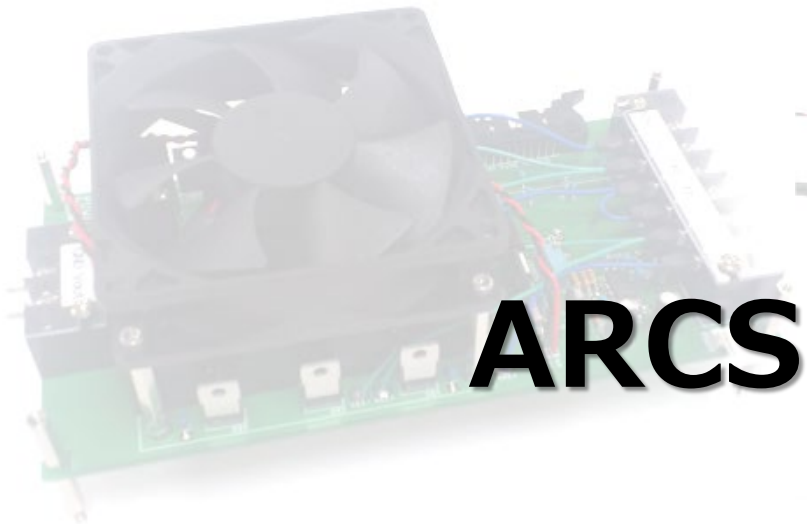
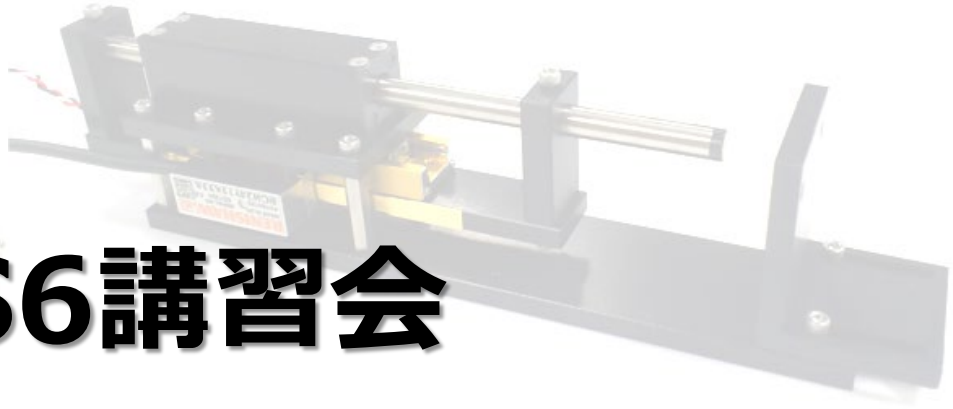
国立大学法人
長岡技術科学大学
Nagaoka University of Technology

MC LAB Motion Control Lab

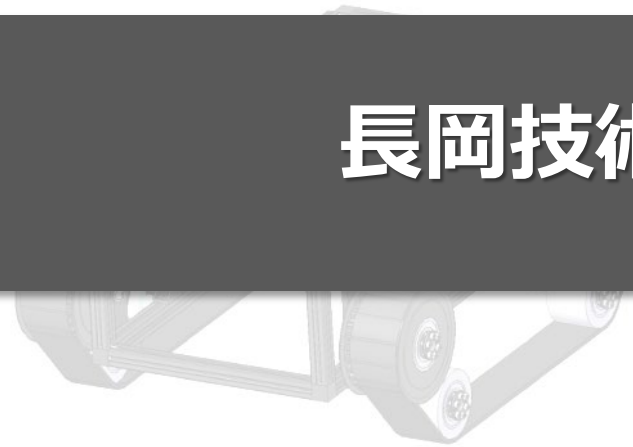
ARCS6レクチャー資料 REV.21032617





ARCS6講習会

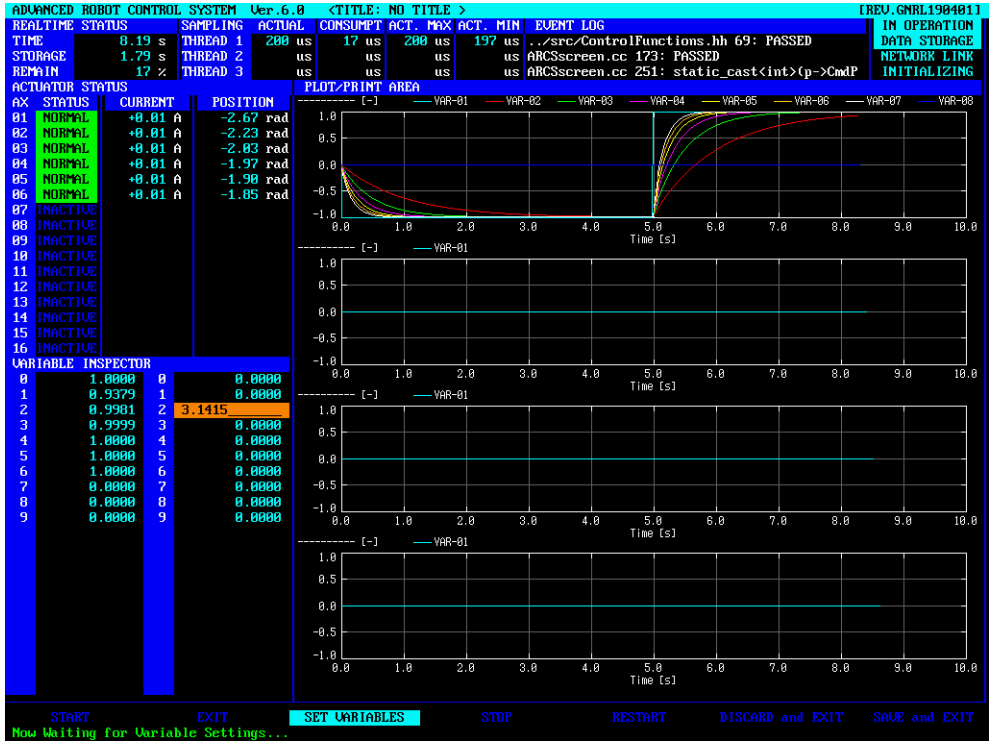
長岡技術科学大学 横倉 勇希



ARCS

ARCSって何？

- Advanced Robot Control System V6, ARCS6
- 通称「えーあーるしーえす」or「あーくす」って言います(好きな方で可)
- Linuxで低層レイヤでのロボット制御をするためのフレームワーク
- 2006年頃から開発スタート、2021年現在も機能追加中
- 言語はC++17、Linux上のCUIで動きます



