenv	路面環境
19		loadenv	loadenv_gui	路面環境 HMI
14		output	output	車状況
15		output	output_gui	車状況出力 HMI
16	受講生作成	balancectl.	balancectl.	重心制御
17	受講生作成	dcs	dcs	Drive Control Simulate
18		transmission	transmission	トランスミッション

20		scenario	scenario	シナリオ、ログ制御
21		scenario_gui	scenario_gui	シナリオ、ログ制御 HMI
22		calets		計算モジュール

２．１　シミュレーションアプリケーション層

Erlang　OTP の application として動作させる。

２．２　シミュレーションプロセス監視

Erlang　OTP の supervisor として動作させる。

２．３　部品プロセス管理サーバ

Erlang OTP の gen_server として動作させ、部品プロセスを生成する機能を有する。部品プロセス毎に、部品プロセス管理サーバを具備させることで、メッセージ通信が容易となる。また、supervisor のワーカーとして動かすことで安定運用も可能となる。

２．４　シミュレータデータベース

Erlang に内蔵されている、分散データベース Mnesia を利用して各種データを管理する。

【管理データ】

外部環境系データ
動作履歴
計算に使う定数値

【動作方法】

①　外部環境系データ

ErlangBeam 上から次のコマンドを投入

■摩擦係数

>sim_db:add_loavenv_item(ポジション,静摩擦,動摩擦,dummy).

ポジション　：　wheelFright（車輪前右）
　　wheelFleft　（車輪前左）
　　wheelRright（車輪後右）
　　wheelRright（車輪後右）
摩擦係数　　：　任意の数字　初期値　0.71

dummy　　：　現状未使用

■空気抵抗面積

>sim_db: update_areRegist (1.9).

空気抵抗面積　：　0～数値で入力　初期値　1.9　㎡

② 動作履歴

ErlangBeam 上から次のコマンドを投入

> start_scenario_sav(シナリオ名)　←　収集開始

シナリオ名　：　タプルで任意の名前を入力　例　‘012’

> stop_scenario_sav ()　←　収集停止

> exe_scenario (“シナリオ名”)　←　再生

シナリオ名　：文字列で入力　例”0123x”　start_scenario_sav で指定した名前＋数字が自動的に付加されている（同一名衝突防止）。

③ 計算に使う定数値の変更

ErlangBeam 上から次のコマンドを投入

(node1@localhost)16> sim_db:add_Caldat(tyre,2).

{atomic,ok}

＜引数＞　tyre：タイヤ外周

数値（例　2）

２．５　車輪回転数管理仕様

車輪の回転数を管理するプロセスである。便宜上、全てトルクで管理する。

－参考－

・トルク＝N・m

・N（ニュートン）＝　kg･m/s^2

・加速度＝m/s^2

・馬力＝トルク×rpm×0.001396

・タイヤ回転数＝rpm(1 分間の回転数)

・路面摩擦（u 係数）

－前提　車データ－

・タイヤ円周　＝　2m

・throttle 値＝１N とする

・最高速度　＝　300 km/h
・タイヤ最高 rpm　＝　2500 rpm
・最大 throttle 値　＝　100000
・車重量　＝　1000 kg
・垂直抗力　=　9800 N

【プロセス辞書　管理データ】

名前	意味	値の範囲
state	状態	0~
remaintorqu	処理 throttle 値	–－100000～100000
nowtorqu	現 throttle 値	0～100000
rpm	タイヤ回転数（rpm）	0～2500

【状態】

値	名前	意味
0	安定状態	速度が安定した状態
1	加速状態	通常加速中の状態
2	減速状態	通常減速状態
3	スリップ加速状態	スリップしながら加速している状態
4	スリップ減速状態	スリップしながら減速中している状態

