

動物を用いた心理学実験における赤外線式タッチパネルの導入

立教大学 中田龍三郎・長坂泰勇・長田佳久

Introduction of optical touch panels for psychological experiments using animals
Ryuzaburo NAKATA, Yasuo NAGASAKA, Yoshihisa OSADA (Rikkyo University)

Recently touch panels have appeared in our daily lives, for example ticket machines or cash dispensers. They are also useful for psychological experiments when presenting stimuli on displays. They are especially suitable for experiments using animals. By using a touch panel we can easily get a response from animals and reduce the days for training. Five different systems are used in touch panels and they have different characteristic. Among these systems, we chose the optical touch panel because they are most useful for laboratory use. Since the control programs were written in the C programming language, it is easy to modify the data processing even for a beginner in computer programming. For experiments, the spatio-temporal resolutions of touch panel are 40 ms per scan and 3 mm per point. This is quite sufficient for experimental purpose.

Key Words: Touch panel, instrumentation, C language

タッチパネルとはモニターの前面に設置して使用する入力装置である。キーボードやスイッチによる入力と違い、モニターに提示された画像に直接ふれることで入力を得ることができる。これにより簡便で無駄のない反応取得を行うことができる。

われわれの日常生活においてもタッチパネルを利用する機会が増えている。駅の券売機や銀行のキャッシュディスペンサーなど、日常の様々な場面で利用されている。人と機械を密接につなげるインターフェイスとしてタッチパネルは急速に普及しつつある。

心理学実験においても、視覚刺激を用いる場合にタッチパネルを反応取得装置として使用することは有効な手段であると考えられる。特に動物を用いた実験に使用する場合にメリットが大きい。

Figure 1 にタッチパネルを用いた動物用実験装置の例を示す。動物実験では被験体に実験に必要

な行動を訓練する必要があるが、できる限り訓練の手間を省くためには、より簡易で扱いやすい入力装置が望まれる。視覚刺激の同定などを課題とする場合、提示された刺激を直接さわって反応を取得することはキーによる反応取得などと比べより直接的な反応取得であり、動物にとっても学習しやすいはずである。また実験において刺激の形

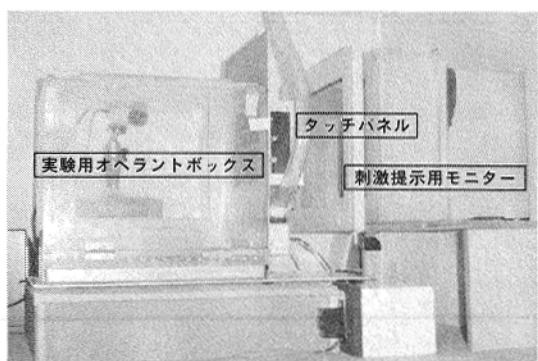


Figure 1 実験装置

や場所などが変化する場合でも、刺激それ自体から入力を得るタッチパネルでは変化に対応し易い。したがってタッチパネルは実験者の意図した計画通りに実験を進めるうえで大変有効であるといえる。

タッチパネルの動作方式の種類

タッチパネルにはいくつかの動作方式がある。どの方式もヒトによる使用を前提としているが、動物を用いた心理学実験においても十分に使用可能である。しかし方式によって特徴は異なっており、より実験に適したタッチパネルの使用が望ましい。どのようなタッチパネルを使用するのが最も適しているのだろうか。以下でそれを検討する。

現在のタッチパネルの動作方式には、①超音波方式、②静電容量方式、③赤外線方式、④抵抗膜方式、・歪方式の5種類がある。Table 1は各方式の特性を比較したものである。各方式の長所短所を以下に簡単に説明する。

超音波方式とはパネルの表面に超音波を伝え、タッチによるその波の弱まりを検出することで反応を取得する方法である。この方法は応答速度7~21msと時間解像度の面で他のタッチパネルより優れ、押圧の強さを測ることができるという他の方式にない特徴をもっている。しかし、光透過率が85~92%となっており、視覚刺激をつかった心理学実験の反応入力装置として用いるにはやや