オフィシャルロゴ

何ができるの？

- マルチリアルタイムスレッドの生成・周期実行・破棄
- 測定データの記憶と出力(CSV形式及びTAB区切りDAT形式対応)
- アクチュエータパラメータ(位置，電流，推力等)のリアルタイム表示と波形描画
- 任意変数値のリアルタイム表示とリアルタイム波形描画
- オンライン変数書き換え機能
- 一時停止＆再始動機能
- 実際の制御周期の計測と標準偏差，最大値，最小値の計算と表示
- 実際の計算消費時間の計測と表示
- D/Aコンバータ，A/Dコンバータ，エンコーダカウンタ，シリアル通信ボード等々とのインターフェースクラス
- 6軸力覚センサ用シリアル通信インターフェース
- MECHATROLINK III 通信対応(内部版のみ)，EtherCAT通信対応
- デバイスドライバ，カーネルモジュール開発支援機能
- モーションコントロールクラス(積分器，擬似微分器，PI/PD/PID/I-PD等々の制御器，位相補償器，各種オブザーバ，各種フィルタ，レギュレータ等々)
- 様々な機能を持ったクラス(リングバッファ，統計処理，リミッタ，信号発生器，UDP送受信器，メルセンヌ・ツイスタ…等々)
- リアルタイムデバッグプリント機能，イベントログ機能，条件指定緊急停止機能
- 行列＆ベクトルの加減乗演算，冪乗，LU/QR/コレスキー分解，SVD，逆行列，複素数行列，constexpr行列…等々の行列計算
- 連続系状態方程式のA行列とB行列の離散化，任意の連続系伝達関数の応答計算
- 周波数特性測定のためのFRAクラス
- 単純/単層パーセプトロン，順伝播ニューラルネットワーク

開発コンセプトのウリ： 短い制御周期でのモーションコントロールに強い！

なぜ独自で開発しているの？

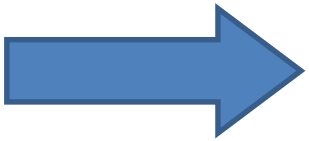
- ROSとの違い
  インバータのゲートを叩くレベルでロボットを
  動かしている我々パワー研からするとROSは上位層用。
  なので，ROSで上位層を，ARCSで下位層を作る
  という住み分けが正解。

- MATLAB/Simulink/Coderとの違い
  裏で何をやっているかがよく分からない。
  ライセンス料かかる。

- モーションコントロールに特化した
  ロボット制御フレームワークが欲しい。

なぜ普通のPCを使うの？

- 制御用コンピュータ



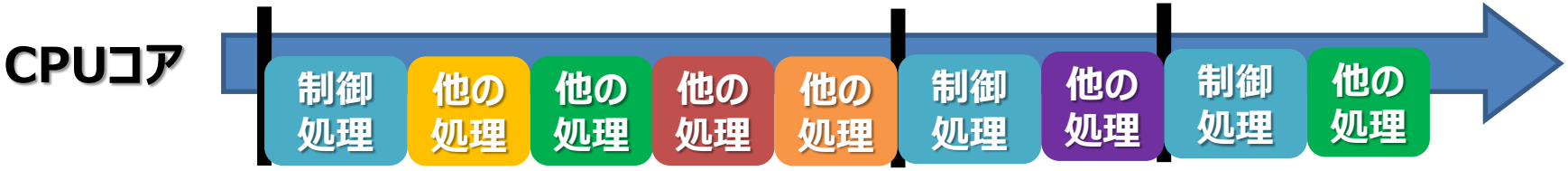
普通のDSP　　普通のパソコン

実践演習では所謂ベアメタル的なDSP・マイコンではなく，
OSが搭載された普通のパソコンを制御に使う。その利点は下記。

- ほぼ無尽蔵な記憶容量 → 実験データの保存に余裕
- ディスプレイ・キーボード・マウスが使える → 使いやすいUI，操作しやすい
- リソース管理をOSに任せられる → 簡単
- ネットワークが使える → ファイル共有・遠隔化が簡単
- クロスコンパイル不要 → 開発環境で悩まなくて良い
- 最新の超高速なCPUが使える → 計算がめっちゃ速い
- 計算性能の割に低価格 → 懐にやさしい