【状態遷移図】－ 基本版　state_inf 信号処理 －

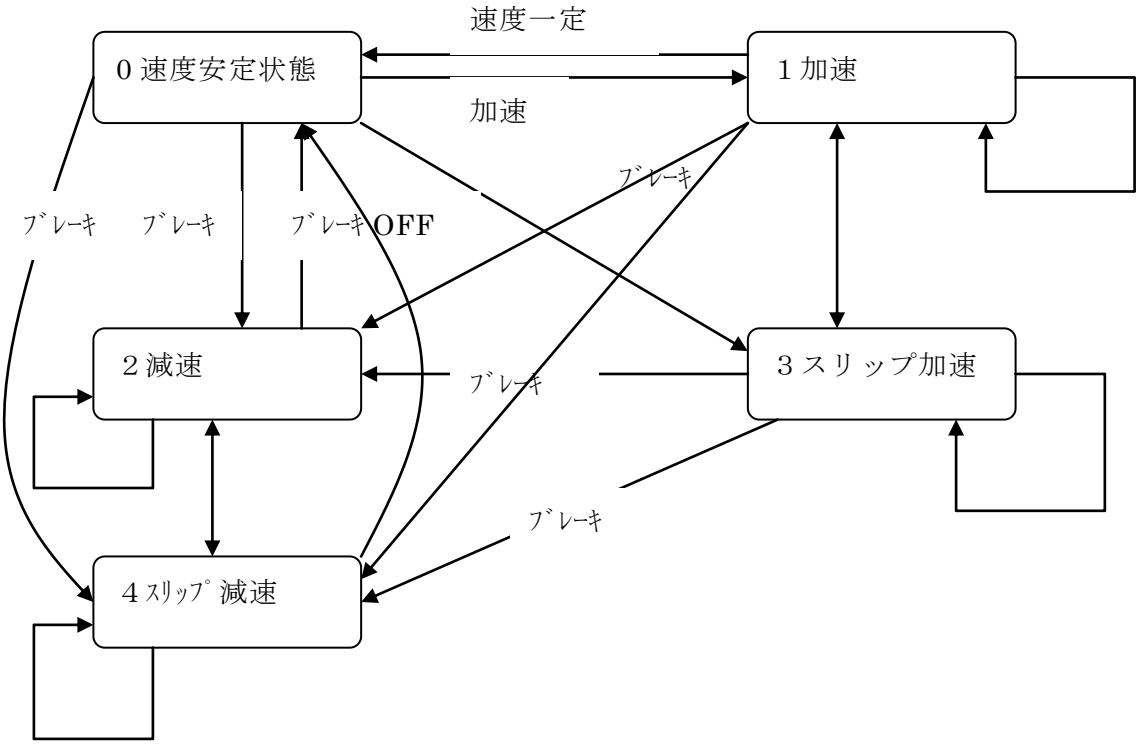


図 2.5-1　状態遷移図

【状態遷移図】－ フィードバック（update_ind 信号）機能追加内容　－

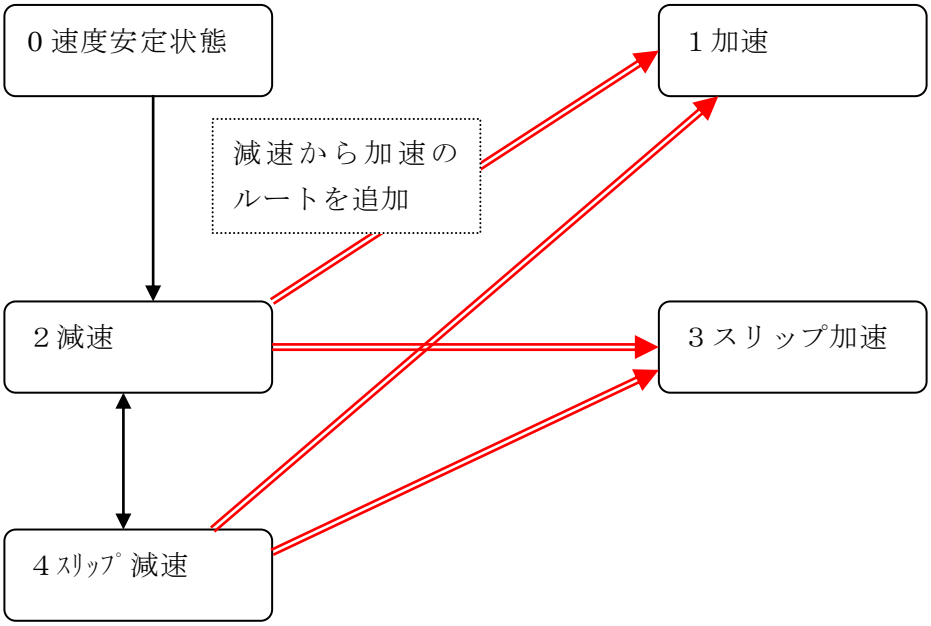


図 2.5-2　状態遷移図

【状態遷移表】

	update_ind 更新指示	state_inf 状態通知　brake	state_inf 状態通知 transmission	state_inf 状態通知 output	周期タイマ満了
0	ﾌｨｰﾄﾞﾊﾞｯｸ内容保存処理	通知内容保存処理	通知内容保存処理	rpm 保存	rpm 表示
1	ﾌｨｰﾄﾞﾊﾞｯｸ内容保存処理	通知内容保存処理	通知内容保存処理	rpm 保存	加速処理
2	ﾌｨｰﾄﾞﾊﾞｯｸ内容保存処理	通知内容保存処理	受け付けない	rpm 保存	減速処理
3	ﾌｨｰﾄﾞﾊﾞｯｸ内容保存処理	通知内容保存処理	通知内容保存処理	rpm 保存	ｽﾘｯﾌﾟ加速処理
4	ﾌｨｰﾄﾞﾊﾞｯｸ内容保存処理	通知内容保存処理	受け付けない	rpm 保存	ｽﾘｯﾌﾟ減速処理

【通知内容保存処理】

状態遷移決定論理と残トルク計算処理を行う。

	値増加	値減少
通常 brake 系	【ブレーキ踏み込む】 残トルク＝０－通知 throttle 値 スリップ計算 true->４スリップ減速状態 false->2 減速状態	【ブレーキ緩める】 残トルク＝０－throttle 値 スリップ計算 true->2 減速状態 false->2 減速状態
通常 throttle 系	【アクセル踏み込む】 残トルク＝通知 throttle 値－現 throttle 値 スリップ計算 true->3 スリップ加速状態 false->１加速状態	【アクセル緩める,エンブレ】 残トルク＝通知 throttle 値－現 throttle 値 スリップ計算 true->1 加速状態 false->１加速状態
ｽﾘｯﾌﾟ中 brake 系	【ブレーキ踏み込む】 残トルク＝０－通知 throttle 値 スリップ計算 true->４スリップ減速状態 false->2 減速状態	【ブレーキ緩める】 残トルク＝０－通知 throttle 値 スリップ計算 true->2 減速状態 false->2 減速状態
ｽﾘｯﾌﾟ中 throttle 系	【アクセル踏み込む】 残トルク＝通知トルク量－現トルク量 スリップ計算 true->3 スリップ加速状態 false->１加速状態	【アクセル緩める,エンブレ】 残トルク＝通知トルク量－現トルク量 スリップ計算 true->1 加速状態 false->１加速状態

【各擬似動作処理内容】

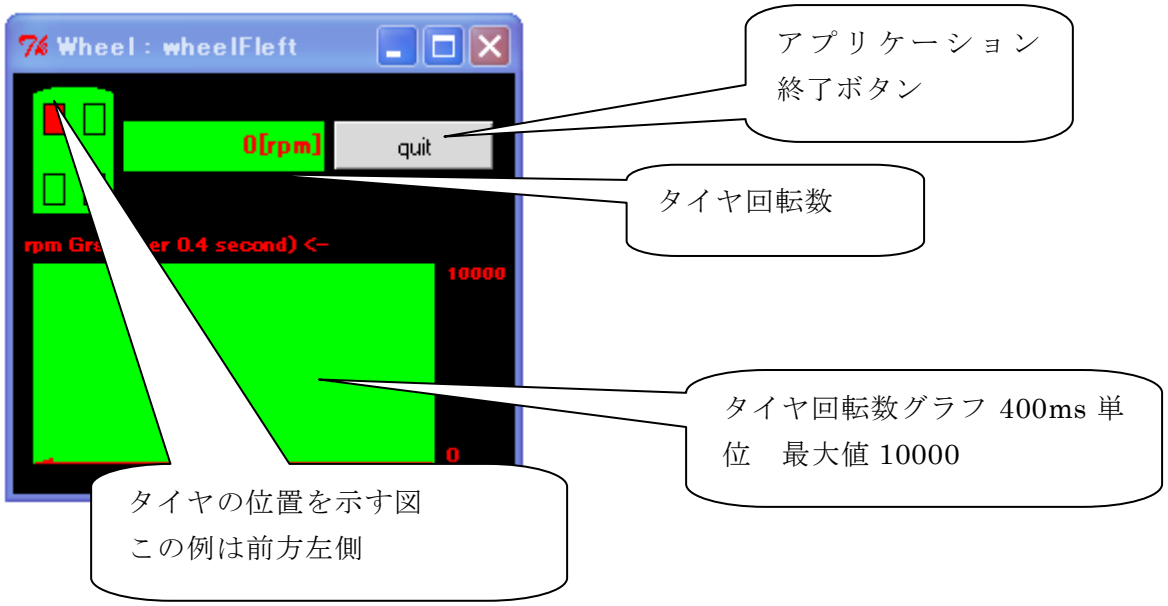
各種擬似動作の処理内容についてまとめる。

	処理内容
速度安定	【値状態】

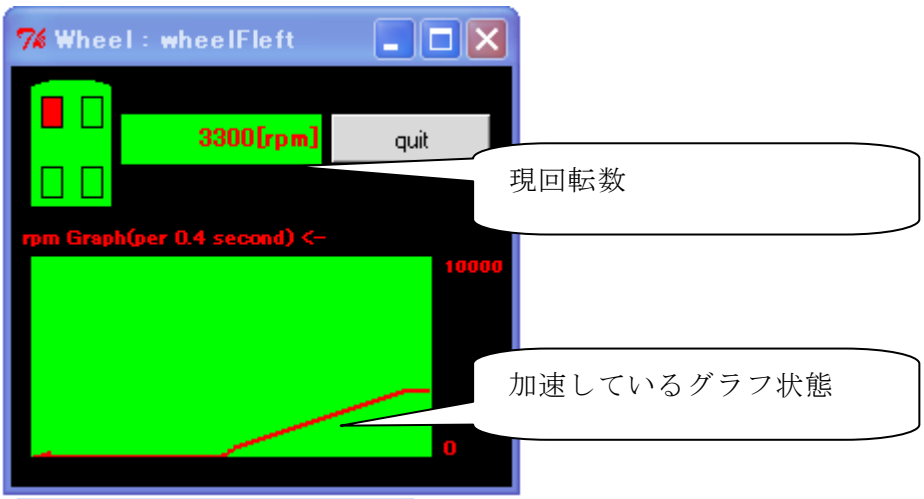
	処理残 throttle 値==0
加速	【処理内容】 加速量を求め、処理残 throttle 値と、加速後の throttle 値、rpm を計算する。(計算式は計算モジュール項参照) HMI 画面表示指示 output、dcs、transmission への通知
減速	【処理内容】 処理残 throttle 値と、減速後の throttle 値、rpm を計算する。(計算式は計算モジュール項参照) HMI 画面表示指示 output、dcs、transmission への通知
スリップ加速	【処理内容】 加速量を求め、処理残 throttle 値と、加速後の throttle 値、実 rpm、表示用 rpm を計算する。(計算式は計算モジュール項参照) HMI 画面表示指示 output、dcs、transmission への通知 HMI 画面表示指示 output,dcs,throttle への通知 スリップ状態終了でなければ、HMI 画面表示スリップ状態表示へ
スリップ減速	【処理内容】 処理残 throttle 値と、減速後の throttle 値、実 rpm を計算する。表示用 rpm は 0 に設定。(計算式は計算モジュール項参照) HMI 画面表示指示 output,dcs,throttle への通知 スリップ状態終了でなければ、HMI 画面表示スリップ状態表示へ