問題がある。また比較的新しい技術のため現時点においては他の方式より高価であり、導入コストの点でも問題となる。

静電容量方式とは電圧電界をパネルに均一に分散したものであり、パネルに触ると接触部分の電界が変化し、その変化を感知することで反応を取得する方式である。この方式は反応が0.25ミリメートル間隔で取得可能とされ空間解像度の面で優れている。しかし耐傷性に問題があり、傷をつけてしまうとその部分の反応取得が行えなくなる。また電圧電界の膜をパネルに散布している影響として光透過率が85~90%と低い。さらに人間の指による入力を主な用途としているため、動物実験に使用する場合はヒトの指と同じように電解を変化させることのできる部位での入力しか用いることができない。たとえば鳥類のくちばしによるベッキングには対応できないと考えられる。

赤外線方式とは赤外線の発光素子(LED)を使用し投受光の光の遮りにより反応を検知する方式である。この方式ではパネル表面には装置がないため、刺激提示部分での光透過率を100%とすることも可能である。また耐傷性、耐久性に非常に優れている点も長所である。さらにこの方式では赤外線を利用するため、実際の入力に使われる対象はどのようなものでも可能である。すなわち人の指だけでなく、実験動物がキー押しに用いるいずれの身体部位であっても利用可能で

Table 1 タッチパネル制御方式の比較

動作方式	超音波方式	静電容量方式	赤外線方式	抵抗膜方式	歪方式
タッチ耐久性	5000万回以上	約2000万回	LED寿命による制限	約100万回	不明
応答速度	7~21ms	8~24ms	25~50ms	20~40ms	150~250ms
光透過率	89~92%	85~90%	100%	50~80%	100%
耐傷性	傷による影響なし	影響有	非常に耐性がある	影響大	非常に耐性がある
分解能	0.85mm	0.25mm	1.5~3mm	0.3~0.5mm	1.5mm
入力不可能物	水分の少ないもの	指以外のもの	光を透過するもの	フィルムを傷つけるもの	不明

※数値はすべて参考値

ある。この方式の短所として、応答速度が25~50msであり、また空間解像度は1ポイントが1.5~3ミリメートル毎と他のタッチパネルに比べてやや劣る点があげられる。さらに赤外線発光・受光装置を画像出力装置の前面枠に取り付ける必要があるため、他の方式と比べて画像出力装置との間に距離ができ実際に提示される刺激とタッチパネルの反応取得部位との間にズレが生じることがある。

抵抗膜方式とは2枚のパネルの間に電極を設け、押圧により通電された電気信号を受け取ることにより反応を取得する方式である。この方式は最も多くのタッチパネルに利用されており、導入コストが安価である。ただし耐久性や耐傷性には問題があり、長期間の使用により反応の取得に悪影響が出る可能性がある。また反応取得には人間の指と同程度の圧力が必要となるため、小型の実験用動物では反応の取得が困難な場合がある。鳥類など、くちばしでのベッキングにより反応を取得する場合も反応の取得と装置の耐久性の両方で問題となることが考えられる。

歪方式とは画像出力装置の台座部分に制御装置を設置し、画像出力装置の傾きを検知することで反応を取得する方式である。画像出力装置の台座部分に装置を設置するために刺激提示部分には装置が設置されず、光透過率100%となる。しかし反応速度が他の方式に比べて大きく劣り、また安定した設置環境や使用ごとのこまめな調整が必要となる。

動物実験における 赤外線式タッチパネルの採用

各方式の様々な特性をふまえ、われわれが動物を用いた心理学実験に使用するうえで採用したタッチパネルは赤外線方式であった。

赤外線式タッチパネルは赤外線発光、受光装置のために他のタッチパネルよりもやや多くのズレを生んでしまう。また、前述したように方式自体の問題点として時間・空間解像度において他のタッチパネルより多少劣る。しかし、ズレの差はわず

かなものであり、実際の実験場面において問題となることは皆無である。また時間・空間解像度に関しても時間解像度で10ms程度の差であり、空間解像度でも数ミリ単位の差でしかない。この程度の差であれば、実際の心理学実験において問題はほとんどないと考えられる。