リアルタイム制御

- 非リアルタイムOSの場合（通常のLinuxやWindows）



現実： 制御アルゴリズムの実行周期がバラバラ。 大問題。

リアルタイム性の重要性と課題

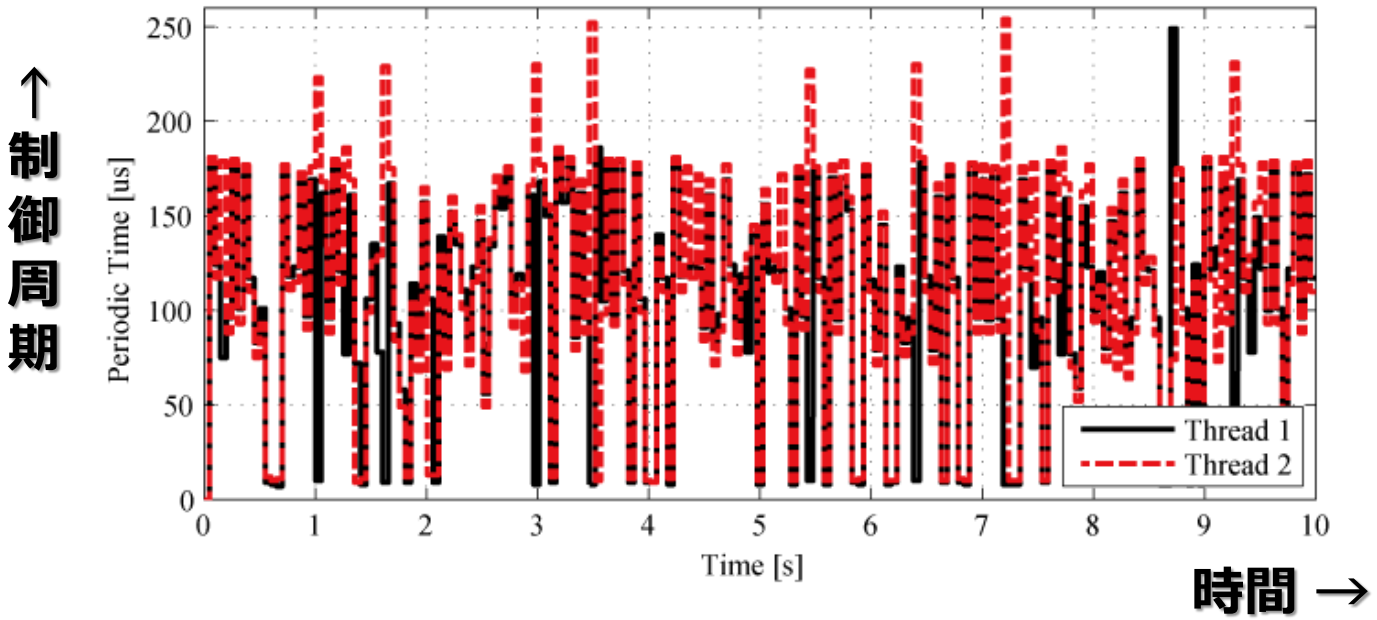
- リアルタイムOSの場合

理想： 何が何でも一定の周期で制御アルゴリズムが実行される
（そうじゃないと，厳密に離散化した方程式が使えなくなる）

Linuxをリアルタイム化するのはかなり辛い。（実際の経験）

リアルタイム制御

- 非リアルタイムOSをそのまま使った場合の実行例



制御周期（制御アルゴリズムの実行周期）が
100 μs になるようにLinuxのコードをC++で作成。

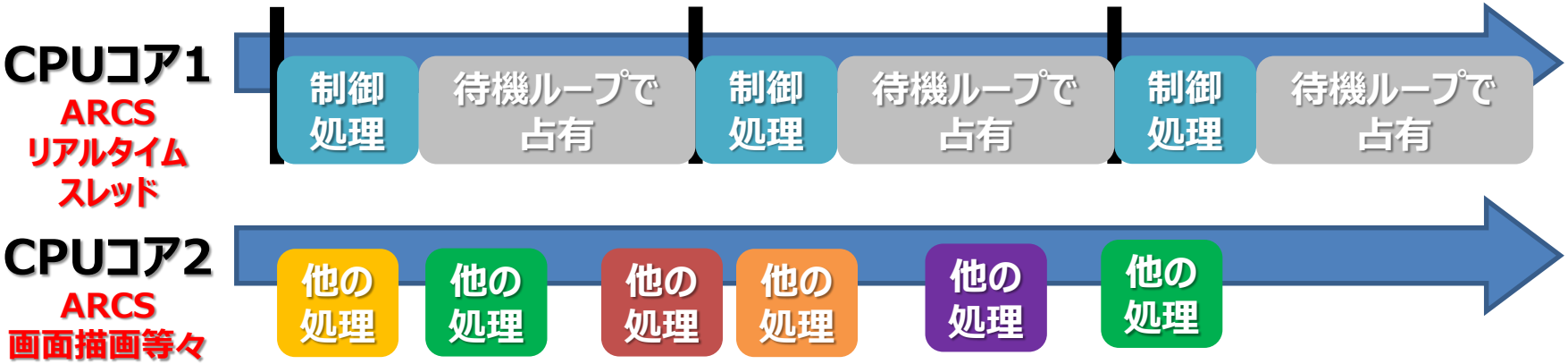
にもかかわらず，ぜんぜん 100 μs にならない。バラバラ。

リアルタイム制御

- リアルタイム性を確保する方法
  - 非リアルタイムOSの場合（通常のLinuxやWindows）

現実： 制御アルゴリズムの実行周期がバラバラ。 大問題。

  - 非リアルタイムOS＋ARCSの場合（通常のLinux＋ARCS）



CPUコアを１つ占有してしまえばいい。 他には使わせない。
最近のPCはマルチコアでディスクアクセスも高速なのでほとんど問題ない。

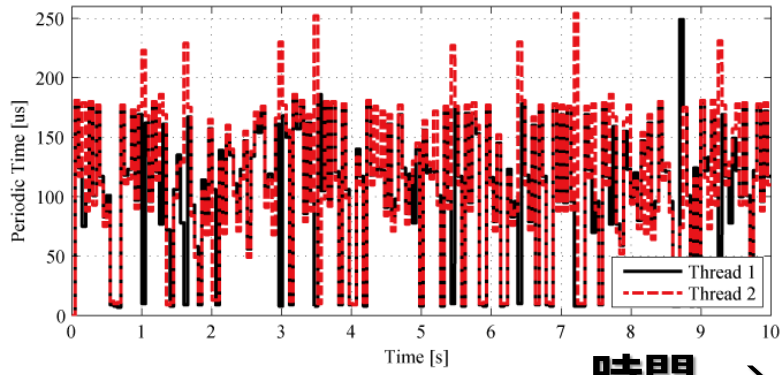
リアルタイム制御

- ハードウェア選定時に気をつけること
- マルチコアCPU（できれば４コ以上）
- HyperThreadingは無くて良い（あっても無効にする）
- HDDは避けたほうが無難
- SSD、M.2 SSDだとなお良い
- PCI Express x1を１つ以上
  積んでいること