２．６　車輪回転数管理 HMI 仕様

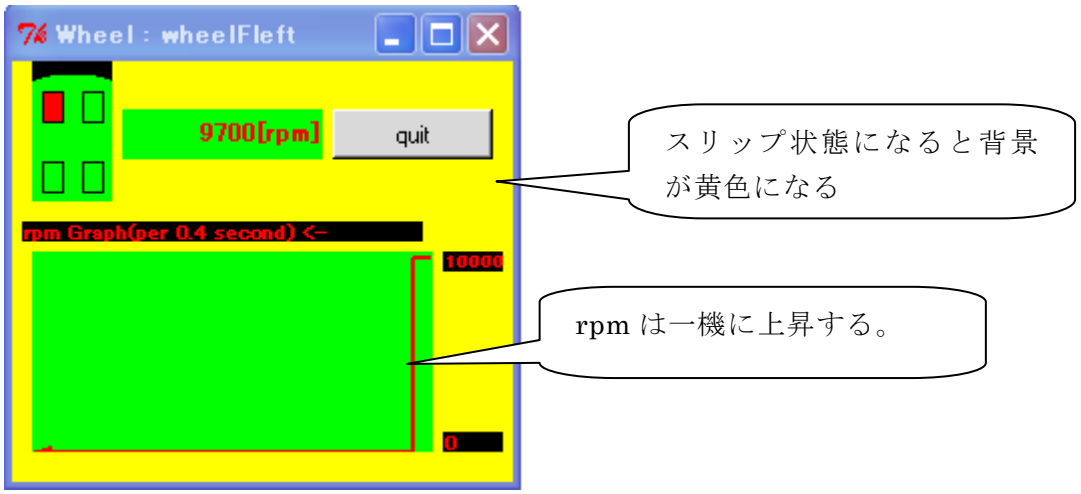
【通常状態　GUI】



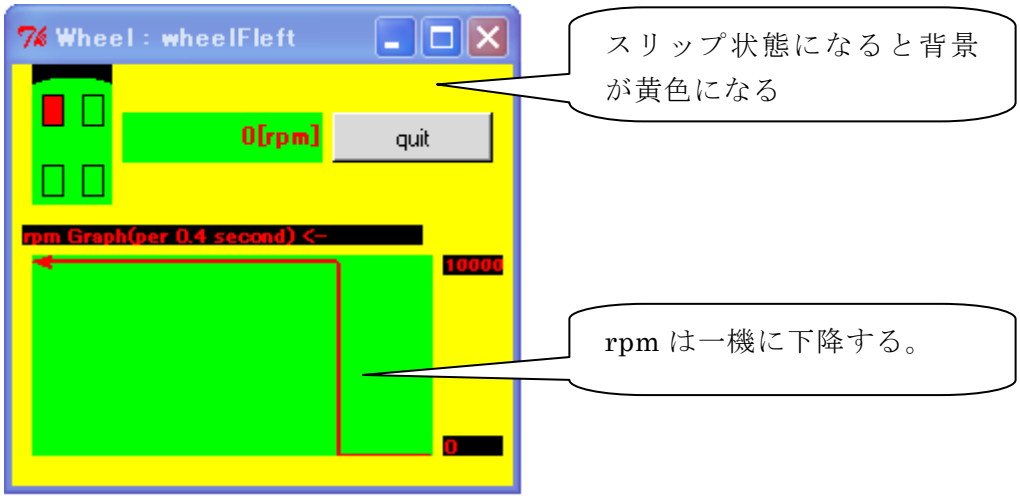
【前進中状態　GUI】



【スリップ加速状態　GUI】



【スリップ減速状態　GUI】



２．７　ブレーキ状態管理仕様

【プロセス辞書　管理データ】

名前	意味	値の範囲
val	前回踏み込み量（トルク）	0～100000

【各種値算出方法】

値	条件	通知信号内容
メッセージパラメータ設定内容	前回踏み込み量より増加	{state_inf, brake, [opetype,brake,userope,speeddown, torque,スライドバーの値]}
	前回踏み込み量より減少	{state_inf, brake, [opetype,brake,userope, speedup, torque,スライドバーの値]}

【状態】

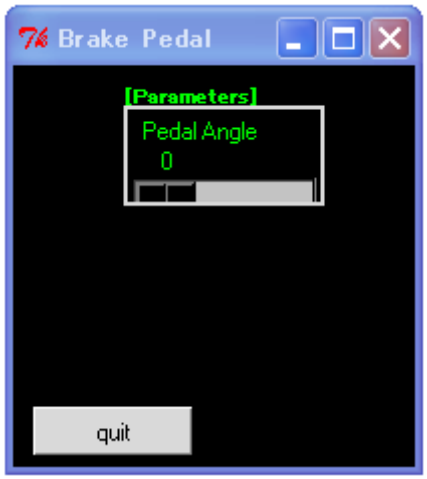
値	名前	意味
―	―	特に無し

【ブレーキ処理内容】

	処理内容
通知	各 Wheel への state_inf 状態変更通知 dcs への state_inf 状態変更通知 output への state_inf 状態変更通知 sim_db への通知内容保存依頼

２．８　ブレーキ状態管理 HMI 仕様

【GUI】



２．９　スロットル（アクセル）仕様

【プロセス辞書　管理データ】

名前	意味	値の範囲
val	前回踏み込み量（トルク）	0～100000

【各種値算出方法】

値	条件	通知信号内容
メッセージパラメータ設定内容	前回踏み込み量より増加	{state_inf, throttle, [opetype, throttle,userope, speedup, torque,スライドバーの値]}
	前回踏み込み量より減少	{state_inf, throttle, [opetype, throttle,userope, speeddown, torque,スライドバーの値]}

【状態】

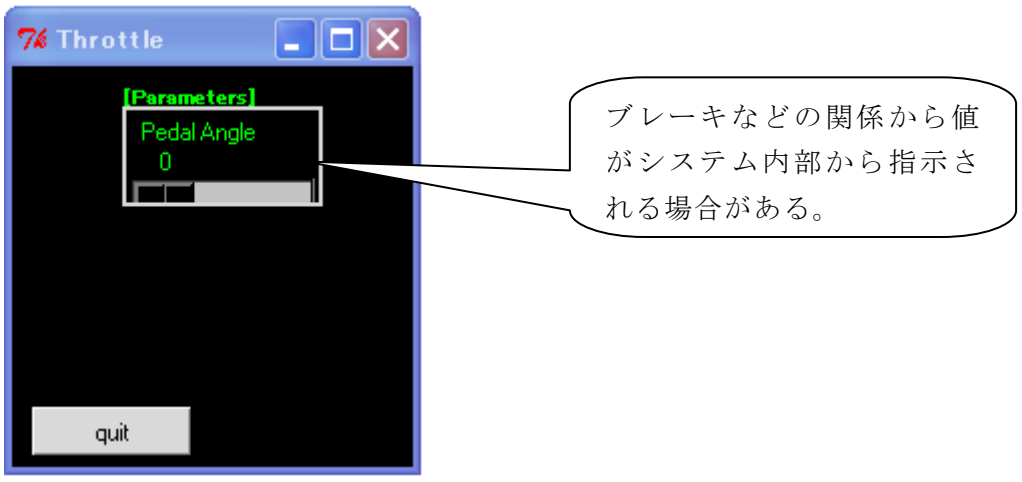
値	名前	意味
ー	ー	特に無し

【ブレーキ処理内容】

	処理内容
状態変更受信	スロットル入力 HMI へ表示するスロットル値の指示を行う。
通知	各 Wheel への state_inf 状態変更通知 dcs への state_inf 状態変更通知 output への state_inf 状態変更通知 sim_db への通知内容保存依頼

