平成 27 年度　特別研究報告書

加速度センサを利用した簡易ポインティングデバイスの作成と最適な操作方法の検討

龍谷大学　理工学部　情報メディア学科

T120421　大森慎也

指導教員　三好　力　教授

内容梗概

現在、人がコンピュータを操作する際、直感的な操作を可能にする操作方法に GUI(Graphical User Interface)が広く採用されており、画面上での座標を指定するための入力機器にポインティングデバイスを使用している。しかし、既存のポインティングデバイスは、他の作業との並列作業を行うことが多い環境で使用できるものが少ない難しい問題がある。一例として、工事現場で使用するポインティングデバイスについて考えると、手袋着用の状態での作業が多く、手袋の汚れ・濡れが想定され、回転・摩擦を利用するタイプのポインティングデバイスが使用しにくい問題と、テーブルがない場所での操作が必要な場面も想定されることから、物体上で操作するタイプのポインティングデバイスは使用しにくい問題がある。そこで、本論文では、工事現場での環境下の使用を想定し、並行作業しやすいポインティングデバイスを考案し、カーソル移動とクリックができる程度の簡易ポインティングデバイスを作成、その操作性を評価・最適な操作法を検討する。

目次

第1章　はじめに

1.1　研究背景

現在、人がコンピュータを操作する際、直感的な操作を可能にする操作方法に GUI(Graphical User Interface)が広く採用されており、画面上での座標を指定するための入力機器にポインティングデバイスを使用している。

ポインティングデバイスには様々な用途・環境・状況に応じたものがあり、代表的なポインティングデバイスとして、机の上でデバイスを移動させ、その移動方向に応じてポインタを制御・ボタンを押すことでクリックを行うマウスや、ディスプレイに触れることで、触れた座標を取得しポインタを制御・ディスプレイから指をタップしクリックを行うするタッチパネルディスプレイなどがあり、人の手の移動による操作方法が広く使用されている。

しかし、これらのポインティングデバイスは、物体上において使用する必要性、手で持つ・触れる必要性などの制約があり、他の作業との並列作業を行うことが難しい問題がある。一例として、工事現場で使用するポインティングデバイスについて考える、手袋着用の状態での作業が多く、手袋の汚れ・濡れが想定され、触れるタイプのポインティングデバイスが使用しにくい問題と、テーブルがない場所での操作が必要な場面も想定されることから、物体上で操作するタイプのポインティングデバイスは使用しにくい問題がある。作業図面の確認やデータの記録など、工事現場での IT を有効に活用するため、作業員が利用しやすいポインティングデバイスは必要である。

1.2　研究の目的

1.1 節で述べた問題から、工事現場のような環境下で作業を行いやすいポインティングデバイスが必要であると考えた。本研究では、工事現場での環境下の使用を想定し、並行作業しやすいポインティングデバイスを考案し、カーソル移動とクリックができる程度の簡易ポインティングデバイスを作成、その操作性を評価・最適な操作法の検討をすることを目的とする。

本論文における、「並行作業を行いやすいポインティングデバイス」とは以下の 3 点を持ったものである。

1. ハンズフリーである。
2. 空中で使用可能である。
3. 省スペースで使用可能である。

まず、第一に、ユーザにとってハンズフリーである点が最も重要だと考える。手を使って様々な作業を行うため、ハンズフリーであれば作業の進行を阻害せず利用できる。

第二に、空中で使用可能である点も重要な要素である。工事現場のような移動の多い環境ではテーブルのない場所でポインティングを行いたい場面も想定されるため、空中でもポインティングを行える必要があると考える。

最後に、省スペースで使用可能な点である。作業スペースの少ない場所での使用を行い易くするためにも、ポインティングする際のアクションは小さく行えると良いと考える。

これら 3 点を解決するため、ポインティングデバイスをウェアラブル化することが有効であると考える。ポインティングデバイスをウェアラブル化するにあたり、工事現場では手袋を装着している場面が多いことから、作成するポインティングデバイスをグローブ型にすることで、ハンズフリーな環境にする。次に、空中で操作可能・省スペースで操作可能にするため、ポインタの制御に姿勢（傾き）・加速度を検出できる加速度センサを利用し、手の小さな動きでポインティングを行えるようにする。

以上のような、加速度センサ利用のグローブ型簡易ポインティングデバイスを考案・作成し、操作性の検討として、デバイスの傾きによる操作方法、マウスとタッチパネルディスプレイと同様の操作方法を再現し、どの操作方法が最も画面上のポインタを操作する上で最適であるかを検討する。