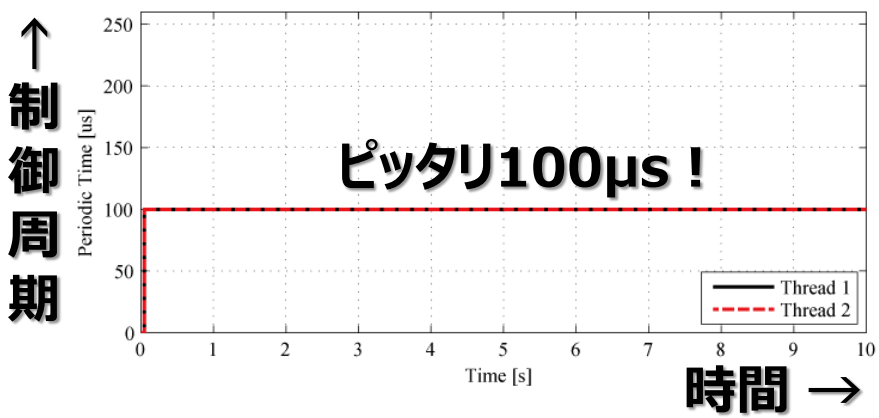
リアルタイム制御

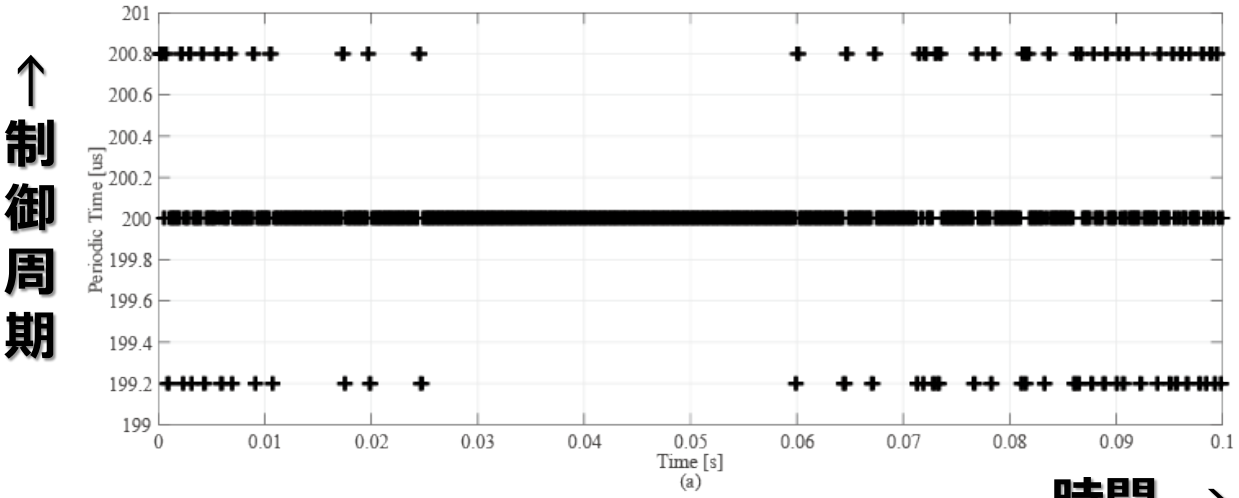
- 非リアルタイムOS上でARCSを使った場合の実行例

そのままの場合



ARCS使用時





制御周期200μsに設定した場合の拡大図

Linux環境の構築

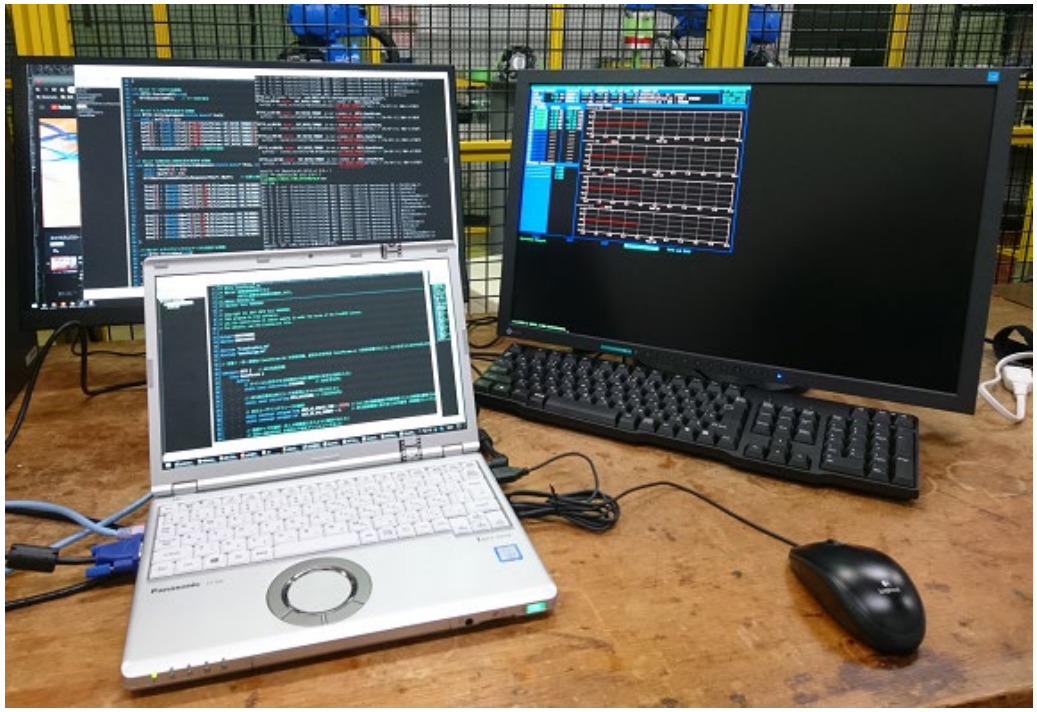
- ARCSを使う前にLinuxの環境を整えていかないといけません
- ARCSはLinuxディストリビューションを選びませんが、
  2021年時点では下記をオススメしています（情報が多いので）

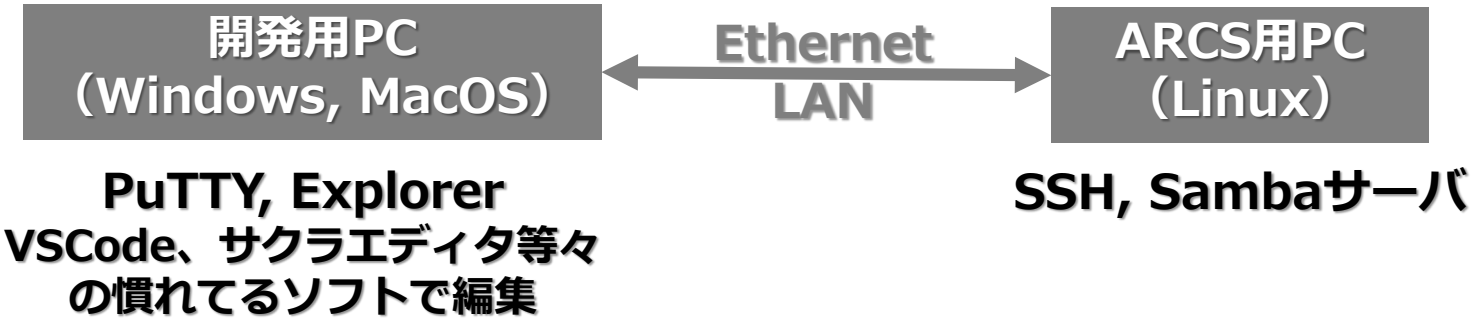
- ただし、CentOSは廃止騒動があってAlmaLinuxに移行するかも？
- 上記をインストールする際には、
  余計なモノが入らないように、Minimal版を選択
- ARCS6の動作には ncurses, libncurses
  と libpng の外部ライブラリが必要なので、
  yum or apt コマンドでインストール

- SSHとSambaがあると開発には便利（次のスライドで説明）

開発環境の整備

- SSHとSamba





まずはダウンロード

- GitHubからの場合
  https://github.com/sidewarehouse/ARCS-PUBLIC.git

- ZIPファイルを直接ダウンロードする場合
  http://www.sidewarehouse.net/arcs6/index.html

- ARCSは有償？無償？　→　無償です！

- ライセンス形態は？　→　BSDライセンスです！
  IT用語辞典から転載
  「BSDライセンスでは、著作権の表示と免責条項さえ明記しておけば、再利用も再配布も自由となる。著作権・免責の表示さえしっかりと行っていれば、BSDライセンスを利用し、改変し、再配布することもできる。商用のプログラムに取り入れることも自由である。」

- GPLライセンスで汚染されてない？　→　完全にクリーンです！

とりあえずコンパイル

- **四の五の言わずにとりあえずコンパイルしてみましょう**

- **サンプルコードがあるので、そこまで移動**
  - **cd ARCS6/robot/sample/00_空の基本コード**
    - ✓ダウンロードしたファイルを展開した先で上記を入力
    - ✓「cd」は”change directory” Linuxコマンドは各自で自習のこと

- **そしてコンパイル**
  - **make**
    - ✓Makeコマンドを打つと、
      同じディレクトリ内にあるMakefileが走り始める