むしろこの方式には上記の問題を補うに足る大きな利点がある。すなわち耐久性や耐傷性などに優れており、さらに光透過率を100%にすることができる。このことは視覚刺激を用いる実験において大きな魅力である。また他の方式と比べ、入力に使われる対象がどのようなものでも可能な点など、動物を対象として使用するとしても問題点が少ないと導入の理由である。

Macintosh用制御プログラムの作成

実験に使用する場合、できる限り自由度が高く様々な実験に対応できる装置であることが要求される。タッチパネルは最初からディスプレイに直接内蔵されているものが主流である。われわれが日常場面で使用するタッチパネルもほとんどが内蔵型である。実験に使用する上では取り外し可能な外付け式で、かつ実験に使用しているパソコンで制御可能なものが望ましい。この条件に適した外付け用の赤外線方式タッチパネルは数社から販売されているが、すべてMicrosoft Windowsでの使用を前提としており、われわれの研究室で使用しているMacintosh上で制御可能なものは皆無であった。そこで今回、MacintoshでのCarroll Touch（以下、タッチパネル）制御を可能にするプログラムをC言語で作成した。

MacOS9用タッチパネル制御サブルーチン

List 1のcarroll Touch.cに示す関数群の説明は下記の通りである。なお以下の関数を作成するにあたり、齊藤（1994）および参考資料④を参考にした。

int CT_Open Init Serial Port (void) 関数

シリアルポート（モデムポート、プリンタポート）のオープンおよびパソコンとタッチパネルと

の通信のための通信パラメータの設定を行う。はじめに出力側を設定し、その後で入力側を設定しなければならない。またList 1で示してあるのはプリントポートを使用する場合であり、モデルポートを使用する場合には "¥p.A Out" および "¥p.A In" をそれぞれ "¥p.B Out" および "¥p.B In" とする。

int CT_Auto Negotiation (void) 関数

パソコンとタッチパネルとの通信を確立する(ネゴシエーション)。ネゴシエーションの最も簡単な方法はプログラムにあるように、1) 400ms 以上のブレーク信号の送信、2) 500ms 待機後、3) 100ms の送信間隔で 0 DH を 5 回送信し、4) 100ms 待機後、5) 3 CH を送信することである。さらに本関数では、パソコンとタッチパネル間通信の確立を確認するために、タッチパネルにシステム構成レポート要求(33H)、データ送信要求(44H)を行い、タッチパネルから正確にデータが送信されているかを確かめている。

int CT_Clear (void) 関数

タッチパネル内のバッファに保持されているデータをクリアする。タッチパネルが動作状態にあるとき、タッチパネルへの反応はすべてこのバッファに保持される。したがってタッチパネルへの反応を取得する直前にこの関数を使用し、それまでのタッチデータをクリアする必要がある。

int CT_Send ON (void) ・ int CT_Send OFF (void) 関数

タッチパネル内のバッファの情報の送信を開始または停止する。タッチパネルが動作状態であっ

ても、送信命令が出されていない場合にはデータはパソコンに送信されず、バッファ内に保持される。

int CT_Measure ON (void) ・ int CT_Measure OFF (void) 関数

タッチパネルへの反応取得を開始または停止する。取得された反応はタッチパネル内のバッファに保持される。

int CT_Set Modes (int mode) 関数

タッチパネルの動作モードを設定する。タッチパネルは 4 つの動作モードを持っており(Table 2)，目的に応じてそのモードを使い分ける必要がある。

void CT_Check Status (void) 関数

現在のタッチパネルの状態を取得する。

Point CT_Read XY (void) 関数

タッチパネルからタッチ情報(反応位置の X, Y 位置情報)を取得する。データが取得されると、バッファ内のデータは自動的に削除される。このデータはタッチパネル上の座標を表しており、パソコンで使用するピクセル単位とは異なる。したがって実際の使用には、後述するキャリブレーションを行って補正式を求め、タッチ情報を変換する必要がある。

int CT_Close (void) 関数

タッチパネルとの通信を切断し、シリアルポートをクローズする。

int com Send (char *buf, int len) 関数

タッチパネルへシリアルポート経由でデータを送信する。上記サブルーチン群の内部で使用して

Table 2 タッチパネルの動作モード(キャロルタッチ入力システム スマートフレームV1.0マニュアルより)