第2章　既存技術

2.1　概要

ポインティングデバイスには様々な用途・環境・状況に応じたものが存在している。本章では、既存のポインティングデバイスについての説明を以下に示す。

2.2　マウス

2.2.1 物体上において使用するマウス

マウスとは一般にボール式と光学式の2種類のタイプがある。図 2.2.1 に2種類のマウスを示す。

ボール式マウスとは、表にボタンが、裏にボールを内蔵したマウスである。内蔵されたボールの回転からマウスの移動の様子を調べて画面上のカーソルを移動[2]・ボタンを押しクリックを行う。

光学式マウスとは、表にボタンが、裏に LED とイメージセンサを内蔵したマウスである。LED から光を発射し、その反射光の変化からマウスの移動の様子を調べて画面上のカーソルを移動[2]・ボタンによりクリックを行う。また、工事現場で使用される既存のポインティングデバイスに、防水・防塵性能の備えた光学式マウスがある。[3]

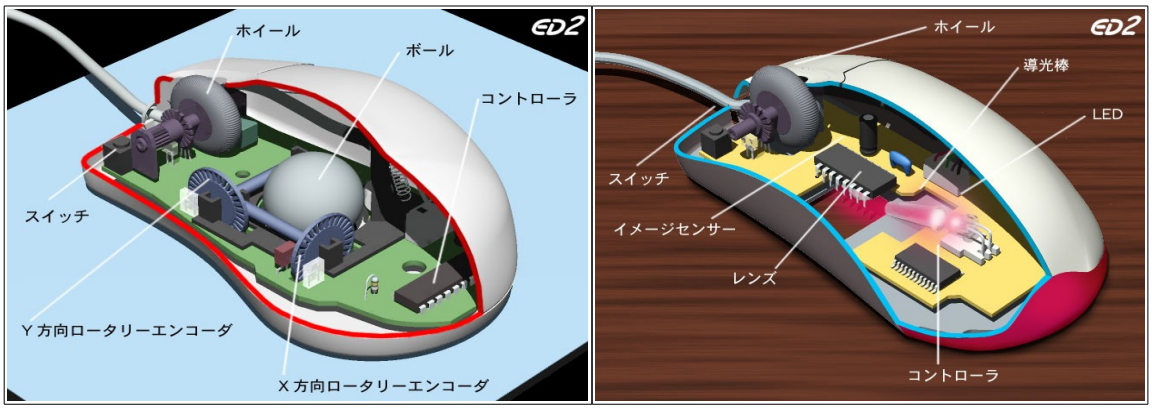

図 2.2.1 左：ボール式マウス, 右：光学式マウス

2.2.2 空中で使用できるマウス

2.2.2.1 ジャイロセンサ搭載マウス

角度・角速度を検出するジャイロセンサを搭載したマウス。これは、ジャイロセンサが自らの移動による角速度を検知し、積分演算処理により角度を検出することで、移動した角度（方向）を検出し、ポインティングを行う。[4] また、クリックは前節 2.2.1 の物体上に置いて使用するマウス同様、ボタンを押して行う。図 2.2.2.1 にジャイロセンサ搭載マウスを示す。

図 2.2.2.1 ジャイロセンサ搭載マウス

2.2.2.2 トラックボール式親指マウス

2.2.1 節のボール式マウスを逆さまにしたようなマウス。空中で握って、親指でボールを転がしてカーソルを操作、人差し指のトリガーボタンでクリックを行う。[5] 図 2.2.2.2 にトラックボール式親指マウスを示す。

図 2.2.2.2 トラックボール式親指マウス

2.2.2.3 指マウス

手の人差し指に装着し、指の動きに応じてポインティングを行うマウス。光学式トラックセンサを使用し、センサー上を親指でなぞり操作するタイプ[6]と、2.2.2.1 節のようにジャイロセンサを使用し、空中で人差し指を動かして操作するタイプ[7]がある。また、クリックはどちらもボタンを押すことでクリックを行う。図 2.2.2.3 に光学式指マウス、ジャイロセンサ搭載指マウスをそれぞれ示す。

図 2.2.2.3 左：光学式指マウス, 右：ジャイロセンサ搭載指マウス

2.3 タッチパネル[8]

タッチパネルとは、表示装置（ディスプレイ）と入力装置（ポインティングデバイス）が一体化したもので、指が触れた位置をセンサーで検知してポインティングを行うことができるため、マウスよりも直感的に操作できる。タッチパネルではタップ（触れて放す）を行うことでクリックを行っている。図 2.3 にタッチパネルを示す。