- **1回目のコンパイルは時間が少し掛かります**

とりあえず実行

- **コンパイルが完了したら実行してみます**

- **下記を入力して実行可能ファイルを起動します**

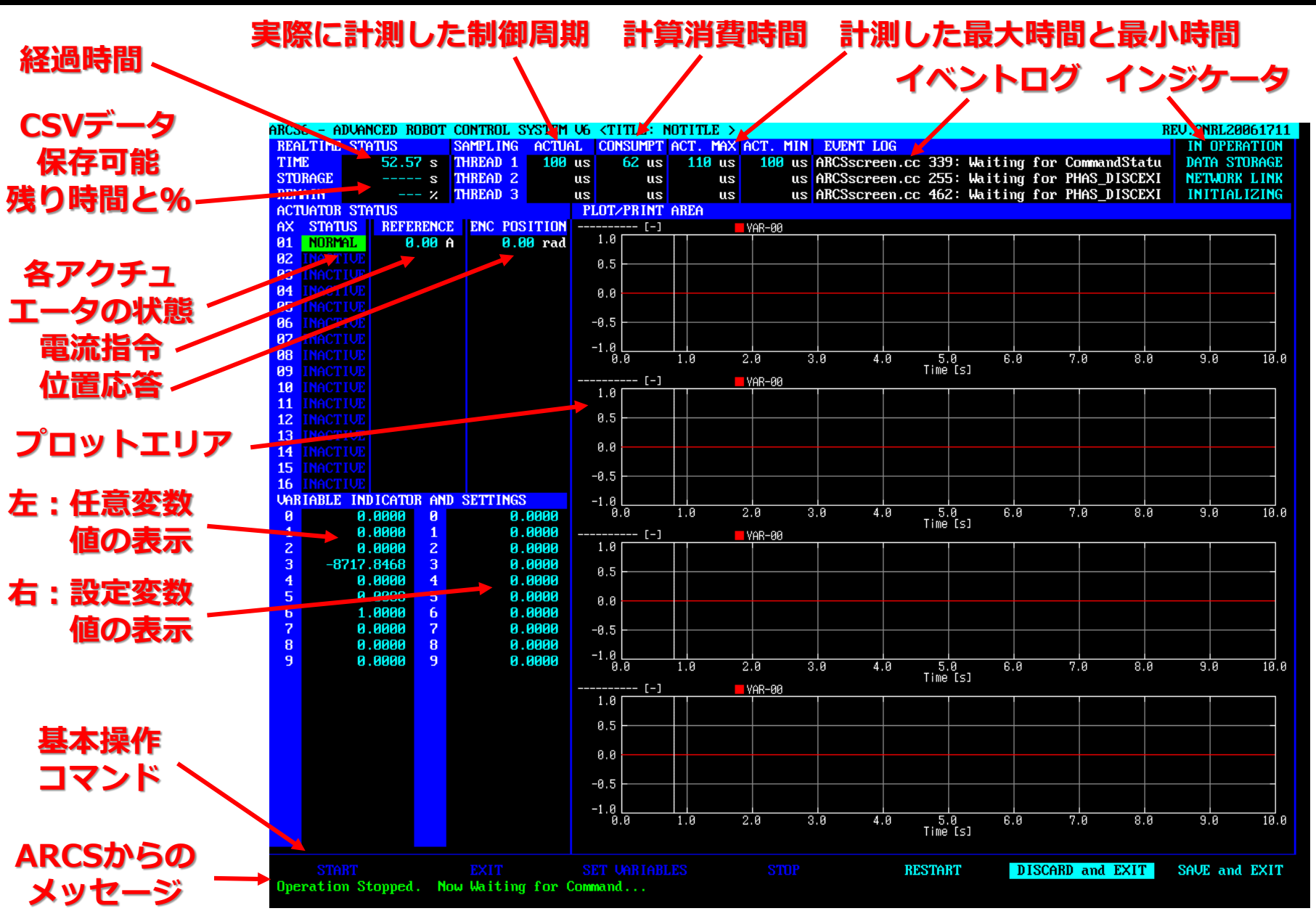
➢**./ARCS**

✓「./」の意味は今いるこのディレクトリ

✓「ARCS」は実行可能ファイル、コンパイルすると自動生成される

- **すると画面が出ます**

画面構成


経過時間
実際に計測した制御周期
計算消費時間
計測した最大時間と最小時間
イベントログ
インジケータ
CSVデータ
保存可能
残り時間と%
各アクチュエータの状態
電流指令
位置応答
プロットエリア
左：任意変数
値の表示
右：設定変数
値の表示
基本操作
コマンド
ARCSからの
メッセージ
ARCS - ADVANCED ROBOT CONTROL SYSTEM V6 <TITLE: NOTITLE >
REV.NRL20061711
REALTIME STATUS
SAMPLING
ACTUAL
CONSUMPT
ACT. MAX
ACT. MIN
EVENT LOG
TIME 52.57 s
STORAGE ----- s
THREAD 1 100 us 62 us 110 us 100 us
THREAD 2
THREAD 3
ARCSscreen.cc 339: Waiting for CommandStatu
ARCSscreen.cc 255: Waiting for PHAS_DISCEXI
ARCSscreen.cc 462: Waiting for PHAS_DISCEXI
IN OPERATION
DATA STORAGE
NETWORK LINK
INITIALIZING
ACTUATOR STATUS
AX STATUS REFERENCE ENC POSITION
01 NORMAL 0.00 A 0.00 rad
02 INACTIVE
03 INACTIVE
04 INACTIVE
05 INACTIVE
06 INACTIVE
07 INACTIVE
08 INACTIVE
09 INACTIVE
10 INACTIVE
11 INACTIVE
12 INACTIVE
13 INACTIVE
14 INACTIVE
15 INACTIVE
16 INACTIVE
PLOT/PRINT AREA
VAR-00
Time [s]
VARIABLE INDICATOR AND SETTINGS
0 0.0000 0 0.0000
1 0.0000 1 0.0000
2 0.0000 2 0.0000
3 -8717.8468 3 0.0000
4 0.0000 4 0.0000
5 0.0000 5 0.0000
6 1.0000 6 0.0000
7 0.0000 7 0.0000
8 0.0000 8 0.0000
9 0.0000 9 0.0000
START
EXIT
SET VARIABLES
STOP
RESTART
DISCARD and EXIT
SAVE and EXIT
Operation Stopped. Now Waiting for Command...

基本操作

- ./ARCSで起動すると自動的に初期化動作が始まる
- 使っているインターフェースの仕様（例えばMechatoroLinkなど）によっては時間が少しかかることもある

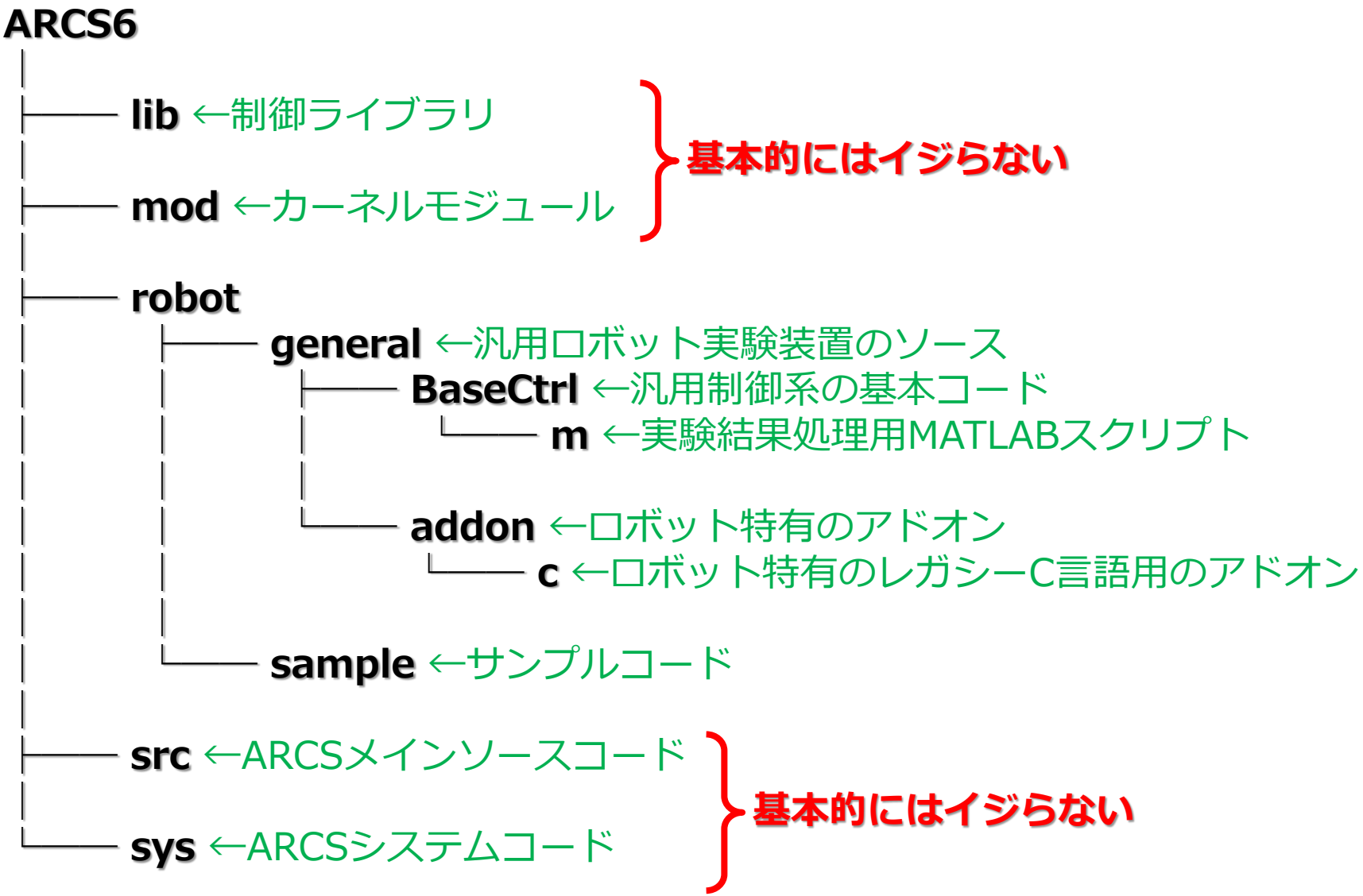
- 操作可能になると「START」が光る
- キーボード上の矢印キー[←]/[→]を押すと左右に移動できる
- 何もしないで終了するときは「DISCARD and EXIT」を選択して[Enter]キーを押すとそのまま終了