動作モード	命令コード	動作
入力点動作	25H	スタイルスがフレームに入ったときの座標を出力。タッチしていない状態が発見されるまで送信を停止
追従動作	26H	スタイルスがタッチ有効エリアにある限り座標を送信。スタイルスが静止している場合には送信しない。
連続動作	27H	追従動作と同じであるが、スタイルスが静止していても座標を送信する。
終了点動作	28H	スタイルスが有効エリアを外れたとき、もしくはタッチしていない状態が発見されたときの座標を送信

*スタイルスとはタッチパネルにふれる対象(ペン・指など)を指す

いる。

int comRecv (unsigned char *buf, int len, int wait) 関数

タッチパネルからシリアルポート経由でデータを受信する。上記サブルーチン群の内部で使用している。

int checkStatus (void) 関数

シリアルポートの状態を取得する。

int doSendBreak (int delay2) 関数

引数で示された時間ブレーク信号を送信する。
引数の単位はミリ秒である。

void dump (char *buf, int len) 関数

受信した情報を表示する。デバッグ時のエラーチェック等で使用する。

プログラムへの実装

上で述べた関数群を使用することによりタッチパネルへの反応を取得することが可能であるが、その手続きは次のようになる。1) シリアルポートのオープン (CT_OpenInitSerialPort関数), 2) タッチパネルとの通信確立 (CT_AutoNegotiation), 3) タッチパネル動作モード設定 (CT_SetModes), 4) タッチパネル内のバッファの情報をクリア (CT_Clear), 5) 座標情報の送信開始 (CT_SendON), 6) タッチパネルの反応取得動作開始 (CT_MeasureON), 7) 反応座標情報の取得 (CT_ReadXY), 8) 以下で述べる補正式を用いたタッチパネルディスプレイ間の座標変換。以降、反応取得のために7), 8) を繰り返す。

タッチパネル座標と 画面座標とのキャリブレーション

上で述べたようにタッチパネルにおける座標と、
刺激が提示されているディスプレイの座標との間には大きな違いがある。そのため、タッチパネル座標とディスプレイ座標との対応づけ（キャリブレーション）を行わなければならない。具体的な
プログラム例は省略するが、以下に示す手続きおよび補正式を用いることにより、タッチパネル座

標とディスプレイ座標との対応が可能となる。

キャリブレーション手続き（タッチパネルをモニタに正確に設置する。画面上の任意の点を2つ設定する。ここでは便宜上、Disp A (0, 0), Disp B (640, 480) とする。Disp AおよびDisp B点上に小円を提示し、タッチパネルの上から各小円をボールペンの先端でポイントし、タッチパネルから得られた座標情報をそれぞれTouch A (taX, taY), Touch B (tbX, tbY) とする。補正係数の算出）一次関数近似を行うため、上記4点に対応した式をたてる。すなわちX座標については、 $0 = taX^* Xa + Xb$, $640 = tbX^* Xa + Xb$, Y座標については、 $0 = taY^* Ya + Yb$, $480 = tbY^* Ya + Yb$ とし、各座標について二元一次連立方程式を解くことにより、補正係数である、Xa, Xb, Ya, Ybを算出し、補正式を導くことができる。この補正式を一度算出してしまえば、タッチパネルの設置位置を変更するまで有効である。なおList 2 のcarrollTouch.hには、この係数を表すsCoef構造体を定義してある。

現在の問題点及び今後の課題

現在使用しているタッチパネルの実験時における時間解像度は40ms、空間解像度は約3mmである。これは通常の実験を行うには十分な値である。時間解像度の限界を超えた速度での反応には対応できない点、リザルの指1本だけの反応は取得できない点、両手で同時にタッチするなど離れた二点をタッチされると反応として認められない点などが現段階で問題となっている。この点を克服したうえで、リザル以外のサル類やハトなどの鳥類の実験にもタッチパネルが対応できるかどうかを今後確かめていく必要がある。なお今回のプログラムはMac OS9上での使用を前提としている。現在われわれの研究室で使用している様々な機器は最新OSであるMac OSXには非対応であり、Mac OSX上では実験が行えない現状である。この状況が改善され次第、対応するドライバーを作成する予定である。