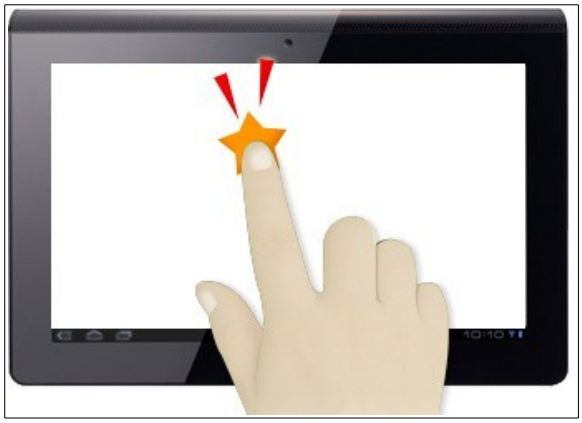

図 2.3 タッチパネル

また、タッチパネルは触れた位置を検知する手法がいくつかある。

2.3.1 抵抗膜方式

指やペンなどで押した画面の位置を電圧変化の測定によって検知する。内部構造は、それぞれ透明電極膜（導電層）を配置したガラス面とフィルム面を少しだけすき間を設けて張り付けたシンプルなものである。フィルムの表面を押すと、フィルム側とガラス側の電極同士が接触して電気が流れ、その電圧の変動を検出することで接点の位置をとらえる。

フィルムへの圧力で入力するため、指だけでなく、手袋をしたままの状態で入力できる。

2.3.2 静電容量方式

画面に指で触れると発生する微弱な電流、つまり静電容量（電荷）の変化をセンサーで感知し、タッチした位置を把握する。指を画面に近づけると、人体の静電容量にセンサーが反応するため、画面に接触する寸前でポインターを動かすような操作が可能である。

抵抗膜方式と違って衣服の袖や通常のペンには反応せず、ホコリや水滴に強く、耐久性や耐傷性が高い。しかし、指以外では静電容量方式に反応する専用タッチペンでしか操作できず、手袋をしたまま扱うことができない。

2.3.3 超音波表面弾性波（SAW）方式

指などで触れた画面の位置を超音波表面弾性波の減衰によって検知する。内部構造としては、ガラス基板の隅に配置した複数の発信子（圧電トランスデューサ）から、パネル表面に振動として伝わる超音波表面弾性波を出し、発信子の向かい側に設置した受信子でこれを受ける。画面に触れた場合、超音波表面弾性波が指などに吸収されて弱まるため、この変化を検知することで、位置を特定できる。

超音波表面弾性波を吸収できる指や柔らかいもの（手袋など）でタッチ操作が行える。しかし、パネル上に付いた水滴や小さな虫などの異物にも反応してしまう。

2.3.4 光学方式

パネル上辺の左右端に赤外線 LED を 1 つずつと、イメージセンサー（カメラ）を配置し、残る左辺、右辺、下辺には入射した光を入射方向に反射させる再帰反射テープを張り付けている。指などで画面に触れると、赤外光が遮光された影をイメージセンサーがとらえ、三角測量によって座標が求められる。

指や手袋をしたままの状態、通常のペンを使って入力ができる。しかし、イメージセンサーを表示領域の外側に設けることから、検出精度が外光の影響を受けやすいため、屋外での使用には不向きである。

2.3.5 電磁誘導方式

センサー部を液晶パネルに統合し、高精度のタッチパネルを実現している。磁界を発生する専用ペンで画面をタッチすることで、パネル側のセンサーが電磁エネルギーを受け取り、位置を検出する。

入力には専用ペンを使うため、指や汎用のペンで入力できない。

2.4 赤外線を利用したポインティングデバイス

2.4.1 wii リモコン

任天堂株式会社より販売されているゲーム機 wii を操作するコントローラー。ポインティングはリモコンに3軸加速度センサと赤外線を検知する赤外線 CMOS センサ、画面付近に2つの赤外線 LED を搭載したセンサーバーを使用している。センサーバーの2つの赤外線 LED をリモコンの先端部分に搭載された赤外線 CMOS センサで読み取り、二つの光点の位置関係と間隔から画面に対する Wii リモコンの向きや距離がある程度見積ることができる。この情報と 3 軸加速度センサから得たリモコンの姿勢（傾き）情報を組み合わせることにより、ポインティングを行っている。[9]

図 2.4.1 左:センサーバー, 右:wii リモコン

2.4.2 Leap Motion

Leap Motion 社より販売された手のジェスチャーによってコンピュータを操作できるデバイス。前節の wii リモコンとセンサーバーを一体化したようなもので、3つの赤外線 LED で赤外線を照射し、跳ね返りの赤外線を2つの赤外線カメラで検知し、手の形を検知する。手の形を検知できるため、ジェスチャーによりポインティングを行うことができる。[10] 図 2.4.2.に Leap Motion を示す。

図 2.4.2 Leap Motion

2.4.3 Kinect

Microsoft 社より販売された、コントローラーを用いずに身体の動き、ジェスチャー、音声などによる操作を可能とするデバイス。カラーカメラと赤外線カメラ、赤外線レーザプロジェクタを搭載しており.赤外線レーザプロジェクタから空間に投影された見えない赤外線のパターンを,赤外線カメラで読み取ることによって,奥行きを計測し、体の動きを検知している。[11] 図 2.4.3 に Kinect を示す。