- 「START」を選択して[Enter]キーを押すと制御開始
- 「STOP」で制御停止

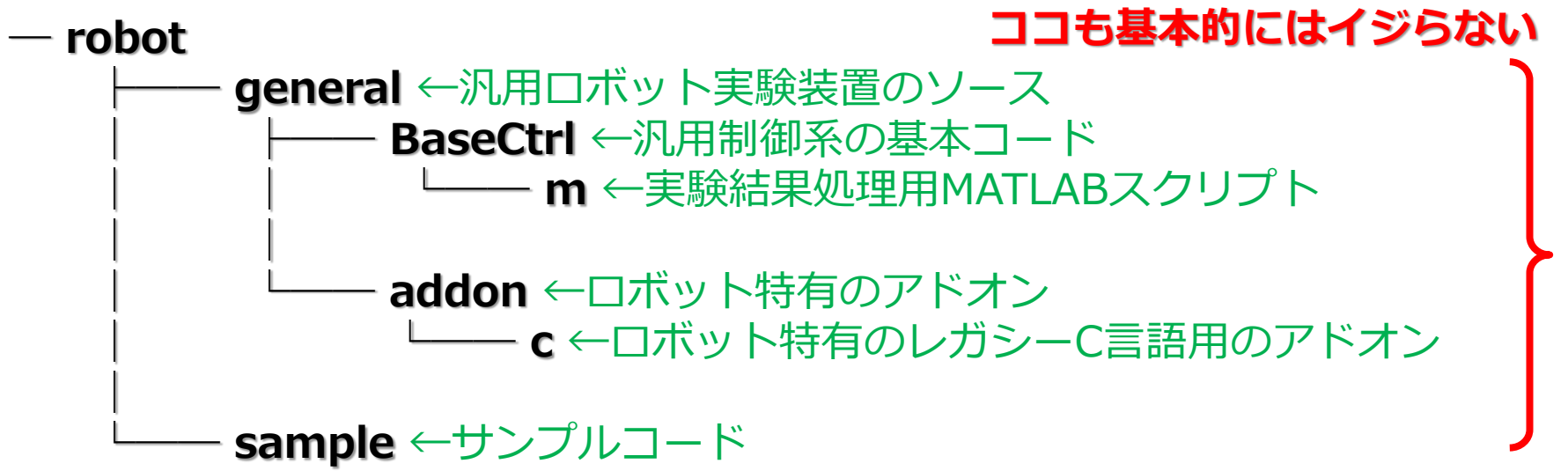
- 「SAVE and EXIT」でCSVファイルに実験データを保存して終了

- 「SET VARIABLES」については後々のスライドで説明

ディレクトリ構成

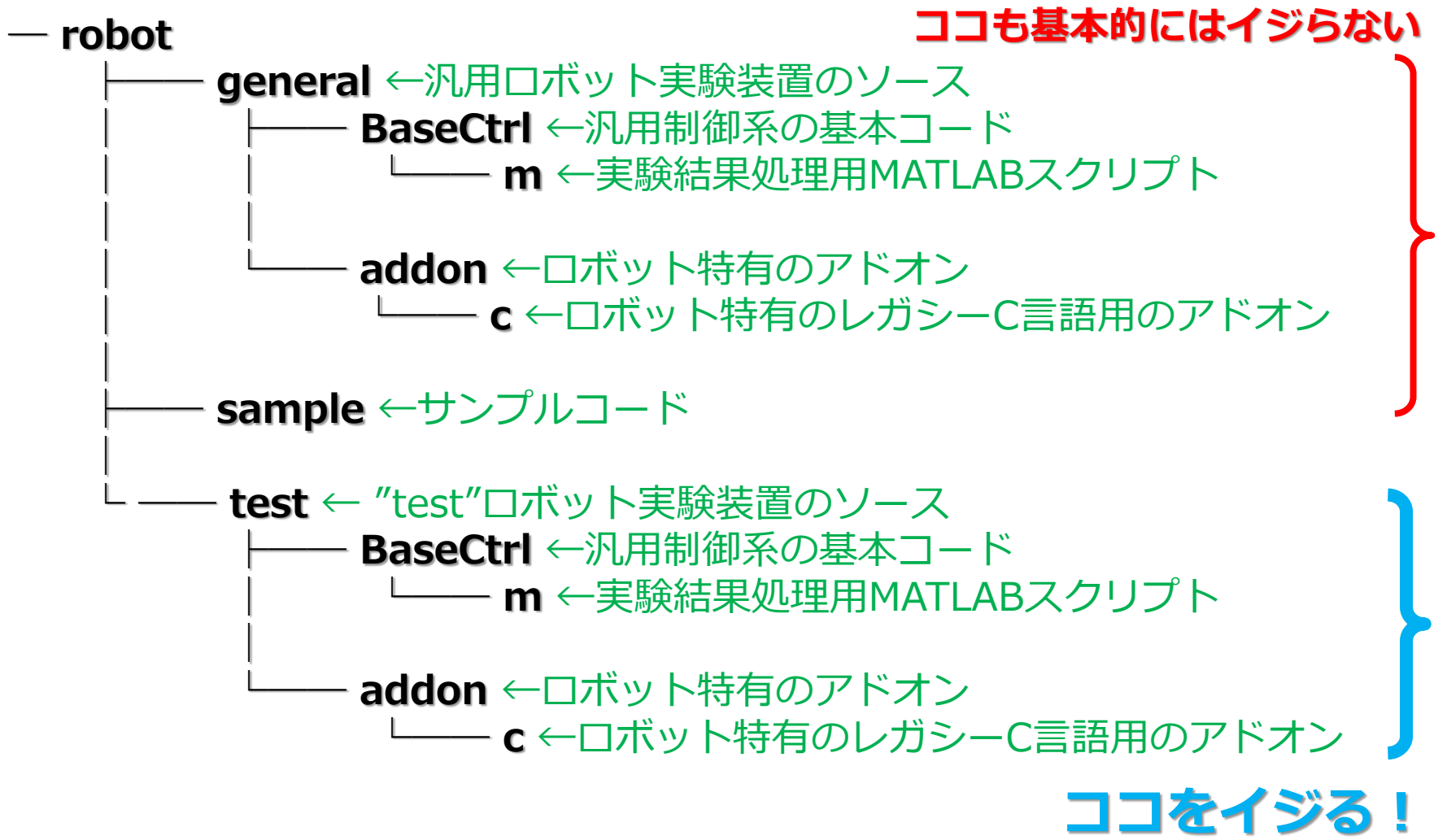

ARCS6
lib ←制御ライブラリ
mod ←カーネルモジュール
基本的にはイジらない
robot
general ←汎用ロボット実験装置のソース
BaseCtrl ←汎用制御系の基本コード
m ←実験結果処理用MATLABスクリプト
addon ←ロボット特有のアドオン
c ←ロボット特有のレガシーC言語用のアドオン
sample ←サンプルコード
src ←ARCSメインソースコード
sys ←ARCSシステムコード
基本的にはイジらない

