Vol.2014-ITS-58 No.3
2014/9/19

加速が不要なタイミングでアイドルストップを行う装置のソフトウェアの改善にともなう，移植性，可読性の高いソフトウェア構造の実現

福島 悠史[†1]　芝田 健男[†1]　谷道 太雪[†1]　金子 彰一[†2]

自車情報，及び外界認識センサを用いて取得した先行車情報を基に，先行車への追従走行時に加速が不要なタイミングでアイドルストップを行う装置のソフトウェアの改善を行い，移植性，可読性の高いソフトウェア構造を実現した．

走行状況は車速，加速度，車間距離，相対速度等の複数のパラメータの組み合わせで時間とともに刻々と変化するため，条件文でソフトウェアを構築した場合には，条件文の複雑化や分岐数の増加等によりソフトウェアの可読性や保守性が損なわれた．そこで，ソフトウェアの可読性，保守性を向上させ量産化に向けて，条件文で使用している各種入力情報をグループ化し，自車停止，渋滞，先行車右左折，先行車停止予測等の走行状況を各状態で定義し，その各状態を用いてのソフトウェアの再構築を図った．

結果，ソフトウェアの可読性や保守性の向上に成功した．

Improving Software Portability and Readability with Modification of the Idle Reduction Control.

Yuji Fukushima　Takeo Shibata　Taisetsu Tanimichi　Shoichi Kaneko

We reconstructed the software of the device that performs an idol reduction if own vehicle need not acceleration during following a precedent vehicle.

The driving situation changes by the change of plural parameters such as vehicle velocity, acceleration, relative distance, the relative velocity every moment. At first, we built software by combining with conditional statements. Portability and readability of the software decreased because the software became getting complex of the condition and increasing the number of the divergence. So, we reconstructed the software by grouping input data and by classifying driving situations.

As a result of this measure, we succeeded in improving portability and readability of software.

1. はじめに

近年, 外界認識センサを用いた運転支援システムの開発，および市販化が進んでいる[1]．外界認識センサを使用することで，先行車情報や道路情報，信号情報などが取得可能になるため，制御システムによる運転支援機能の幅が拡大しつつある．

外界認識センサを用いて燃費向上を図る機能として，自車情報，及び外界認識センサを用いて取得した先行車情報を元に，先行車への追従走行時に加速が不要なタイミングを判定し，自動でアイドルストップを行う装置を開発した．

この装置を搭載した車両の構成を図 1-1 に示す．車両は自車に搭載された各種 ECU[*1] に加え，外界認識センサを搭載している．外界認識センサからは先行車との相対速度，車間距離などの外界認識情報を取得することができる．

アイドルストップ制御は，ACC[*2] などの制御を行う制御 ECU 内に実装されており，制御 ECU は車両に搭載されている各種 ECU と CAN[*3] で接続されている．

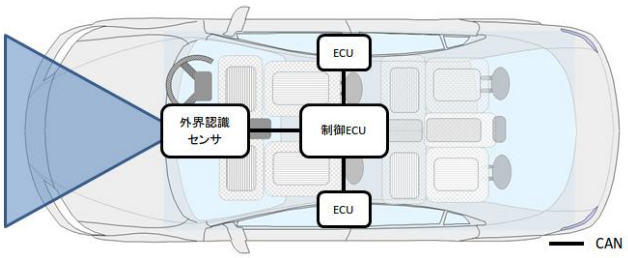

図 1-1 車両構成

この制御に使用しているソフトウェアは，装置の効果検証に使用する目的で開発速度を重視して開発したため，条件文の複雑化や分岐数の増加が発生し可読性や保守性が低下している．そこで，ソフトウェアの解析を行い，問題点を分析し対策を実施した．

本論文では，外界認識センサを用いた加速が不要なタイミングでアイドリングストップを行う装置のソフトウェアにおける問題点，対策の手法について述べる．

†1 日立オートモティブシステムズ株式会社
Hitachi Automotive Systems, Ltd.
†2 株式会社日立ソリューションズ
Hitachi Solutions, Ltd.