図 2.4.3 Kinect

2.5　既存技術の問題点

まず 2.2.1 節のマウスはハンズフリーでなく、空中使用ができないため、ユーザの並行作業が困難であると考える。2.2.2.1, 2.2.2.2 節については、どちらのデバイスも「持つ」必要性がありハンズフリーな状態ではない。また、2.2.2.1 節のジャイロセンサによるポインティングデバイイスは手の移動でポインティングを行うことから、ある程度のスペースが必要であると考えられる。2.2.2.3 節は指につけることで操作を行うのでハンズフリーな点と空中操作可能な点を満たすが、ジャイロセンサ搭載のタイプは前述したとおり、スペースを必要とし、光学式トラックセンサの使用は手袋をつけている状態で指をセンサにかざした場合、誤作動を起こしやすい。また、汚れ・濡れなどの要素からも、うまくポインティングを行えないと思われる。

次に 2.3 節については、工事現場では手袋を装着しており、手袋の汚れ・濡れ、屋外での使用など考慮すると、どの方式のタッチパネルも使用できない。

最後に 2.4 節についてはどちらも赤外線を利用しているため、太陽光による誤作動が考えられ、屋外使用ができない問題がある。

第 3 章　提案手法

3.1 概要

第 2 章 2.5 節で前述したように工事現場のような汚れや濡れのあるで環境では既存のポインティングデバイスは使用しにくい問題点がある。

そのため、本研究では，使用環境に工事現場を想定し、並行作業しやすい簡易ポインティングデバイスを提案する。

3.2 使用するセンサについて

本論文で提案する簡易ポインティングデバイスに使用するセンサを以下に示す。

3.2.1 加速度センサ

加速度センサとは、その名の通り加速度を検出することができるセンサである。加速度センサには様々な種類があり、加速度の検出方式によって大別される。今回提案するポインティングデバイスに使用する加速度センサは静電容量式加速度センサであり、簡単な動作原理について説明する。

静電容量式加速度センサは、ガラスやシリコンなどのおもりを使ったセンサ素子可動部（可動する素子）、固定部（固定された素子）から構成されている。加速度が印加されると可動部が動き、可動部と固定部の間の静電容量が変化するので、この変化を検出し加速度の変位を得る方式である。[12]

加速度を検出できるので、重力加速度も検出できる。これを利用することでセンサの傾きを検出できる。本実験では、このセンサを使用し、センサの移動による加速度とセンサの傾きによりポインティングを行う。