ディレクトリ構成



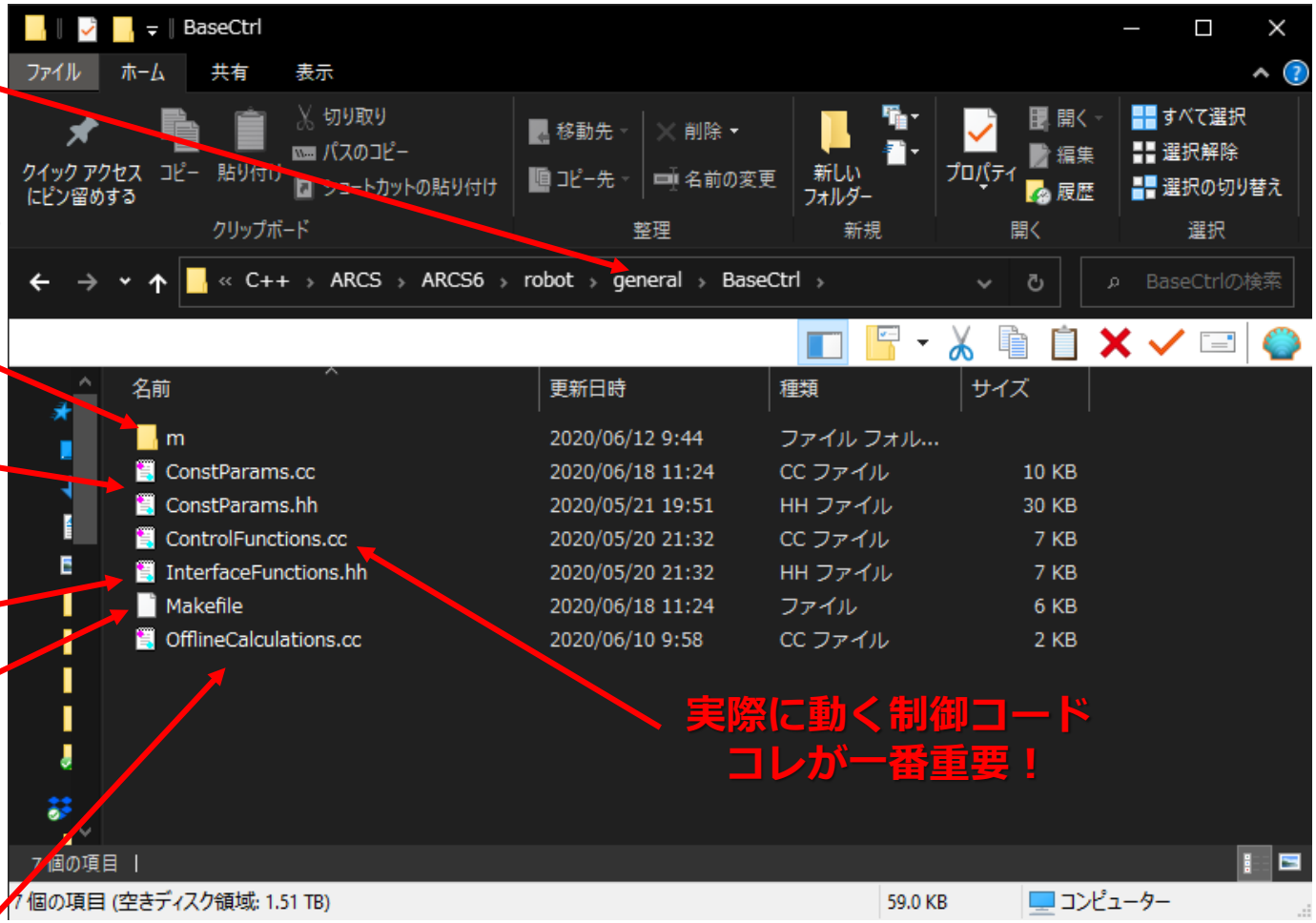
ここに
新しいロボット用の作業ディレクトリを追加してみましょう！

ディレクトリ構成


ココも基本的にはイジらない
robot
general ←汎用ロボット実験装置のソース
BaseCtrl ←汎用制御系の基本コード
m ←実験結果処理用MATLABスクリプト
addon ←ロボット特有のアドオン
c ←ロボット特有のレガシーC言語用のアドオン
sample ←サンプルコード
test ← "test"ロボット実験装置のソース
BaseCtrl ←汎用制御系の基本コード
m ←実験結果処理用MATLABスクリプト
addon ←ロボット特有のアドオン
c ←ロボット特有のレガシーC言語用のアドオン
ココをイジる！