* 1 Electronic Control Unit
*2 Adaptive Cruise Control
*3 Controller Area Network

2. ソフトウェア構造

制御 ECU に実装されているソフトウェアは，ACC や AEB*4 などの制御機能を含む Application 層，Application から出力される制御指令値を調停する Control 層，車種毎の処理を行う Virtual Car 層，H/W に準じたドライバ処理を行う Platform 層から構成されている．アイドルストップ制御は Application 層に実装されている．制御 ECU のソフトウェア構成を図 2-1，アイドルストップ制御の処理フローを図 2-2 に示す．

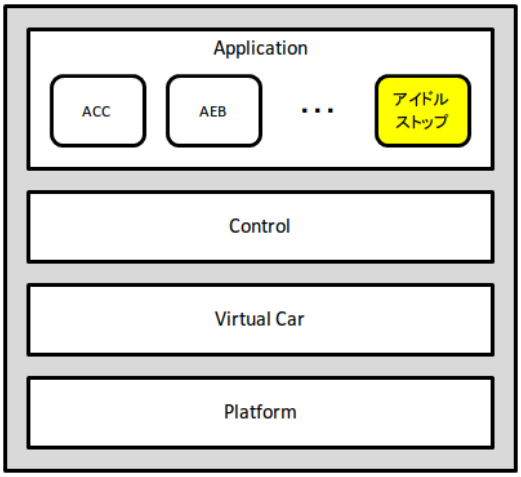

図 2-1 制御 ECU のソフトウェア構成図

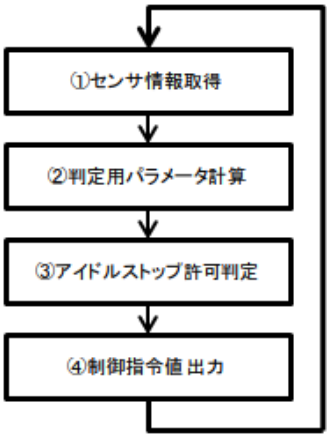

図 2-2 アイドルストップ制御の処理フロー

図 2-2 に示した①は CAN 等で接続された各種センサからの情報を取得する．②は各種センサから取得した情報をもとに，アイドルストップ許可判定に使用するパラメータを算出する．③は現在の走行状況からアイドルストップの許可条件に該当するか判定する．④ではアイドルストップの許可状態に応じてアイドルストップを行うよう制御指令値を出力する．

*4 Autonomous Emergency Braking

①および④は，アイドルストップ以外の他の制御機能でも使用する処理である．そのため，再構築を検討する対照処理は②判定用パラメータ計算，③アイドルストップ許可判定の 2 つとした．

3. 問題点

現在のソフトウェア構成は効果検証のために実車でチューニング等を実施しているため，開発速度を重視した開発を行っており，保守性，可読性の確保に重点を置いた開発を行っていなかった．その為，条件分岐の増加や条件文の複雑化により保守性と可読性が低下している．

これらの問題の原因を特定するため，プログラムの複雑度を示す指標である経路複雑度とネストレベルの測定を実施した．現状のアイドルストップ制御ソフトウェアの経路複雑度の測定結果を図 3-1 に，ネストレベルの測定結果を図 3-2 に示す．

なお，経路複雑度は関数内部に存在する条件分岐の数を示している．経路複雑度の高い関数は分岐数が多いため，テスト工程において評価する経路数が多くなり，保守性の低下につながる．また，分岐数が多いため可読性が低下する．経路複雑度の指標は McCabe の定義によると 10 以下が望ましいと言われている[2].