3.2.2 曲げセンサ

曲げセンサとは、曲げることで抵抗値が変化するセンサである。この変化からセンサの曲がり具合を検出することができる。

本実験では、簡易的なグローブ型ポインティングデバイスを作成するにあたり、人差し指部分に曲げセンサを装着し、指の曲げ具合を検出し、クリック処理を行う。

3.2 提案するポインティングデバイスの構成図

本研究で提案するポインティングデバイスの構成図を図 3.2.1 に示す。

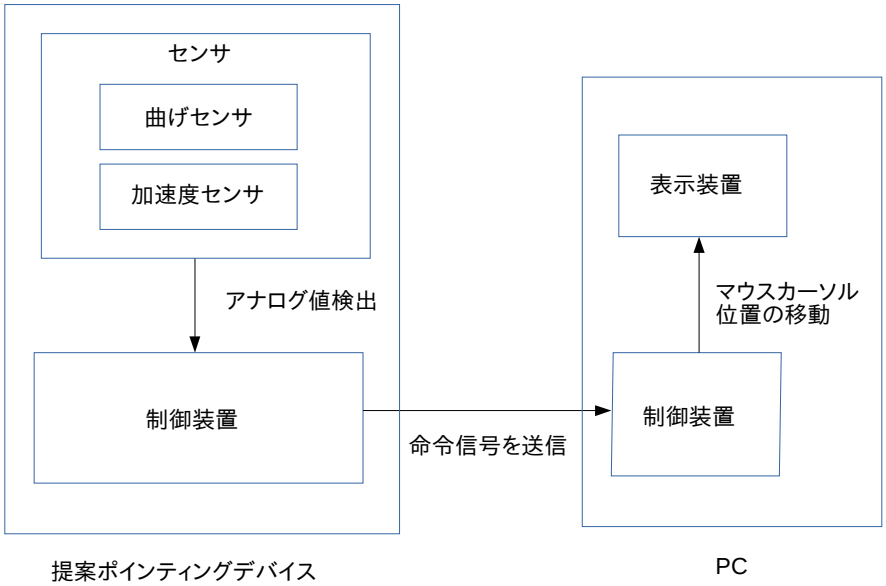

図 3.2.1 ポインティングデバイスの構成図

ポインティングデバイス側で各センサで検出したアナログ値を制御装置で計算し、命令信号を PC に送信する。

PC 側でポインティングデバイスから受信した命令信号よりマウスカーソルの移動・クリック処理を行い、表示装置に反映させる。

3.3 システムのフローチャート

本研究で作成するポインティングデバイスのシステムフローチャートを図 3.3.1 に示す。

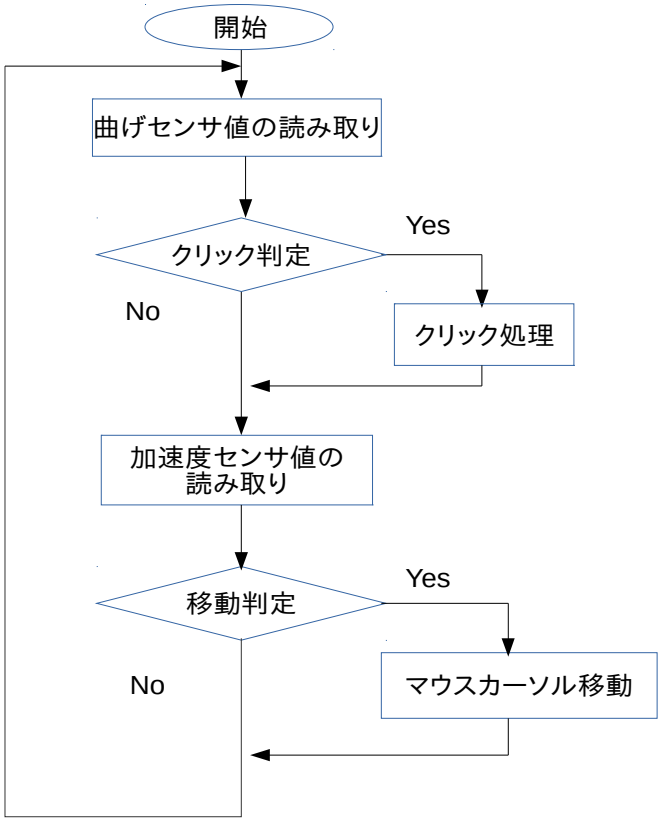

図 3.3.1 システムのフローチャート

曲げセンサのクリック判定は、曲げセンサ値を読み取り、一定の曲がり具合を超えた場合にクリックを行う。また、マウスカーソルの移動は、加速度センサ値を読み取り、その値に応じて方向・移動距離を変化させる。

3.4 ポインティングメタファ

本研究では最適な操作法の検証として 3 つのポインティングメタファを用意し、各メタファを実現するための専用プログラムを作成する。

3.4.1 十字キーメタファ

手をゲームパッドの十字キーに見立て、ポインタ制御に手の傾きを利用することにより制御する。加速度センサは傾き具合を検出することができるため、手の傾きでポインタを制御する。ポインタの移動は手の甲を傾きに応じた方向へ移動させる。

ポインタ操作方法は、ポインティングデバイスを装着した手の甲を上に向け、地面に対し平行な状態を静止状態の基準とし、基準からの手の傾き方向によるポインタ移動方向を表 3.4.1 に示す。

表 3.4.1 十字キーメタファの移動

手の傾き方向	奥	手前	右	左
ポインタ移動方向	上	下	右	左

3.4.2 テーブルメタファ

第 2 章 2.2.1 節のマウスを操作するモーション（地面に対し、x 軸 y 軸：水平移動）を空中で行い、手の移動による加速度を用いてポインタを制御する。ここで、加速度センサが移動し始めると移動した方向の加速度が生じ、静止する際に移動した方向とは逆方向の加速度が生じるので、検出する加速度すべてを利用してメタファを作成すると、加速度センサの移動し始めは正しい方向にポインタが動くが、静止時には逆向きにポインタが移動してしまう問題がある。なので、利用する加速度を移動し始めた時の加速度のみを利用することで、ポインタを正しく操作できるようにする。

ポインタ操作方法は、ポインティングデバイスを装着した手の甲を上に向け、地面に対し平行な状態を静止状態の基準とし、基準から手の移動方向によるポインタ移動方向を表 3.4.2 に示す。

表 3.4.2 テーブルメタファの移動

手の移動方向	奥	手前	右	左
ポインタ移動方向	上	下	右	左

3.4.3 ディスプレイメタファ

第 2 章 2.3 節のタッチパネルを操作するようなモーション（地面に対し、x 軸:水平移動, y 軸:垂直移動）を空中で行い、手の移動による加速度を用いてポインタを制御する。ディスプレイメタファも 3.3.2 節と同じく、手の移動による加速度を利用するため、加速度センサの移動し始めた時の加速度のみを利用する。

ポインタ操作方法は、ポインティングデバイスを装着した手の甲を自分に垂直に向けた状態を静止状態の基準とし、基準から手の移動方向によるポインタの移動方法を表 3.4.3 に示す。