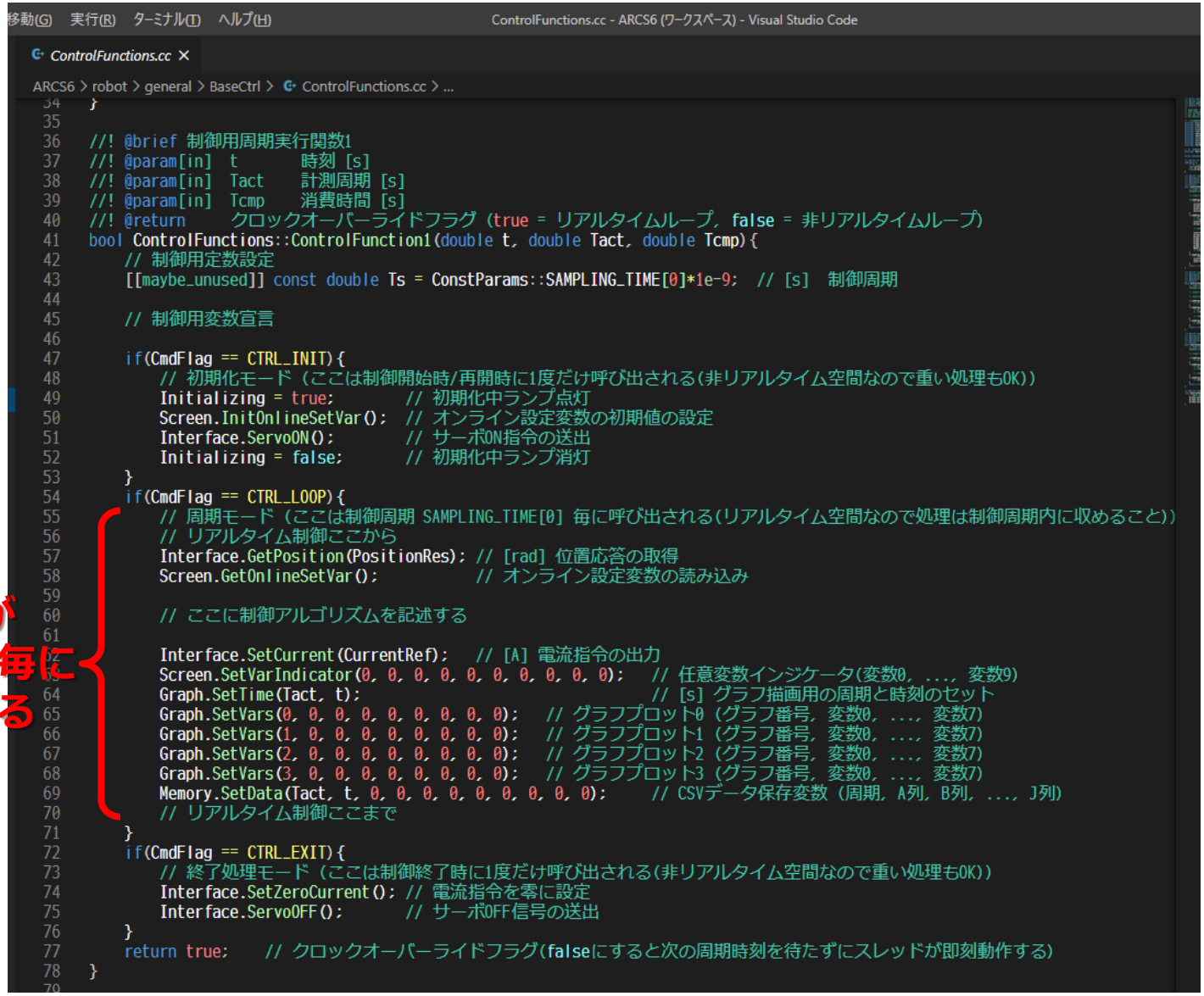
ディレクトリ構成

- Samba経由でアクセスすると



実際に動く制御コード

- **ControlFunctions.cc**



ココが
制御周期毎に
呼ばれる

実際に動く制御コード

- ControlFunctions.cc（拡大）

エンコーダから位置を貰う

やりたい制御をココに書く

```
    }
    if(CmdFlag == CTRL_LOOP){
        // 周期モード（ここは制御周期 SAMPLING_TIME[0] 毎に呼び出される(リアルタイム空間なので処理は制御周期内に
        // リアルタイム制御ここから
        Interface.GetPosition(PositionRes); // [rad] 位置応答の取得
        Screen.GetOnlineSetVar();           // オンライン設定変数の読み込み

        // ここに制御アルゴリズムを記述する

        Interface.SetCurrent(CurrentRef);   // [A] 電流指令の出力
        Screen.SetVarIndicator(0, 0, 0, 0, 0, 0, 0, 0, 0, 0);   // 任意変数インジケータ(変数0, ..., 変数9)
        Graph.SetTime(Tact, t);                                  // [s] グラフ描画用の周期と時刻のセット
        Graph.SetVars(0, 0, 0, 0, 0, 0, 0, 0, 0);   // グラフプロット0 (グラフ番号, 変数0, ..., 変数7)
        Graph.SetVars(1, 0, 0, 0, 0, 0, 0, 0, 0);   // グラフプロット1 (グラフ番号, 変数0, ..., 変数7)
        Graph.SetVars(2, 0, 0, 0, 0, 0, 0, 0, 0);   // グラフプロット2 (グラフ番号, 変数0, ..., 変数7)
        Graph.SetVars(3, 0, 0, 0, 0, 0, 0, 0, 0);   // グラフプロット3 (グラフ番号, 変数0, ..., 変数7)
        Memory.SetData(Tact, t, 0, 0, 0, 0, 0, 0, 0, 0, 0);     // CSVデータ保存変数 (周期, A列, B列, ..., J列)
        // リアルタイム制御ここまで
    }
    if(CmdFlag == CTRL_EXIT){
```

サーボアンプに電流指令を送る

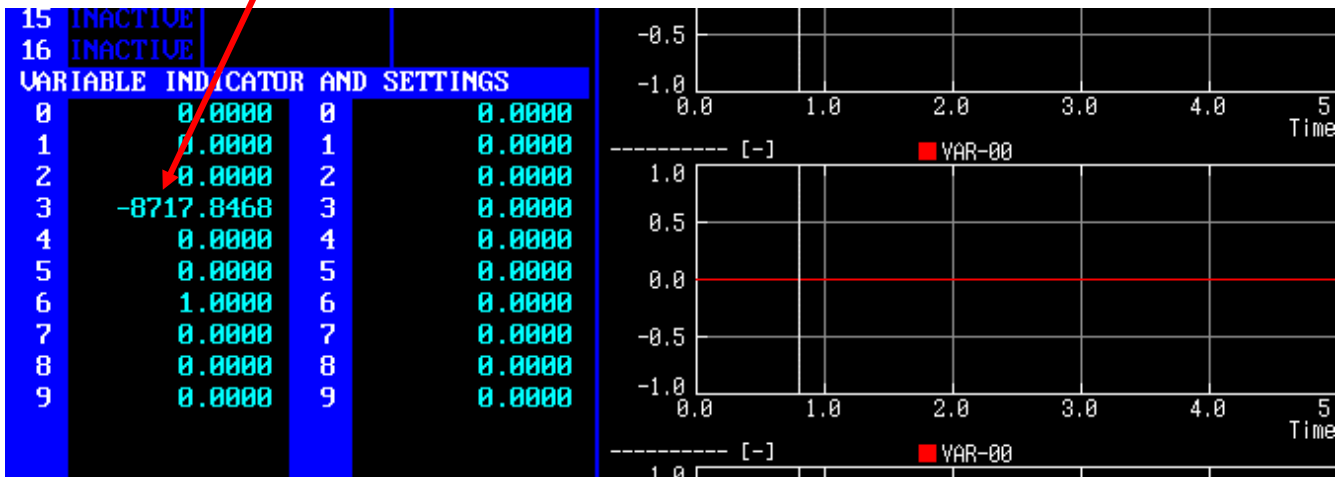
任意変数インジケータ

```
}
if(CmdFlag == CTRL_LOOP){
    // 周期モード（ここは制御周期 SAMPLING_TIME[0] 毎に呼び出される(リアルタイム空間なので処理は制御周期内に収
    // リアルタイム制御ここから
    Interface.GetPosition(PositionRes); // [rad] 位置応答の取得
    Screen.GetOnlineSetVar();           // オンライン設定変数の読み込み

    // ここに制御アルゴリズムを記述する

    Interface.SetCurrent(CurrentRef);   // [A] 電流指令の出力
    Screen.SetVarIndicator(0, 0, 0, hoge, 0, 0, 0, 0, 0, 0);    // 任意変数インジケータ(変数0, ..., 変数9)
    Graph.SetTime(Tact, t);                                     // [s] グラフ描画用の周期と時刻のセット
    Graph.SetVars(0, 0, 0, 0, 0, 0, 0, 0, 0);   // グラフプロット0 (グラフ番号, 変数0, ..., 変数7)
    Graph.SetVars(1, 0, 0, 0, 0, 0, 0, 0, 0);   // グラフプロット1 (グラフ番号, 変数0, ..., 変数7)
    Graph.SetVars(2, 0, 0, 0, 0, 0, 0, 0, 0);   // グラフプロット2 (グラフ番号, 変数0, ..., 変数7)
    Graph.SetVars(3, 0, 0, 0, 0, 0, 0, 0, 0);   // グラフプロット3 (グラフ番号, 変数0, ..., 変数7)
    Memory.SetData(Tact, t, 0, 0, 0, 0, 0, 0, 0, 0, 0);     // CSVデータ保存変数 (周期, A列, B列, ..., J列)
    // リアルタイム制御ここまで
}
if(CmdFlag == CTRL_EXIT){
```

値をリアルタイムに表示できる



ARCS6の機能

- 以降は作成中
- まだまだ機能の解説は続く……