ネストレベルは条件分岐のネストの深さを示している．ネストレベルの高い関数は，ネストが深い位置の処理に到達する条件が複雑になり，テストケースの作成が困難となる[3]．ネストレベルの指標はソフトウェアの特性により上下するため，本ソフトウェアではネストレベル 4 以下が望ましいと定義した．

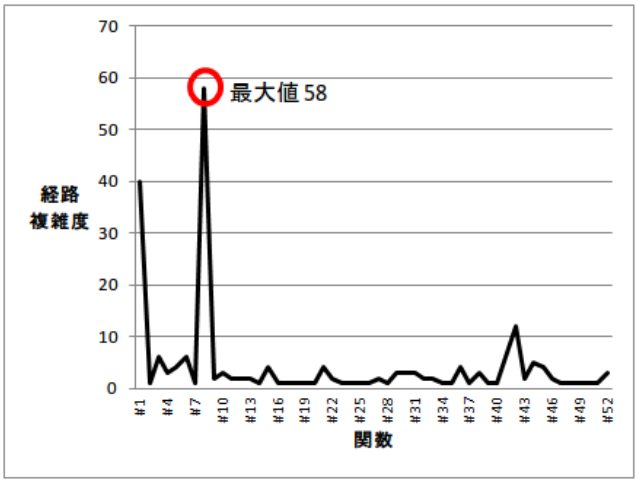

図 3-1 再構築前の経路複雑度

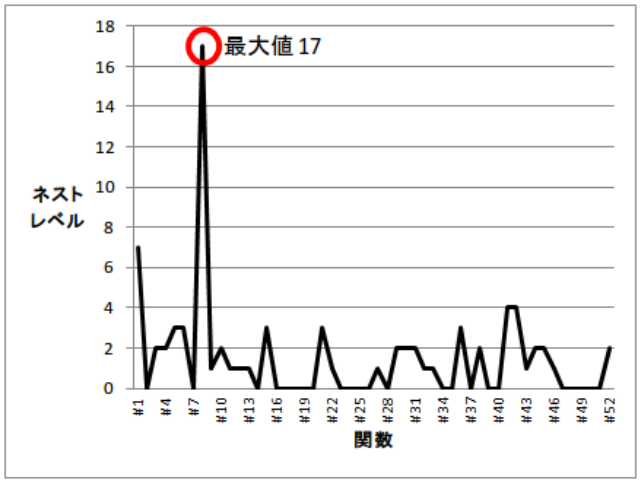

図 3-2 再構築前のネストレベル

測定結果から，1 つの関数が経路複雑度 57，ネストレベル 17 と他の関数より高い数値を示していることとなった．

この関数は現在の走行状況を判定する関数である．走行状況の判定関数は，車速，加速度，車間距離，相対速度等の複数の入力パラメータの組み合わせで，時間とともに刻々と変化するため，経路複雑度，ネストレベルが他の関数より高い数値を示していた．また，本関数は運転者に違和感を与えないようにする為，実車上でチューニングを行い，走行状況の追加，削除，及び条件変更を頻繁に行っていた．

この関数に注目して構造を解析し，次に示す問題点を抽出した．

(1) 判定条件の複雑化

走行状況の判定には，自車の速度，加速度，運転者の操作などの自車情報に加え，外界認識センサから取得した先行車情報などが必要となる．これらの入力パラメータの値を閾値と比較し，その結果を組み合わせることで走行状況を判定している．走行状況の判定条件の例を図 3-3 に示す．