表 3.4.3 ディスプレイメタファの移動

手の移動方向	上	下	右	左
ポインタ移動方向	上	下	右	左

第4章　実験

4.1 実験システムの概要

本章は、第 3 章で述べたポインティングデバイスの特性と各メタファの操作性の検証を行うため、加速度センサを利用したグローブ型ポインティングデバイスを作成し、それを用いて実験を行う。作成したグローブ型ポインティングデバイスを図 4.1 に示す。

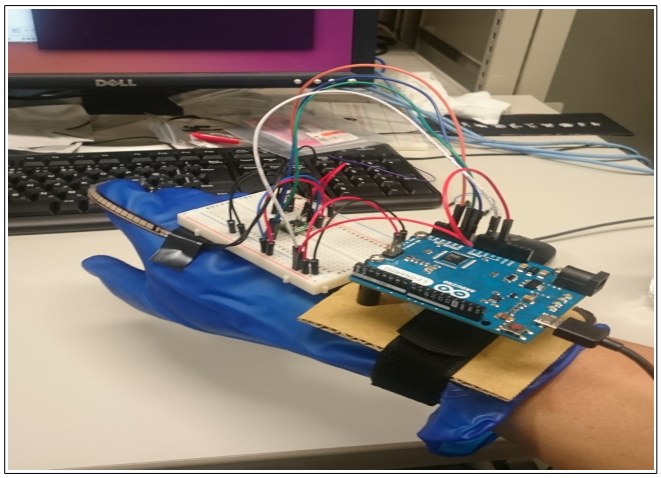

図 4.1 グローブ型ポインティングデバイス

作成したグローブ型ポインティングデバイスは arduino を手首にリストバンドで装着し、手の甲には、加速度センサの乗ったブレッドボードを貼っている。また、人差し指にブレッドボートから曲げセンサを伸ばして貼っており、曲げることで、クリックを行うことができる。

4.2 使用機材

前節で作成したグローブ型ポインティングデバイスでは、ポインタ制御に加速度センサを利用。そして、加速度センサから値を検出し、マウスカーソルの移動するための制御装置に Arduino を用いている。また、クリックの処理については曲げることで抵抗値が変化する曲げセンサ(Flex sensor)を用いた。

本節では作成したグローブ型ポインティングデバイスで使用した機材を述べる。

4.2.1 Arduino

本実験では Arduino Leonardo を用いた。Arduino とは、AVR マイコン、入出力ポートを備えた基板であり、Arduino 言語と呼ばれる開発言語と、それの統合開発環境から構成されるシステムである。Arduino 基板はマイクロコントローラーの I/O ピンのほとんどを他の回路で使えるようにそのまま解放されており、デジタル I/O ピンとアナログ入力ピンが利用可能である。ディジタル I/O ピンでは外部のスイッチの ON/OFF のディジタル値を得ることができたり、ディジタル値を出力し、LED を光らせ

るなどができる。アナログ入力ピンは、0V から数 V の連続的に変化するアナログ電圧を読み取ることができるため、センサを接続し計測処理が行える。電源電圧が 3.3V と 5V のピンがあり、用途に応じた電源電圧を使用できる。また、Arduino は USB ポートによる通信が可能である。図 4.2.1 に本実験で用いた Arduino Leonardo を示す。

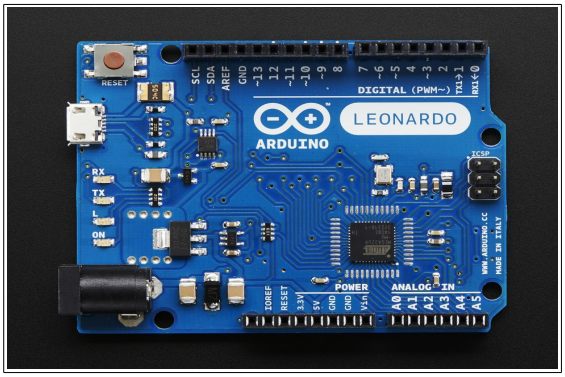

図 4.2.1 Arduino Leonardo

本実験では、加速度センサ・曲げセンサの値を取得して、制御するための制御装置として使用した。また、Arduino Leonardo はマウス・キーボードライブラリが組み込まれており、簡易ポインティングデバイスの作成に適しているため使用した。電源電圧は 5V を使用した。