2．10　スロットル入力 HMI 仕様

【GUI】



2．11　センサー仕様

【プロセス辞書　管理データ】

名前	意味	値の範囲
–	ー	特に無し

【各種値算出方法】

値	条件	通知信号内容
メッセージパラメータ設定内容	前回踏み込み量より減少	{state_inf, センサー位置, [distance, スライドバーの値]}

センサー位置値：frsensor　前方センサー
bksensor　後方センサー
risensor　右側センサー
lfsensor　左側センサー

【状態】

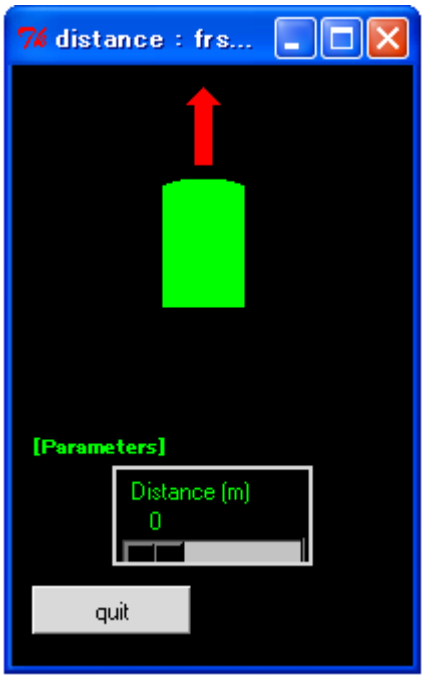
値	名前	意味
ー	ー	特に無し

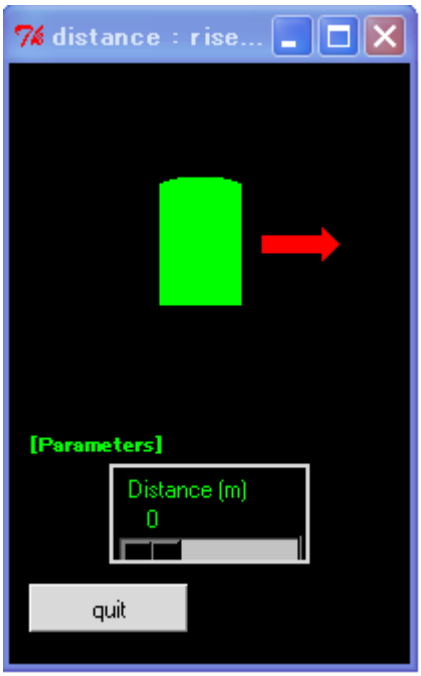
【ブレーキ処理内容】

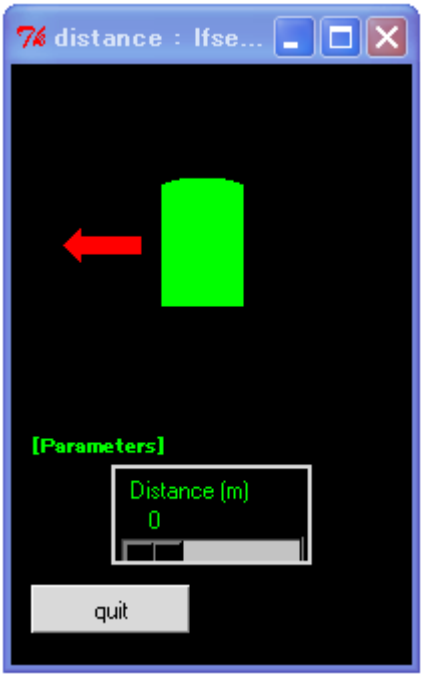
	処理内容
通知	dcs への state_inf 状態変更通知 sim_db への通知内容保存依頼

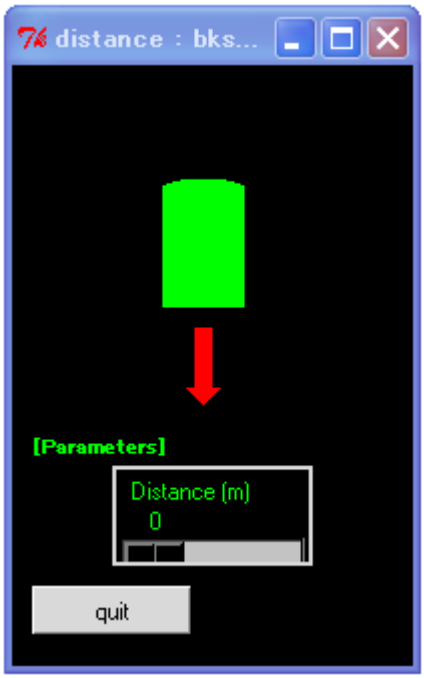
２．12　センサー入力 HMI 仕様

【GUI】

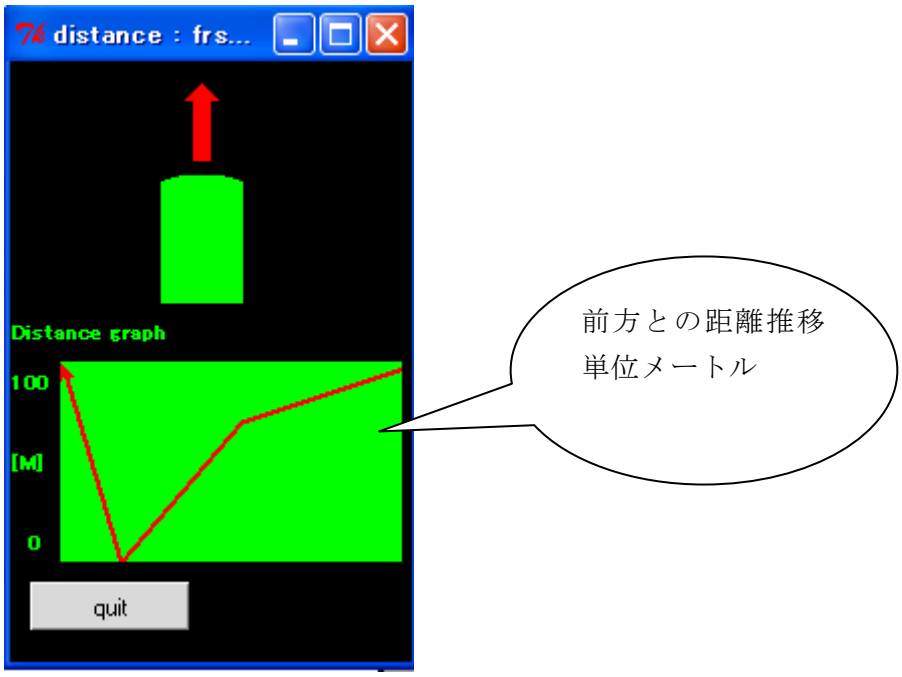








前方車シナリオ実行中のセンサーHMI



前方車シナリオ動作中の HMI 画面と通常画面は違う。自分の車の速度と前方車の速度から車間距離を測る。(計算式　は計算モジュール参照)

【センサー動作について】

実時間　0.5s 毎に処理。内部計算も 0.5s で計算する。

①初期　　初期値距離

②タイムアウト　　自分の車の速度と前方車の速度差を求める

③車間更新　　速度差から 0.5s で変化する車間を計算

以降タイムアウトを繰り返し

２．13　路面環境仕様

路面環境に関するデータは、Erlang 内臓の分散 DB Mnesia を使用している。

【管理データ】

名前	意味	値の範囲
statictor	路面静摩擦係数	0～1
dynamictor	動摩擦係数	静摩擦　2/3
area	空気抵抗面積（㎡）	1~

【各種値算出方法】

値	条件	通知信号内容
特になし		

【状態】

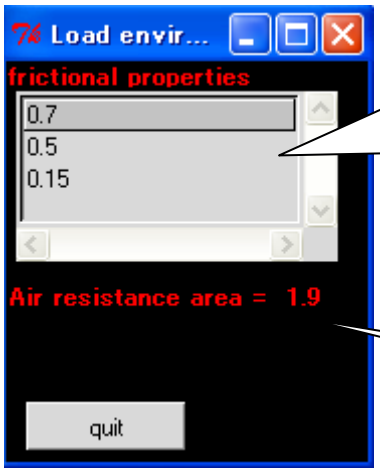
値	名前	意味
―	―	特に無し

【ブレーキ処理内容】

	処理内容
	HMI から受け取った値で、DB の値を変更する。

現在も動作中に変更も可能である。sim_db の仕様を参照のこと。

２．19　路面環境入力 HMI 仕様



摩擦抵抗値を選択

選択リストを変更するには、
loadenv_gui.erl 内の define 定義を変更する。
LIST_TITLE　：　表示用
VALUE　：　計算に使う値

空気提供面積
動的に変えるには sim_db のｲﾝﾀｰﾌｪｰｽ関数を起動して DB 内データを変更する。

２．14　車状況出力（Output）仕様

【プロセス辞書　管理データ】

名前	意味	値の範囲
state	状態	0~
speed	現走行時速	0~

stopc	停止経過時間	0~ 0.2ms づつ
fl	前方左　rpm	車輪から通知された値
fr	前方右　rpm	車輪から通知された値
rr	後方左　rpm	車輪から通知された値
rl	後方右　rpm	車輪から通知された値