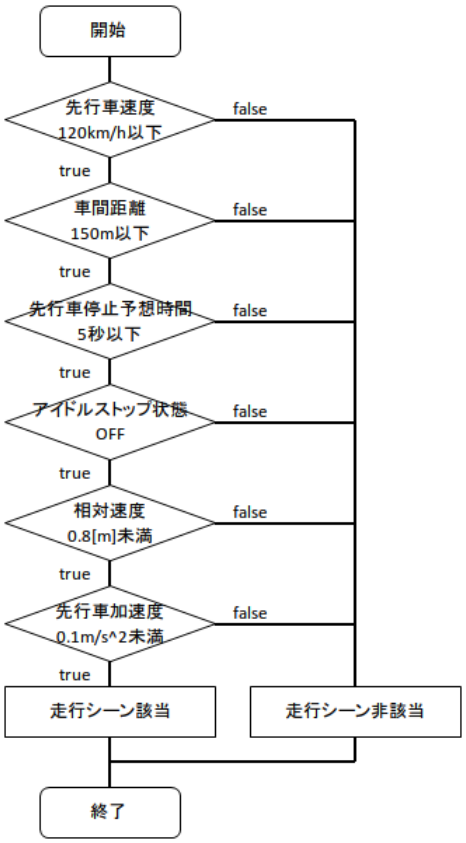

図 3-3 走行状況 判定条件の例

上記の例では 6 つの入力パラメータに対する判定が必要であり，これらの判定を組み合わせて走行状況を判定している．このように 1 つの走行状況を判断するには複数の入力パラメータが必要となるため，条件文が複雑となり可読性が低下する要因となっている．

(2) 分岐数の増大

走行状況は低速追従走行，高速追従走行，渋滞追従走行，坂道走行，自車停止等，様々な走行状況を定義している．

走行状況を複数定義すればするほど，走行状況に応じたアイドリングストップ制御を実施する事が出来るため，運転車に違和感の無いアイドリングストップを実施することができる．その為，走行状況は複数となり，その分走行状況判定分も複数となっている．走行状況判定処理のフローを図 3-4 に示す．

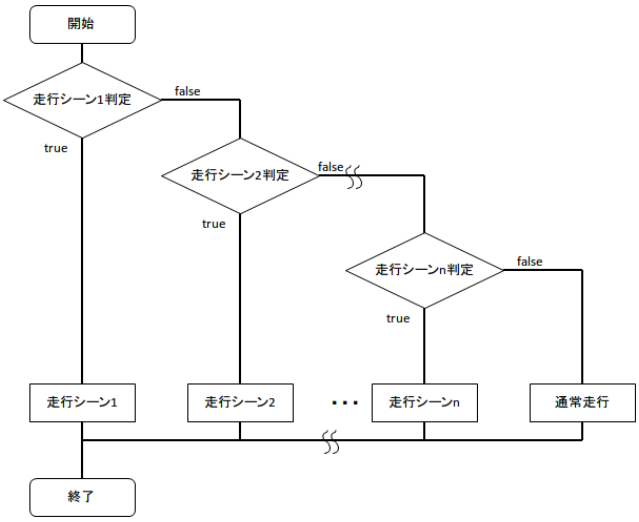

図 3-4 走行状況判定処理の構造

現在の構造では走行状況の数を増やすたび関数の経路複雑度が上昇し，保守性が低下する．

(3) 参照するグローバル変数の増加

走行状況判定に使用する入力パラメータは，各種センサから取得しグローバル変数に設定している．参照するグローバル変数が増加すると，変数の更新を行う別の処理にタイミング等で依存した構造となり，依存関係にある変数が更新前，更新後と変数間で矛盾が発生する可能性がある．また，ソフトウェアの結合度が高くなり，ソフトウェアは保守性および移植性が低下する原因となる．

4. 対策

前章で抽出した問題を改善するため，走行状況の判定処理の構造を再実装することにし，以下に示す対策を実施した．

4.1 対策内容

(1) 入力パラメータのグループ化

走行状況の判定に使用する入力パラメータの種類は，自車および先行車の速度，加速度，相対速度など多岐にわたっている．これらの入力パラメータは複数の走行状況の判定に使用しており，複数の箇所で参照される．例えば，走行状況 A にて自車速度から自車が停止していることを判定し，その後走行状況 B にて自車速度が 5km/h 以上であることを判定するという参照である．複数回の入力パラメータ参照によりコード量が増加し，判別条件が複雑化している．