List.1 ソースコード "carrollITouch.h"

```
/* carrollITouch.h */

/* struct */
typedef struct {
    double Xa;
    int Xb;
    double Ya;
    int Yb;
}sCoef;

/* prototypes */
void dump(char *buf, int len);
int CT_OpenInitSerialPort(void);
int do_SendBreak(int);
int comSend(char *buf, int len);
int comRecv(unsigned char *buf, int len, int wait);
int CT_AutoNegotiation(void);
sCoef CT_Calibration(void);
int CT_Clear(void);
int CT_SendON(void);
int CT_SendOFF(void);
int CT_SetModes(int);
int CT_MeasureON(void);
int CT_MeasureOFF(void);
Point CT_ReadXY(void);
int CT_Close(void);
int checkStatus(void);
void CT_CheckStatus(void);

/* definition */
#define TOUCH      1
#define TRACKING   2
#define CONTINUOUS 3
#define RELEASE    4

#define TICKS      16
```

List.2 ソースコード "carrollITouch.c"

```
/* carrollITouch.c */

#include <OSUtils.h>
#include <Serial.h>
#include <stdio.h>
#include <string.h>
#include "carrollITouch.h"

#define conv(a) ((a)<0x20?'.':(a))

static short gOutRefNum;
static short gInRefNum;
unsigned long *gFinalTicks;
int gSerialStatus=0;

int CT_OpenInitSerialPort(void)
{
    int error=0;

    // Open serial driver
    if (OpenDriver("Wp.AOut", &gOutRefNum)) {
        printf("error in opening OUT\n");
        return 1;
    }
    if (OpenDriver("Wp.AIn", &gInRefNum)) {
        printf("error in opening OUT\n");
        return 2;
    }
    error=0;
    KillIO(gOutRefNum);
    KillIO(gInRefNum);

    // Initialize serial driver
    //           with 9600bps, data 8bit, stopbit 1, noParity
    if(SerReset(gOutRefNum, baud9600 + data8 + stop1 + noParity)) {
        printf("error in init terminal\n");
        return 3;
    }

    return error;
}

int CT_AutoNegotiation(void)
{
    char sendBuffer[1];
    unsigned char getBuffer[1];

    long delay;
    int len;
    int i;

    do_SendBreak(400);
    delay = 500/TICKS;
    Delay(delay, gFinalTicks);

    delay = 100/TICKS;
    sprintf(sendBuffer, "%c", 13);

    for(i=0; i<5; i++) {
        comSend(sendBuffer, strlen(sendBuffer));
        Delay(delay, gFinalTicks);
    }
}
```

```

delay = 300/TICKS;
sendBuffer[0] = '\x3c';
len = comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* reset */
if(len != 1)printf("error in CT_AutoNegotiation1 send %d bytes\n", len);
Delay(delay, gFinalTicks);

delay = 100/TICKS;
sendBuffer[0] = '\x33';
len = comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* request status */
if(len != 1)printf("error in CT_AutoNegotiation2 send %d bytes\n", len);
Delay(delay, gFinalTicks);

sendBuffer[0] = '\x44';
len = comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* transaction on */
if(len != 1)printf("error in CT_AutoNegotiation3 send %d bytes\n", len);
Delay(delay, gFinalTicks);

getBuffer[0] = '0';
while(getBuffer[0] != 255 /*ff*/) {
    if(comRecv(getBuffer, 1, delay))
        printf("AutoNegotiation... %x received\n", getBuffer[0]);
}

sendBuffer[0] = '\x32';
len = comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* status report */
if(len != 1)printf("error in CT_AutoNegotiation4 send %d bytes\n", len);
Delay(delay, gFinalTicks);

getBuffer[0] = '0';
while(getBuffer[0] != 255 /*ff*/) {
    if(comRecv(getBuffer, 1, delay))
        printf("Status... %x received\n", getBuffer[0]);
}

getBuffer[0] = '0';
for(i=0; i<10; i++) {
    if(comRecv(getBuffer, 1, delay))
        printf("After Negotiation... %x received\n", getBuffer[0]);
}

sendBuffer[0] = '\x43';
comSend(sendBuffer, strlen(sendBuffer));/* transaction off*/
Delay(delay, gFinalTicks);

gSerialStatus=1;
return 0;
}

int CT_Clear(void)
{
    long delay;
    char sendBuffer[1];

    delay = 50/TICKS;
    sendBuffer[0] = '\x3d';
    comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* clear CT buffer*/
    Delay(delay, gFinalTicks);
    return 0;
}

int CT_SendON(void)
{
    long delay;
    char sendBuffer[1];
    long len;

    delay = 50/TICKS;
    sendBuffer[0] = '\x44';
    len= comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* transaction on */
    if(len != 1)printf("error in CT_SendON got %d bytes\n", len);
    Delay(delay, gFinalTicks);

    return 0;
}

int CT_SendOFF(void)
{
    long delay;
    char sendBuffer[1];
    long len;

    delay = 50/TICKS;
    sendBuffer[0] = '\x43';
    len= comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* transaction off */
    if(len != 1)printf("error in CT_SendOFF\n");
    Delay(delay, gFinalTicks);
    return 0;
}

int CT_MeasureON(void)
{
    long delay;
    char sendBuffer[1];
    long len;

    delay = 50/TICKS;
    sendBuffer[0] = '\x2a';
    len = comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* scan on */
    if(len != 1)printf("error in CT_MeasureON got %d bytes\n", len);
    Delay(delay, gFinalTicks);
}

int CT_MeasureOFF(void)
{
    long delay;
    char sendBuffer[1];
    long len;

    delay = 50/TICKS;
    sendBuffer[0] = '\x2b';
    len = comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* scan off */
    if(len != 1)printf("error in CT_MeasureOFF\n");
    Delay(delay, gFinalTicks);
    return 0;
}

int CT_SetModes(int mode)
{
    long delay;
    char sendBuffer[1];
    long len;

    delay = 100/TICKS;
    sendBuffer[0] = '\x23';
    len = comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* coordination report
mode*/
    if(len != 1)printf("error in CT_SetModes1 send %d bytes\n", len);
    Delay(delay, gFinalTicks);

    if(mode == TOUCH)
        sendBuffer[0] = '\x25';
    else if(mode == TRACKING)
        sendBuffer[0] = '\x26';
    else if(mode == CONTINUOUS)
        sendBuffer[0] = '\x27';
    else if(mode == RELEASE)
        sendBuffer[0] = '\x28';
    else
        sendBuffer[0] = '\x28';
    len = comSend(sendBuffer, 1/*strlen(sendBuffer)*/); /* endpoint report
mode*/
    if(len != 1)printf("error in CT_SetModes2 send %d bytes\n", len);
    Delay(delay, gFinalTicks);
}