【状態】

値	名前	意味
0	安定状態	速度が安定した状態

【状態遷移表】

	state_inf 状態通知　各 Wheel	state_inf　状態通知　brake	state_inf 状態通知 transmission	state_inf　状態　通知 throttle	周期タイマ満了
0	通知内容保存処理	停止時間計測開始	rpmスピード変換 wheel へ通知	何もしない	時速再計算

【各種値算出方法】

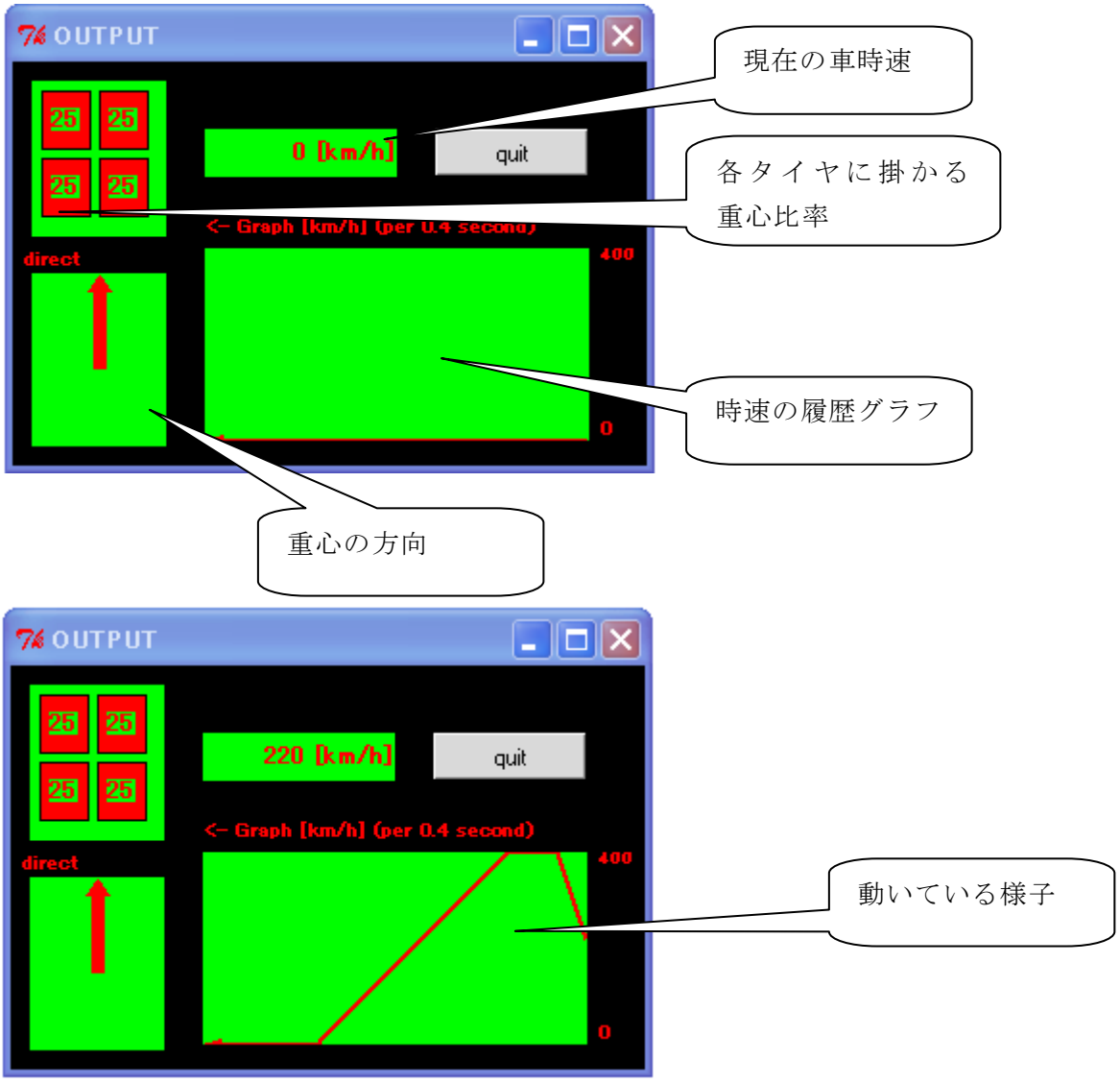
値	条件	算出方法
重心比率	全てのタイヤ 0rpm	[25,25,25,25]固定
	それ以外	タイヤ毎に以下の式で計算 rpm ＊ 100 ／（タイヤ 4 本 rpm 合計）
時速	安定状態	transmission から通知された rpm から時速計算
重心方向	特になし	前輪　左右の重心比率から算出

【参考】　前提値一覧

タイヤ円周	2ｍ			
最高時速	300	km/H		
ﾀｲﾔ最高rpm	2500	rpm		
throttle値	100000			
最高ﾆｭｰﾄﾝ	1000000	N		
1トルク	1	throttle		
車重量	1000	Kg		
垂直抗力	9800	N		
摩擦係数	可変	μ	0.7	乾いたアスファルト/乾いたコンクリートかつ、タイヤは普通
	可変	μ	0.5	濡れたアスファルト
	可変	μ	0.15	固くなった雪
摩擦力	6860	N	乾いたアスファルト/乾いたコンクリートかつ、タイヤは普通	
デフォルト値	4900	N	濡れたアスファルト	
	1470	N	固くなった雪	
前方車距離	可変	m	30	
空気抵抗	F＝Ｐ＊Ｃ＊Ｓ＊Ｖ^2／2□			
大気密度(P)	1			
空気抵抗(C)	0.35			
速度(V)	自動取得			
空気抵抗面積(S)	可変	㎡	1.9	

２．15　車状況出力（Output）HMI 仕様

【GUI】



２．16　重心制御　仕様

開発演習対象

２．17　Drive Control Simulate 仕様

開発演習対象

２．18　transmission 仕様

スロットルからの入力トルクを駆動輪へ伝える。駆動輪から通知された、現 rpm 値の平均を output へ通知する。

【管理データ】

名前	意味	値の範囲
now	各車輪回転数	0～

【各種値算出方法】

値	条件	通知信号内容
現 rpm		駆動輪の平均

【状態】

値	名前	意味
ー	ー	特に無し

【処理内容】

	処理内容
throttle から状態変化受信	駆動輪へ中継する。
update_ind 受信	駆動輪へ中継する。
タイムアウト	現平均 rpm を output へ通知する。

２.20　scenario

前方車の動作シリナオを管理する。また、シミュレート結果データを保存する機能も有する。

【管理データ】

名前	意味	値の範囲
scname	作成するシナリオ名	文字列
sc	作成するシナリオ	0.5 秒後毎の速度リスト[速度 1,速度 2]