そこで，判定に使用する入力パラメータを種類毎に分類し，閾値の判定を一括で行うことにした．入力パラメータの分類を表 4-1 に示す．

表 4-1 入力パラメータの分類

No	分類名
1	自車速度
2	先行車速度
3	自車加速度
4	先行車加速度
5	車間距離
6	相対速度
7	操舵角
8	スイッチ操作※1
9	動作タイマー※2
10	演算パラメータ※3
11	アイドルストップ状態
12	先行車停止予想
13	その他

(※1) 運転者によるスイッチ操作の判定
(※2) エンジン始動からの経過時間等の判定
(※3) 衝突時間等の内部計算したパラメータの判定

入力パラメータの分類を用いて，各入力パラメータに対する判定を局所化し，判定の結果をビットフィールドとして管理した．対象のビットが 1 になれば，判定条件が正であると判断出来る．局所化した判定処理の例を図 4-1 に，ビットフィールドの定義例を表 4-2 に示す．

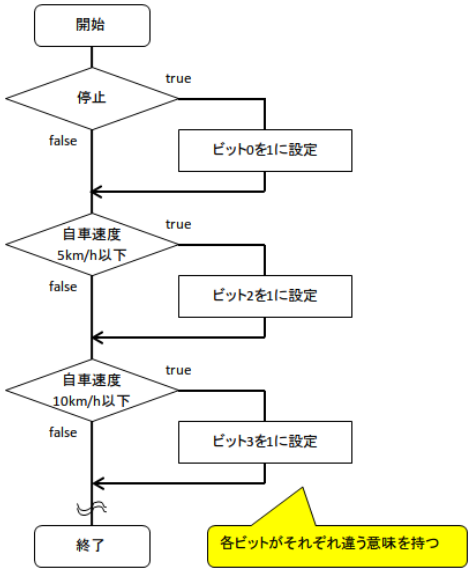

図 4-1 局所化した判定処理の例

表 4-2 ビットフィールドの定義例

ビット	意味
0	自車速度が 0km/h 以下
1	自車速度が 5km/h 以上
2	自車速度が 10km/h 以上
⋮	⋮
16	自車速度が 100km/h 以上

このように，1 つの入力パラメータに対する複数の閾値による判定を実施し，その結果を 1 つの変数で管理する．

また，グローバル変数である入力パラメータの参照箇所をまとめることで，依存関係にある変数に矛盾が発生するという問題を防止でき，結合度の低い関数を実現することができる．

この問題は，例えば走行状況 A と走行状況 B で同一の入力パラメータ X を参照する場合，走行状況 A の判定直後に割込み処理で X の値が更新されると，本来は走行状況 A が成立するべきタイミングで走行状況 B が成立する可能性があるということである．割込みによる変数の矛盾を図 4-2 に示す．

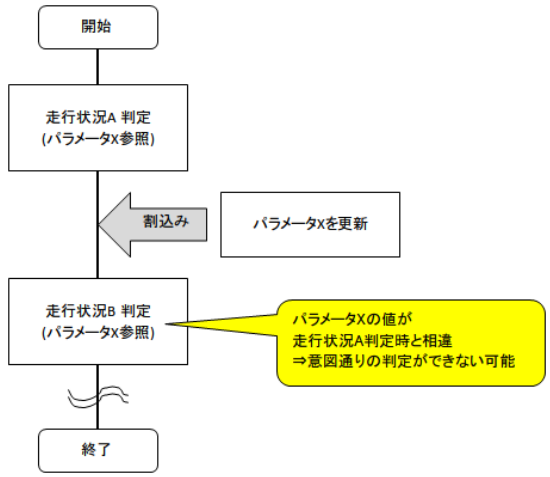

図 4-2 割込みによる変数の矛盾

(2) 走行状況判定処理の再構築

従来の走行状況判定処理は，複数の走行状況に対する判定を複数の条件判定文を組み合わせることで実現している．この構造では走行状況の個数に応じて分岐数が増加するため，走行状況判定の経路複雑度が上昇している．