4.2.2 加速度センサ

本実験では、Kionix 社製の加速度センサ KXR94-2050 を使用した。図 4.2.1 に本実験で用いた加速度センサ KXR94-2050 を示す。

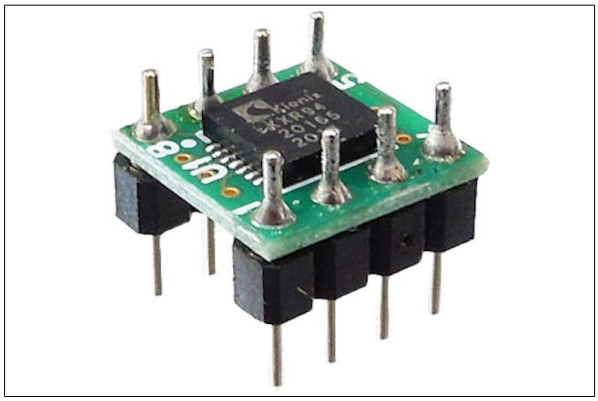

図 4.2.2 KXR94-2050

4.2.3 曲げセンサ

本実験では、spectra symbol 社製の曲げセンサ（Flex sensor）を用いた。本実験で用いた曲げセンサを図 4.2.3 に示す。図 4.2.3 のようにセンサを曲げることで抵抗値が変化する。この変化からセンサの曲がり具合を検出する。

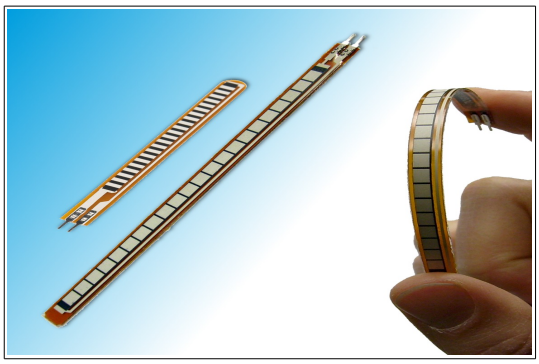

図 4.2.3 曲げセンサ

4.3 回路構築

実験システムを作成するため、ブレッドボード、ジャンプワイヤーを用いて回路を構築する。前節で示した機材を用いてブレッドボード上に構築した回路を図 4.3 に示す。[13]

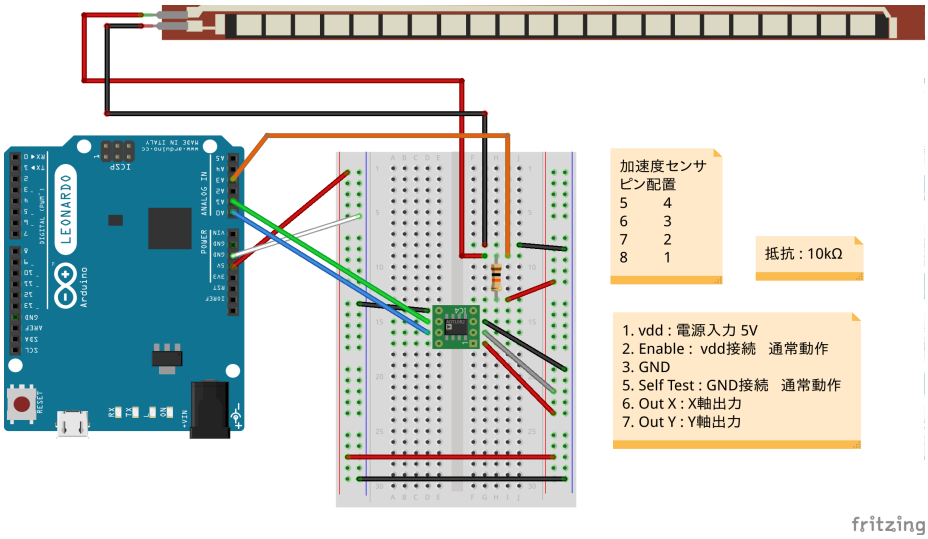

図 4.3 実験システムのブレッドボード図

4.4 開発環境

実験システムの開発環境を以下に示す。

表 4.4 開発環境

OS	Linux (Ubuntu 14.04)
開発ツール	Arduino IDE
開発言語	Arduino言語

Arduino 言語は C/C++をベースにしており、C 言語のすべての構造と、いくつかの C++の機能をサポートしている。また、AVR Libc にリンクされていて、その関数を利用できる。[14]

4.5 各メタファの加速度センサの設定

本節では、第 3 章 3.4 節で述べたポインティングメタファごとの加速度センサの設定について、以下の事前実験を行い、設定した。

4.5.1 十字キーメタファの設定

4.5.1.1 加速度センサの測定レンジ

十字キーメタファでは、傾きによりポインタを制御する。そのため、加速度センサを傾けた際の測定レンジを計測した。加速度センサを右（x 軸）・手前(y 軸)に傾けた時を+90 度(max)、左（x 軸）・奥（y 軸）に傾けた時を-90 度(min)とする。

