ソフトウェア設計
プログラム実装

初版：2014/12/10

改訂：2015/12/16

本資料の目的

- この資料は「陸橋修復競技」の修復機のソフトウェアを例に、ソフトウェア設計の手順を示すものである。
- 修復機の動作は「動画」を参照せよ。
- なお、ここに示すソフトウェア設計例では修復位置の確認、完走機との通信は考慮していない。

ソフトウェア設計

- ある程度以上大きな(複雑な)プログラムを作る場合は設計を行うことが重要
  - 手戻りの少ない開発が出来る。
  - 複数人で開発が可能になる。
- ソフトウェア設計の手順
  1. 動作シナリオを明確にする。
  2. 機能を列挙する。
  3. 機能間の関係性を整理する。
  4. 関数の仕様を決める。
  5. ファイル分割を決める。
  6. マルチタスク化について検討する。
  7. 機能の実現方法を検討する。

動作シナリオを明確にする

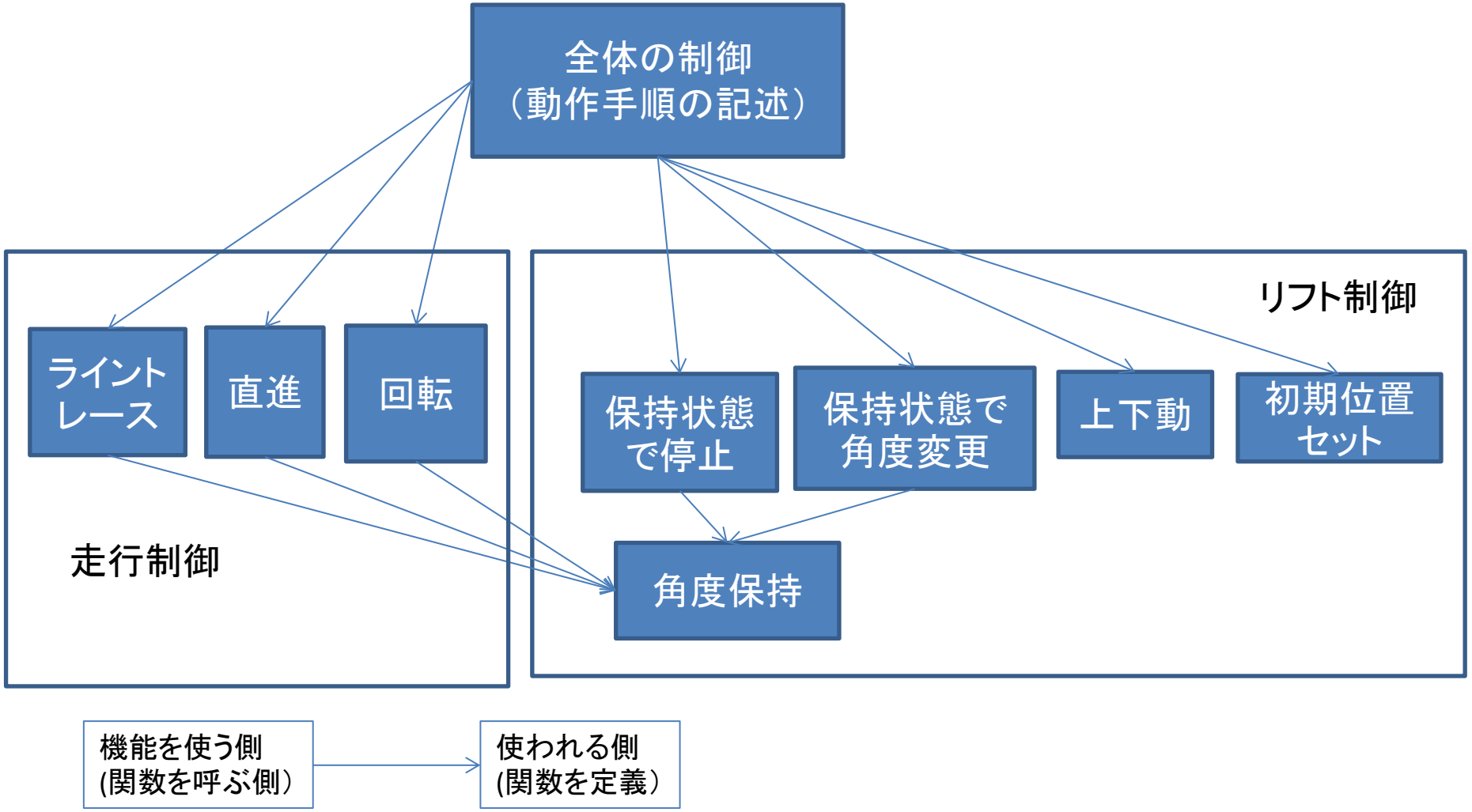
1. スタートする。
2. 修復材置場の分岐位置までライントレースして進む。（左エッジ）
3. 修復材置き場のガイドラインが見つかるまで右回転する。
4. リフトを降ろし、修復材の近くまでライントレースして進む。（左エッジ）
5. リフトを上げ修復材を持ち上げて、元の位置まで下がる。
6. コースのラインが見つかるまで左回転する。
7. 修復場所の分岐位置までライントレースして進む。
8. 修復場所のガイドラインが見つかるまで右回転する。
9. 修復位置の近くまでライントレースして進む。
10. リフトを下げ修復材を橋脚上に置く。
11. もとの位置まで下がる。
12. リフトを初期位置に戻す。