【状態】

値	名前	意味
―	―	特に無し

【前提情報】

	処理内容
scenario	.¥scenario　配下をシナリオとみなす。
log	.¥log　　配下にログを格納する。

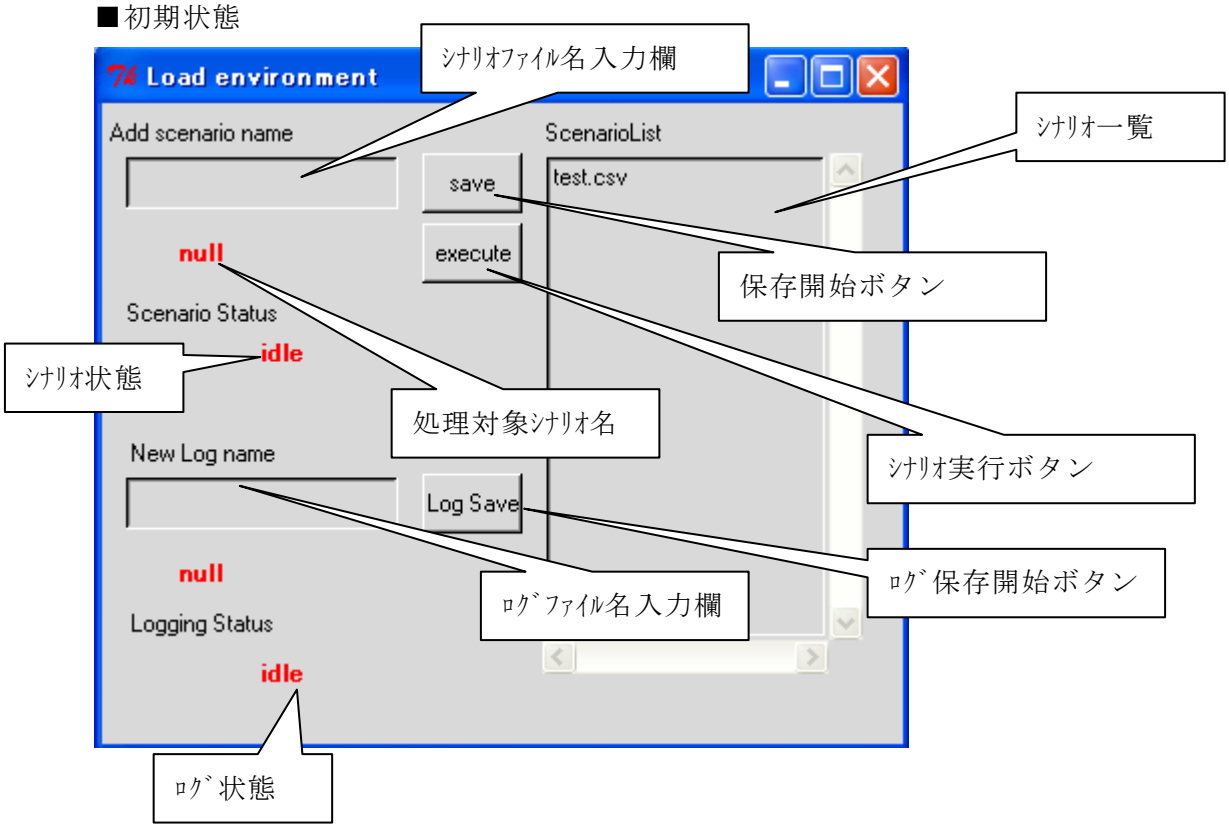
【処理概要】

	処理内容
シナリオ作成	.HMI のシナリオ作成依頼により、指定されたファイル名で scenario 配下に csv 形式のシナリオを作成する。
ログ保存	HMI のログ保存依頼により、指定されたファイル名で log 配下にログを格納する。