```

```

    return 0;
}

void CT_CheckStatus(void)
{
    char sendBuffer[1];
    unsigned char getBuffer[1];
    int delay;

    delay = 100/TICKS;
    sendBuffer[0] = 'X32';
    comSend(sendBuffer, strlen(sendBuffer)); /* status report */
    Delay(delay, gFinalTicks);

    getBuffer[0] = '0';
    while(getBuffer[0] != 255 /ff/) {
        if(comRecv(getBuffer, 1, delay))
            printf("Status... %x received\n", getBuffer[0]);
    }
}

Point CT_ReadXY(void)
{
    Point theXY;
    unsigned char getBuffer[1];
    unsigned char xyData[2];
    long delay;
    int len;

    delay = 100/TICKS;
    theXY.h = 0;
    theXY.v = 0;

    getBuffer[0] = '0';
    len=comRecv(getBuffer, 1, delay);

    if(getBuffer[0] == '0') {
        return theXY; /* no data */
    }
    else if(getBuffer[0] == 252/*fc*/){ /* error data */
        while(getBuffer[0] != 255 /ff/) {
            comRecv(getBuffer, 1, delay);
        }
        return theXY;
    }
    else if(getBuffer[0] == 254/*fe*/){ /* xy data */
        comRecv(xyData, 2, delay);
        while(getBuffer[0] != 255 /ff/)
            comRecv(getBuffer, 1, delay);
        theXY.h = xyData[0];
        theXY.v = xyData[1];
    }
    return theXY;
}
return theXY;
}

int CT_Close(void)
{
    OSerr err;
    long delay;
    char sendBuffer[1];

    delay = 200/TICKS;
    sendBuffer[0] = 'X3d';
    comSend(sendBuffer, strlen(sendBuffer)); /* reset */
    Delay(delay, gFinalTicks);

    KillIO(gOutRefNum);
    KillIO(gInRefNum);

    if(CloseDriver(gOutRefNum) )
        printf("error in closing OUT\n");
    return false;
}

int CT_CheckStatus(void)
{
    if(CloseDriver(gInRefNum) )
        printf("error in closing IN\n");
    return false;
}

int comSend(char *buf, int len)
{
    long length;
    OSerr err;

    length = len;
    if( err = FSWrite(gOutRefNum, &length, buf) )
        printf("error in comSend code: %d\n", err);
    return 0;
}

int comRecv(unsigned char *buf, int len, int wait)
{
    long delay;
    long l, length = 0;
    OSerr err;

    delay = wait;
    while(len > 0) {
        err = SerGetBuf(gInRefNum, &l);
        if(err)printf("error in SerGetBuf\n");
        if(l > 0) {
            if(l > len) {
                l = len;
            }
            err = FSRead(gInRefNum, &l, buf);
            if(err) {
                printf("error occurred in FSRead, %d\n", err);
                break;
            }
            buf += l;
            len -= l;
            length += l;
        }
        else{
            Delay(delay, gFinalTicks);
            break;
        }
    }
    return length;
}

int checkStatus(void)
{
    OSerr err;
    SerStaRec status;

    if(SerStatus(gInRefNum, &status))
        printf("error occurred in SerStatus\n");
    return true;
}

int do_SendBreak(int delay2)
{
    OSerr err;
}

```

```

long    delay;
delay = delay2/TICKS;

if(SerSetBrk(gOutRefNum)) {
    printf("error in sending break\n");
    return false;
}
printf("Sending break...\n");
Delay(delay, gFinalTicks);
if(err = SerCirBrk(gOutRefNum)) {
    printf("error in clear break\n");
    return false;
}
printf("Stop sending break...\n");
return true;
}

void dump(char *buf, int len)
{
    int i;

    printf("dump: length = %d(0x%02x)\n", len, len);
    while(len > 16) {
        for(i = 0; i < 16; i++) {
            printf("%02x ", (unsigned char)buf[i]);
        }
        printf(" : ");
        for(i = 0; i < 16; i++) {
            printf("%c", cnv((unsigned char)buf[i]));
        }
        printf("\n");
        len -= 16;
        buf +=16;
    }
    for(i = 0; i < len; i++) {
        printf("%02x ", (unsigned char)buf[i]);
    }
    printf(" : ");
    for(i = 0; i < len; i++) {
        printf("%c", cnv((unsigned char)buf[i]));
    }
    printf("\n");
}

```

引用文献

齐藤亮 1994 高速・長距離が可能なシリアル・ポート 金子俊夫（編）OPEN DESIGN No.2 第2章, CQ出版株式会社 Pp. 17-32.

参考資料

①三菱電気エンジニアリング株式会社ホームページ
<http://www.mee.co.jp/>

②高木商会株式会社ホームページ

<http://www.takagishokai.co.jp/>
 ③タッチパネル・システムズ株式会社ホームページ
<http://www.tps.co.jp/>
 ④キャロルタッチインターナショナル日本支社
 キャロルタッチ入力システム スマートフレーム V1.0取扱説明書