（それぞれの動作の区切りには、若干の停止時間を入れる）

機能を列挙する

- 指定距離、ライントレース走行する。
- 指定距離、直進走行する。
- ラインを発見するまで回転する。
- リフト
  - 初期位置にセットする。
  - 指定角度まで下げる、上げる。
  - 指定角度を保持する。
  - 修復材を保持した状態で角度を変える。
  - 修復材を保持した状態で一定時間停止する。

機能間の関係（構造）


全体の制御
（動作手順の記述）
ライントレース
直進
回転
走行制御
リフト制御
保持状態で停止
保持状態で角度変更
上下動
初期位置セット
角度保持
機能を使う側
(関数を呼ぶ側）
使われる側
(関数を定義）

関数の仕様を決める（1）

- ライントレース

関数名と引数	戻り値	説明
void line_trace(int distance, int edge, int hold_value)	なし	distance:走行距離(cm) edge: 0:左エッジ、1:右エッジ hold_value:リフト保持角(度）

- 直進

関数名と引数	戻り値	説明
void straight(int distance, int hole_value)	なし	distance: 走行距離(cm) hold_value:リフト保持角(度）

- 回転

関数名と引数	戻り値	説明
void rotate(int angle, int hold_value)	なし	angle: 回転角度（度） hold_value:リフト保持角(度）
void rotate_light(int angle, int hold_value)	なし	ラインが見つかるまで回転する。 angle: 想定回転角（度） hold_value:リフト保持角(度）

関数の仕様を決める(2)

- リフト制御

関数名と引数	戻り値	説明
void lift_init()	なし	リフトを初期位置にセットする
void lint_down(int angle)	なし	angle[deg] の角度までリフトを下げる
void lift_up(int angle)	なし	angle[deg] の角度までリフトを上げる
void lift_move_hold(int angle)	なし	修復材を保持した状態で、angle[deg] の角度までリフト角を変更する
void lift_hold(int angle)	なし	angle の角度を保持する
void lift_hold_time(int angle, int time)	なし	time [msec] 時間、angle[deg] の角度を保持する

ファイル分割

- 方針
  - 大きな機能単位でファイルを分割する。

- ソースファイル

ファイル名	説明
pilot.c	全体の制御の実装（動作手順の記述）
line_trace.c	line_trace 関数の実装
straight.c	straight 関数の実装
rotate.c	rotate* 関数の実装
lift.c	lift_* 関数の実装

ヘッダファイル

- 作成方針
  - マクロの定義、関数宣言をヘッダファイルとして分割する。
- ヘッダファイル

ファイル名	説明
myrobot.h	入出力ポート、白黒値、タイヤ半径、タイヤ間距離のマクロの定義
run.h	line_trace, straight, rotate* 関数の extern 宣言
lift.h	lift_* 関数の extern 宣言

- Include の仕方

ソースファイル	Include するヘッダファイル
pilot.c	myrobot.h run.h lift.h
line_trace.c straight.c rotate.c	myrobot.h lift.h
lift.c	myrobot.h

マルチタスク化・周期タスク化の検討

- この動作シナリオには、動作の並行性（並行して行う動作）がないので、マルチタスク化は行わない。
  - Bluetooth 通信を走行制御と並行して行う場合は、マルチタスク化する必要がある。

- リアルタイム性が要求される動作はないので、周期タスク化（リアルタイムタスク化）の必要はない。

機能の実現方法の検討

- ライントレース、直進、回転走行
  - PD制御を用いる。
- リフト角の保持
  - PI制御を用いる。
- リフトの初期位置セット
  - リフト上げていって、止まったところ(リフト角がその前のループと変わらなくなったところ)とする。
- ガイドラインの検知
  - 想定回転角の半分以上回ったところで、黒判定の閾値を超えたらライン検知とする。

PID制御

- PID制御

$u(k) = Kp\ Er(k) + Ki \Sigma Er(k) + Kd\ (Er(k)-Er(k-1))$

$Er(k) = Y(k) - Ystd$:kスッテプ目の誤差

- ラインとレース　PD制御

$Er(k) = L(k) - Lref$

L(k):光センサの値、Lref: 白黒中間値

$Vdif(k) = Kp\ Er(k) + Kd\ (Er(k)-Er(k-1))$

$Vr(k) = Vstd + Vdif(k)$　右モータ指令値

$Vl(k) = Vstd - Vdif(k)$　左モータ指令値

- リフト角度保持　PI制御

$Er(k) = A(k) - Aref$

A(k):現在の角度、 Aref:目標角度

$V(k) = Kp\ Er(k) + Ki \Sigma Er(k)$

実装に当たって(1)

- テストファースト
  - 実装(コーディング)を行なったら(他と組み合わせる前に)まずテスト
  - ビッグバンテスト(＝単体試験なしで統合試験を行うこと)は禁じ手!

1. 関数機能単位で動作試験(単体試験)を行う。
   - 機能試験を行うテストプログラムを用意する。
2. テストを終えた機能を組み合わせての動作試験(統合試験)を実施する。

実装にあたって（2）

- 可読性の高いプログラム（他人が見ても分かるプログラム）を作成する。
  - 意味が分かる変数名、関数名とする。
  - 適切なコメントを入れる。
  - 適切な空行を入れる。
  - 制御構造に応じたインデント（TAB）を入れる。