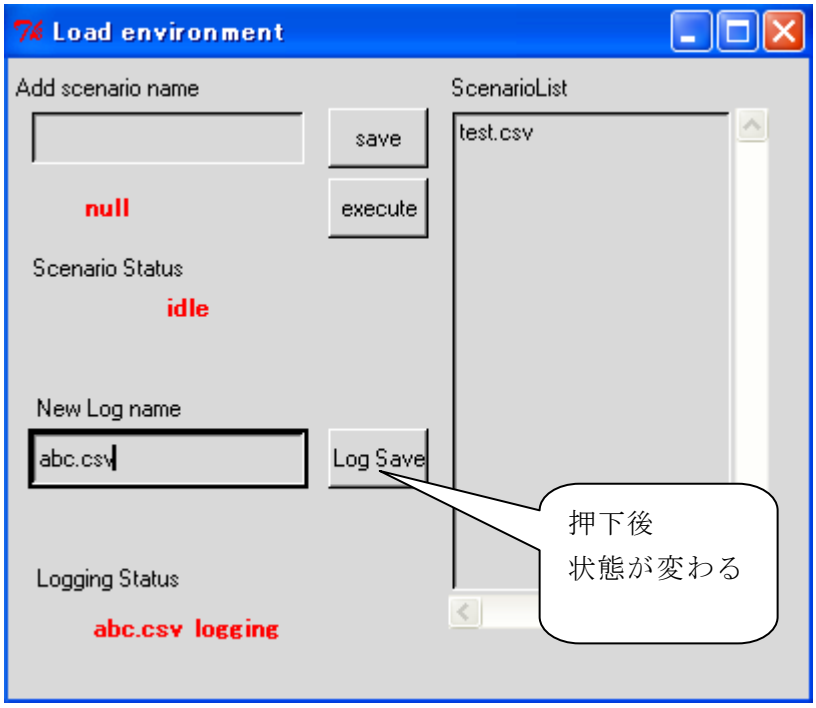
2.21　scenario HMI

【GUI】

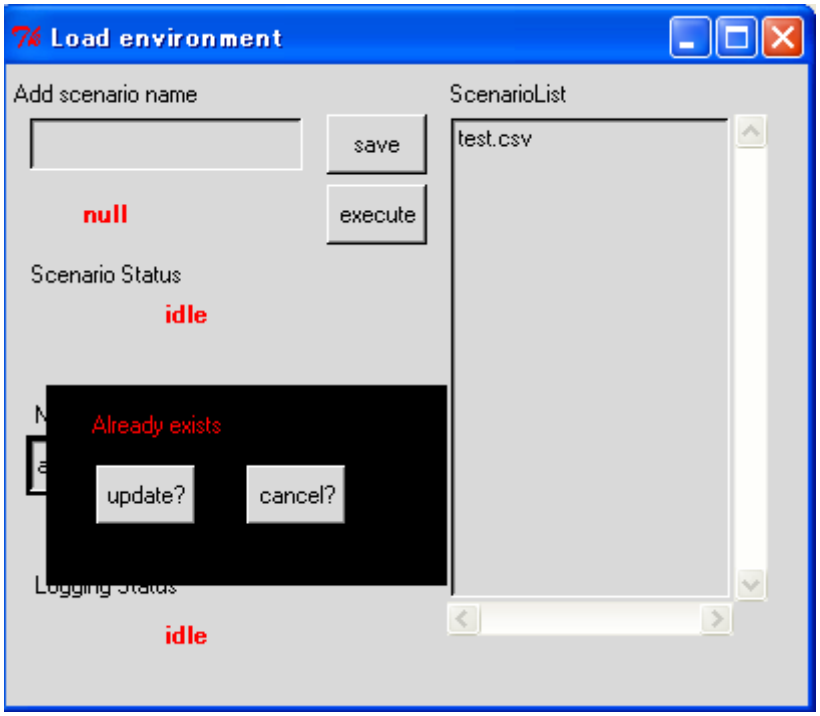
■初期状態



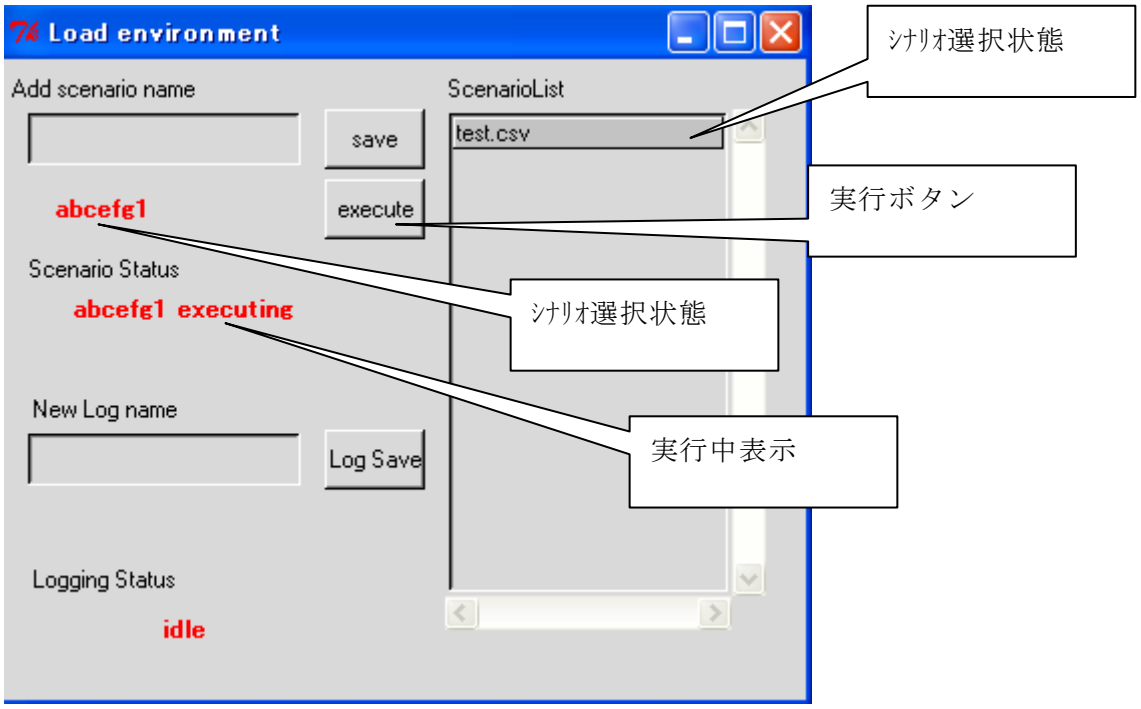
■ログ指定入力例



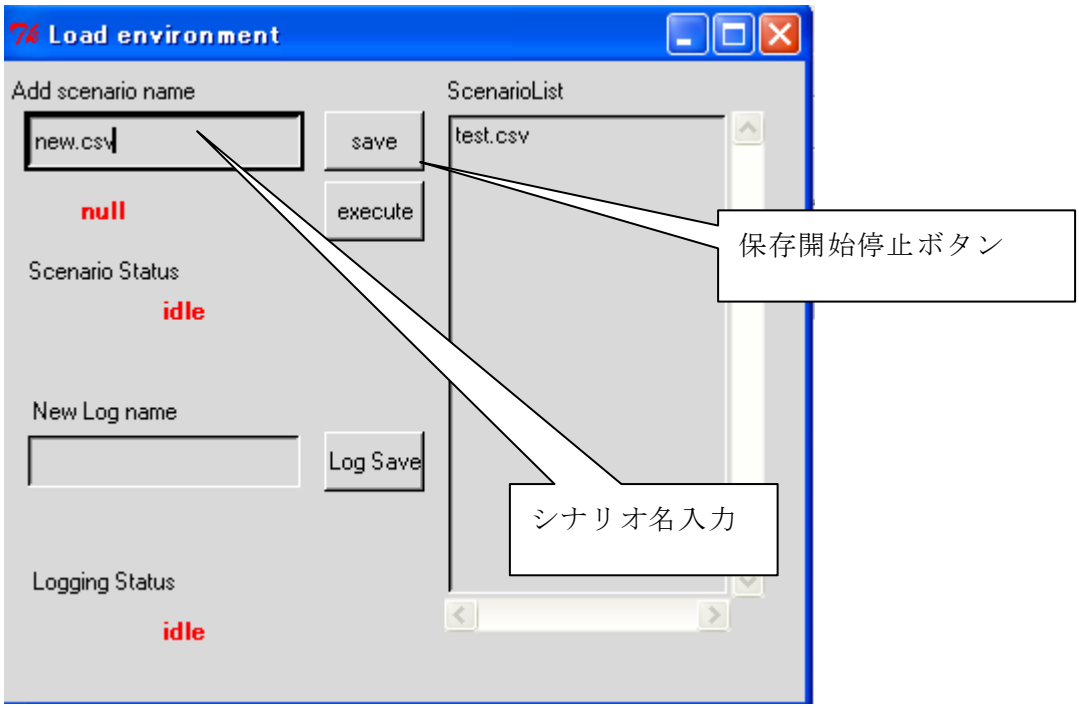
■ログ指定時同じファイル名があった場合の確認画面



■シナリオ実行状態



■シナリオ新規作成



２．22　計算式

【各種計算式一覧】

種別	内容
車間計算[m]	前回車間[m]＋（自分車時速[km/h]－前方車時速[km/h]） ×　1000 / 3600 ×　秒比率[s]

	■エンブレ時 減速量＝加速量/3 ・減速量より小さい 0 ・残減速量が減速量より小さい 残減速量 ・残減速量以上 現状態 throttle 値－　残減速量 ■加速時 ・残りの処理すべき throttle 値より加速量が小さい 現状態 throttle 値＋加速量 ・残りの処理すべき throttle 値より加速量が大きい 現状態 throttle 値＋残りの処理すべき throttle 値 ■スリップ加速時 加速量＝加速量/2 ・残りの処理すべき throttle 値より加速量が小さい 現状態 throttle 値＋加速量 ・残りの処理すべき throttle 値より加速量が大きい 現状態 throttle 値＋残りの処理すべき throttle 値 ■減速時 減速量＝brake 値 ・残りの減速すべき throttle 値が減速量より小さい 0 ・残りの減速すべき throttle 値が減速量より大きい 現状態 throttle 値－ brake 値×2/3 ■スリップ減速時 減速量＝動摩擦量／3 ・残りの減速すべき throttle 値が減速量より小さい 0 ・残りの減速すべき throttle 値が減速量より大きい
現状態 throttle 値	

残 throttle 値	■エンブレ時 減速量分を足す ■加速時 加速分を引く ■スリップ加速時 加速分を引く ■スリップ減速時、減速時 残 throttle 値
スリップ表示用 RPM	■スリップ加速時 残 throttle 値＋現 throttle 値 から rpm 変換 ■スリップ減速時 0
空気抵抗を考慮した実 RPM	throttle 値から RPM 変換−空気抵抗から計算した値
throttle 値から RPM 変換	round((throttle 値) / 40)
RPM から throttle 値 変換	round(RPM * 40)
rpm 加算量	round （1000 ×(1−最大 RPM との比の 2 乗))
スリップ開始	静摩擦量から計算した throttle 値×2 ＞ 絶対値(残 throttle 値)
スリップ停止	動摩擦量から計算した throttle 値×2 ＞ 絶対値(残 throttle 値)
空気抵抗	P(1)＊C(0.35)＊S＊V(speed^2 ／2

３． インターフェース仕様

【メッセージ基本構成】
　{メッセージ名、発アドレス、
　　[パラメータ１名、パラメータ１値,パラメータ２名,パラメータ２値,,,]}

　メッセージ名　表３－１
　発アドレス　　表３－２
　パラメータ　　表３－３

コーディング例

gen_server:cast({global,wheelFright},
　{update_ind,dcs,[opetype,throttle,userope,speedup,torque,Des]})

表２．１gensever名
メッセージ

　下線部が３章で規定する内容である。

　DCS（Drive Control Simulation）は gen_server として動作させるため、１．２項で示した部品間の通信方法を用いる。他プロセスへは、update_ind メッセージを通じてフィードバック制御を行う。

表３－１　メッセージ一覧

	メッセージ名	メッセージ名称	パラメータ	発アドレス
1	state_inf	状態変更通知	opetype, ユーザ操作種別 userop, ユーザ操作 torque　トルク	throttle brake
2	state_inf	状態変更通知	rpm　回転数 remaintorqu　残トルク	wheelFright wheelFleft wheelRright wheelRleft
3	state_inf	状態変更通知	distance　距離	frsensor bksensor risensor lfsensor
4	state_inf	状態変更通知	rpm　回転数 opetype, ユーザ操作種別 userop, ユーザ操作 torque　トルク	transmission

5	update_ind	更新指示	opetype, userop, torque	dcs
6	msgsav	メッセージ保存	保存メッセージ内容 【例】 {state_inf, risensor ,[distance,Parameters]}	frsensor bksensor risensor lfsensor throttle brake
7	getinfo_req	情報収集	speed	scenario output distance
8	getinfo_res	情報収集応答	speed	scenario output distance
9	scenario_req	シナリオ開始	scenario	scenario distance
10	scenario_res	シナリオ応答	scenario	scenario distance
11	scenario_can	シナリオ中止		scenario distance
12	logstart_ind	ログ収集開始指示		scenario distance
13	log_inf	ログ情報	speed、distance	scenario distance
14	logstop_ind	ログ収集停止		scenario distance