そこで，走行状況の判定条件を前段階で定義したビットフィールドを使用してテーブル化し，共通の判定関数で判定するよう機能の再構築を実施した．走行状況の必要条件定義例を表 4-3 に示す．

表 4-3 走行状況の必要条件定義例

走行状況	自車速度	先行車速度	…	その他
走行状況 1	0x000E	0x0004	…	0x0000
走行状況 2	0x0002	0x001A	…	0x0000
⋮	⋮	⋮	⋮	⋮
走行状況 n	0x0000	0x0000	…	0x0010

上記のように，走行状況の条件判定を共通化し条件をテーブルで管理すると，経路複雑度の低減によりソフトウェア構造は改善するが，一方で条件テーブルの管理において各条件ビットの意味を調査する必要がある．そのため，走行状況の追加や条件の変更が発生すると，作業工数の増加や条件の設定ミスによる不具合を作りこむ可能性があり，保守性に優れていない．

そこで，判定条件テーブルをソースコードではなく，Excel 上で管理することとした．Excel シート上では，1 つの走行状況に対して表 4-2 で示した入力パラメータ分類毎の判定条件を選択する．この際，判定条件をビットでなく自然言語で定義することで，可読性の向上を図った．判定条件管理表の例を図 4-3 に示す．

	1 運転者キャンセル	2 渋滞キャンセル	3 舵角キャンセル	4 先行車右左折
自車速度	–	–	20[km/h]以下	–
先行車速度	–	–	–	–
自車加速度	–	–	–	-1[m/s]以上
先行車加速度	–	–	–	–
車間距離	–	–	–	25[m]未満
相対速度	–	–	–	-5[km/h]未満
舵角	–	–	70[deg]以上	1.5[deg]以上
スイッチ	スイッチON	–	–	–
タイマー	–	再始動後5秒未満	–	–
その他	–	–	–	先行車が0.3[m]右へ

図 4-3 判定条件管理表の例

また，自然言語からソースコードへの変換を簡略化するため，ソースコードを自動生成する機能を実装した．自動生成コードの例を図 4-4 に示す．

```
/* アイドルストップ判定条件 */
const is_permit_param mIS_COND_TBL[] =
{
  {
    iIS_CANCEL_USER_CANCEL_MODE,   //走行状況              運転者キャンセル
    0,                             //自車速度
    0,                             //先行車速度
    0,                             //自車加速度
    0,                             //先行車加速度
    0,                             //車間距離
    0,                             //相対速度
    0,                             //舵角
    iCOND_SW_CANCEL_MODE_ON,       //スイッチ              キャンセルモードがON
    0,                             //タイマー
    0                              //その他
  },
  {
    iIS_CANCEL_SCENE_TRAFIC,       //走行状況              渋滞キャンセル
    0,                             //自車速度
    0,                             //先行車速度
    0,                             //自車加速度
    0,                             //先行車加速度
    0,                             //車間距離
    0,                             //相対速度
    0,                             //舵角
    0,                             //スイッチ
    iCOND_TIMER_RESTART_5SEC_OFF,  //タイマー              再始動後5秒が経過していない
    0                              //その他
  }
}
```

図 4-4 自動生成コードの例

4.2 対策内容の評価

ソフトウェア構造の見直しと再構築の結果を評価するため，再度経路複雑度とネストレベルの測定を実施した．再構築後の経路複雑度の測定結果を図 4-5，ネストレベルの測定結果を図 4-6 に示す．

測定の結果，経路複雑度の最大値が 58 から 10 へ，ネストレベルの最大値が 17 から 3 へ低下した．この結果から，ソフトウェア構造の再構築により保守性と可読性の向上に成功したと判断できる．