表３－２．メッセージアドレス一覧

	アドレス名	名称
1	wheelFright	車輪回転数管理　前右
2	wheelFleft	車輪回転数管理　前左
3	wheelRright	車輪回転数管理　後右
4	wheelRleft	車輪回転数管理　後左
5	brake	ブレーキ
6	throttle	スロットル（アクセル）
7	frsensor	前方センサー
8	bksensor	後方センサー
9	risensor	右センサー
10	lfsensor	左センサー

11	loadenv	路面環境
12	output	車状況
13	balancectl.	重心管理
14	dcs	Drive Control　Simulate
15	transmission	トランスミッション
6	scenario	シナリオ

表３－３　メッセージパラメータ名

	パラメータ名	パラメータ名称	値　範囲　単位
1	rpm	回転数	0~5000　1rpm/1 単位
2	torque	トルク	0~100　0.2ﾄﾙｸ/1 単位
3	remaintorqu	残トルク	0~100　0.2ﾄﾙｸ/1 単位
6	distance	センサー識別距離	0~100　残り 1000cm-10cm*単位 例 前方センサー　50 残り 5m に接近
7	userope	ユーザ操作	speedup,speeddown
8	opetype	操作種別	throttle, brake
9	speed	スピード	
10	scenario	シナリオ	

４．構成ファイル一覧

	ファイル名	概要
1	sim_app.erl	シミュレーションアプリケーション層
2	sim_sup.erl	シミュレーションプロセス監視
3	sim_part.erl	部品プロセス管理サーバ
4	sim_db.erl	ｼﾐｭﾚｰﾀﾃﾞｰﾀﾍﾞｰｽ
5	wheel.erl	前右　車輪回転数管理
6	wheel_gui.erl	前右　車輪回転数表示
7	brake.erl	ブレーキ状態管理
8	brake_gui.erl	ブレーキ入力 HMI
9	throttle.erl	スロットル（アクセル）
10	throttle_gui.erl	スロットル入力 HMI
11	distance.erl	前方センサー
12	distance_gui.erl	前方センサー入力 HMI
13	balancectl.erl	重心制御
14	dcs.erl	Drive Control Simulate
15	output.erl	車状況出力
16	output_gui.erl	車状況出力 HMI
17	loadenv.erl	路面環境
18	brakesim.app	Erlang アプリケーション定義ファイル
19	make.bat	win 用 コンパイルバッチファイル
20	transmission.erl	トランスミッションソースファイル
21	scenario.erl	シナリオファイル
22	loadenv_gui.erl	路面環境 HMI
23	scenario_gui.erl	シナリオファイル HMI

５．DCS コーディング例

デフォルトの dcs.erl 雛形ファイルについて説明する。

```
-module(dcs).
%% --------------------------------------------------------------------
%% Include files
%% --------------------------------------------------------------------
%% ---------- インクルードファイルを記載 --------
%% External exports
%% --------------------------------------------------------------------
-export([start/0]).
%% ---------- 外部公開関数を定義時記入 --------
%% Macros
%% --------------------------------------------------------------------
%% ---------- マクロを定義時記入 --------
%% Records
%% --------------------------------------------------------------------
%%          レコードを定義時記入
%% ====================================================================
%% External functions       外部公開関数
%%
%% ====================================================================
start() ->
        io:format("------------dcs Start node=~p-------------~n",[ erlang:node()]),
        process_flag(trap_exit, true),
        event_loop(),
        ok.
```

gen_server から起動された時に動く関数
子プロセスを生成する場合はここに spawn 文を記述する。

```
%% Internal functions
%% ====================================================================
event_loop() ->
        receive
                {state_inf,frsensor, Parameter} ->
                        Des = getParam(distance,Parameter),
                        gen_server:cast({global,wheelFright},                    {update_ind,
dcs,[opetype,throttle,userope,speedup,torque,Des ]});
                        event_loop();
    ～～～～～～～略　～～～～～～～～～～～～～～
                {state_inf,throttle, _Parameter} ->
                        event_loop();
                {state_inf,brake, _Parameter} ->
                        event_loop();
                {'EXIT', Pid, Why} ->
                        io:format("child process[~p] terminated : ~p~n", [Pid, Why]),
                        exit(error);
                Any ->
                        io:format("DCS unknown type received : ~p~n", [Any]),
                        event_loop()
        end.

%%get parameter from message------
getParam(Key,[Hk,Hv|_]) when Key =:= Hk->
        Hv;
getParam(Key,[_,_|T])->
        getParam(Key,T);
getParam(_,[])->
        false.
```

各種メッセージ受信処理はここの処理を膨らませる

現状は速度変更時に意図的にエラーメッセージをが表示している。

メッセージパラメータ取り出し関数

別紙－１

インストール手順

１．概略

１台の PC で node1@localhost と node2@localhost の分散環境で構築した場合を例にとる。

手順１　コンパイル

手順２　シミュレータで必要な Mnesia 環境などを構築

手順３　起動

２．手順詳細

■手順１

ソースファイル一覧を解凍する。

OS　Window 時は、配布の make.bat をコマンドプロンプトで実行する。

【実行イメージ】

```
C:¥work¥brakesim_V1>make

C:¥work¥brakesim_V1>echo off
"######wheel_gui.erl######"
"######wheel.erl######"
"######sim_sup.erl######"
"######sim_part.erl######"
"######dcs.erl######"
"######distance.erl######"
"######distance_gui.erl######"
"######throttle.erl######"
"######throttle_gui.erl######"
"######brake.erl######"
"######brake_gui.erl######"
"######output.erl######"
"######output_gui.erl######"
"######loadenv.erl######"
"######sim_db.erl######"
"######sim_app.erl######"
"######balancectl.erl######"
```

■手順２

手順１で解凍した Dir 上で erl を 2 ノード起動

>erl –sname node1@localhost

>erl –sname node2@localhost

いずれかの erlangBeam 上で環境構築コマンド実行

(node2@localhost)3> sim_sup:init_sim().

=INFO REPORT==== 31-May-2010::00:47:55 ===

application: mnesia

exited: stopped

type: temporary

ok

(node2@localhost)4>

node1 も同様のメッセージが出力される。

起動した Dir 配下に次の 2 つの Dir が作成されることを確認する。

Mnesia.node1@localhost

Mnesia.node2@localhost

■手順３

OSS Erlang 開発演習用教材　ブレーキシミュレータは、erlang アプリケーションとしてパッケージ化されている。よって、Erlang アプリケーションの起動方法について示す。

Erlang アプリケーション名：brakesim

起動時　入力方法（注：各ノードを立ち上げておくこと）

(node2@localhost)4> application:start(brakesim).

sim_sup init start

~~以降略~~

終了方法

(node2@localhost)5> application:stop(brakesim).

※タイミングによって、Mnesia の開始が遅れて起動失敗するケースがあるが、再度続けて実行すると 2 度目は成功する。

別紙－２

ノード構成変更方法

2 台以上の PC を使う場合は、erlang の起動方法はインストール時に示した内容ではなく、次の例に示すようなコマンドパラメータとなる（詳細は、Erlang マニュアル参照)。

erl –name xxx@IP アドレス（ホスト名）-cookie yyyyyy

■手順１

sim_sup.erl のマクロ定義を変更

例

-define(NODE1, 'node1@localhost').
-define(NODE2, 'node2@localhost').

これを変更

使用するノード名に変更 ‘xxx@IP アドレス’

■手順２

各部品の起動ノードを、手順１で記述したマクロ名に変更

例

```
Dcs = {{dcs,?NODE1},
                {sim_part, start_link,[dcs,?NODE1,dcs,start,[]]},
                permanent,
                brutal_kill,
                worker,
                [sim_part]},
```

ここを変える

注：分散 DB　Mnesia もノードを意識するので、再構築が必要である。（インストール手順１）

別紙－３

ソースファイル追加手順

Erlang ソースファイルを追加した場合の手順について示す。

■手順１
コンパイル用のバッチファイルへ追加する erlang ファイル名を追記

編集ファイル：make.bat

■手順２
アプリケーション定義ファイルへ追加する erlang モジュール名を追記

編集ファイル：brakesim.app
編集箇所　　：modules 欄

■手順３　部品（sim_part）として機能させるとき
部品起動処理を追加。

編集ファイル：sim_sup.erl
他部品を参考に編集する。