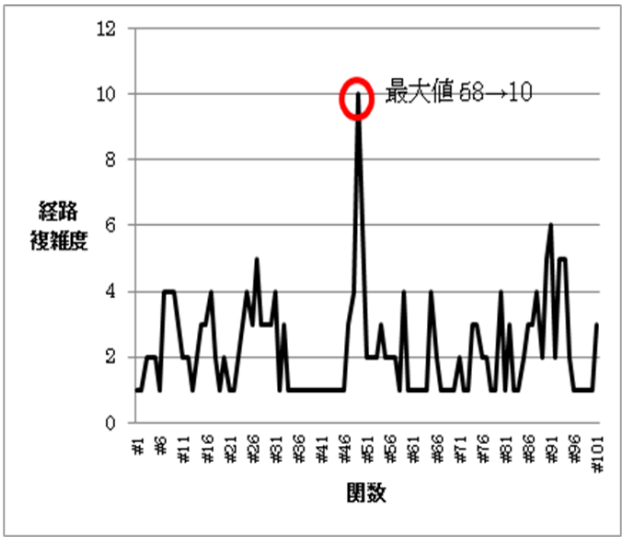

図 4-5 再構築後の経路複雑度

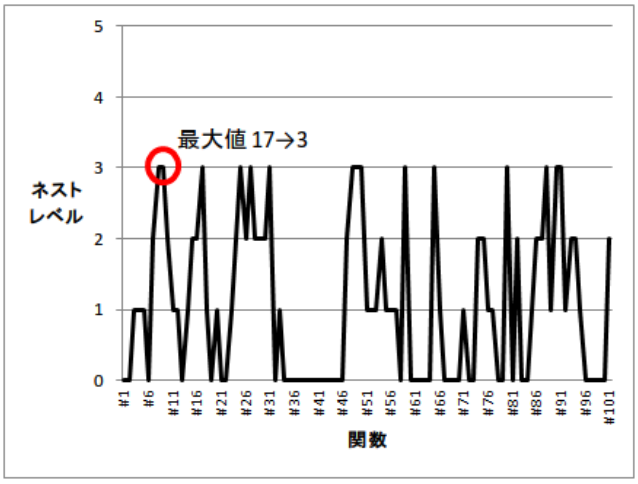

図 4-6 再構築後のネストレベル

また，入力パラメータの参照箇所を局所化することで結合度の高い処理を局所化できたため，移植性を向上させることに成功した．

対策を実施した再構築後の処理フローを図 4-7 に示す．再構築によりアイドルストップ許可判定が分割され，③走行状況判定と④優先度判定が機能として独立した．

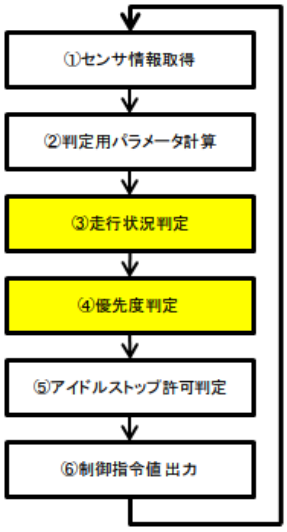

図 4-7 再構築後の処理フロー

5. おわりに

本論文では，外界認識センサを用いたアイドルストップ制御ソフトにおける構造を再構築する手法について示した．本手法はセンサ情報に対する閾値判定，及び多数のセンサ情報を組み合わせた状態判定を整理・構造化することで，ソフトウェア構造に起因する保守性の低下および可読性の低下という問題に対する有効な対策である．

参考文献

[1] 国土交通省　-　現在実現している運転支援システムの概要
http://www.mlit.go.jp/road/ir/ir-council/autopilot/pdf/9.pdf
[2] THOMAS J. McCABE　-　A Complexity Measure
http://www.literateprogramming.com/mccabe.pdf
[3] 東陽テクニカ – メトリックの概要
http://www.toyo.co.jp/ss/qac/product_summary4.html