表 4.5.1.1 十字キーメタファでの加速度センサの測定レンジ（mV）

	x軸	y軸
基準値(水平時)	506	505
+90度(max)	744	740
-90度(min)	280	275

4.5.2 テーブルメタファの設定

4.5.2.1 加速度センサの測定レンジ

テーブルメタファでは、マウスを操作するモーション（地面に対し x 軸 y 軸：水平移動）を空中で行い、ポインタを制御する。利用する値は手の移動し始める際に生じる加速度である。手の移動には人により個人差があるため、測定レンジを静止時の基準値から一定間隔を開けたものとし、レンジ外の値が検出された場合、測定レンジの max・min となるよう調整した。

表 4.5.2.1 テーブルメタファでの加速度センサの測定レンジ（mV）

	x軸	y軸
基準値(静止時)	506	505
max	756	755
min	256	255

4.5.2.2 4 方向移動による値の変化の測定

テーブルメタファでの静止基準状態で、右・左（x 軸）、奥・手前（y 軸）の 4 方向にそれぞれ移動させ、x 軸 y 軸の値が基準値からどのように変化したか計測した。

表 4.5.2.2 テーブルメタファでの4方向の移動による加速度センサの値の変化

移動方向	値の変化
右	小→大
左	大→小
奥	大→小
手前	小→大

表 4.5.2.2 より、x 軸の加速度センサの値が動き出しで小さくなった場合、加速度センサは右に移動したことがわかる。これより、x 軸が小さくなってから x 軸の基準値より大きい値を使用しなければ、右に正しくポインティングを行うことができる。しかし、これでは、最初に右に移動させてしまうと、左に移動させることができない。なので、本実験で作成したテーブルメタファは、150 ミリ秒静止状態にすることで、方向判定をリセットし、再び加速度センサの値の変化により、移動方向を判定するように調整した。

4.5.3 ディスプレイメタファの設定

4.5.3.1 加速度センサの測定レンジ

ディスプレイメタファでは、タッチパネルディスプレイを操作するようなモーション（地面に対し、x軸:水平移動, y 軸:垂直移動）を空中で行い、ポインタを制御する。検出する値はテーブルメタファ同様、手の移動する際に生じる加速度である。4.5.2.1 節と同様に測定レンジを静止時の基準値から一定間隔を開けたものとし、レンジ外の値が検出された場合、測定レンジの max・min となるよう調整した。

表 4.5.3.1 ディスプレイメタファでの加速度センサの測定レンジ（mV）

	x軸	y軸
基準値(静止時)	506	740
max	756	990
min	256	490

4.5.3.2 4 方向移動による値の変化の測定

ディスプレイメタファでの静止基準状態に、右・左（x 軸）、上・下（y 軸）の 4 方向にそれぞれ移動させ、x 軸 y 軸の値が基準値からどのように変化したか計測した。4.5.2 節のテーブルメタファの静止時の手の姿勢が手前に 90 度傾いているため、移動方向をテーブルメタファでの奥・手前がディスプレイメタファでの上・下である。

表 4.5.3.2 ディスプレイメタファでの4方向の移動による加速度センサの値の変化

移動方向	値の変化
右	小→大
左	大→小
上	大→小
下	小→大

4.5.2.2 節同様、本実験で作成したディスプレイメタファも、150 ミリ秒静止状態にすることで、方向判定をリセットし、再び加速度センサの値の変化により、移動方向を判定するように調整した。

4.6 実験方法

本節では、作成したポインティングデバイスで各メタファの操作性の検証実験について述べる。

4.6.1 特定のポインティング動作における所要時間の計測

各メタファの操作性を評価するため、HTML で作業用ページを作成し、指定したポインティングを被験者に行ってもらい、ポインティング終了までにかかる時間を計測する。図 4.6.1.1 に作成した作業用ページを示す。

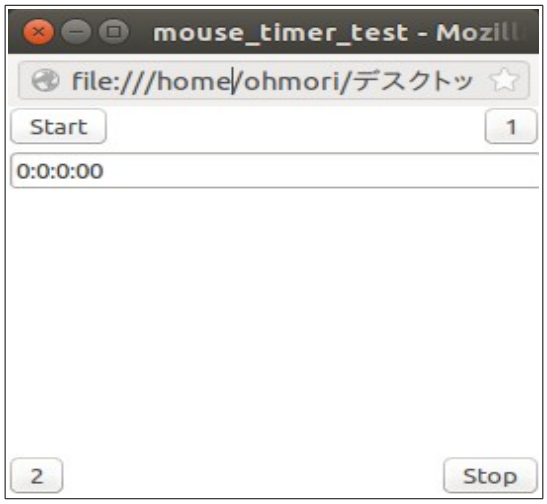

図 4.6.1.1 作業用ページ

作業用ページでは start ボタンをクリックすると時間計測を開始し、1 ボタン-2 ボタン-stop ボタンの順にクリックすることで時間計測を終了する。作業手順を図 4.6.1.2 を示す。なお実験は Fire fox ブラウザで作業用ページを開き、window サイズは 500×400 に設定して行う。

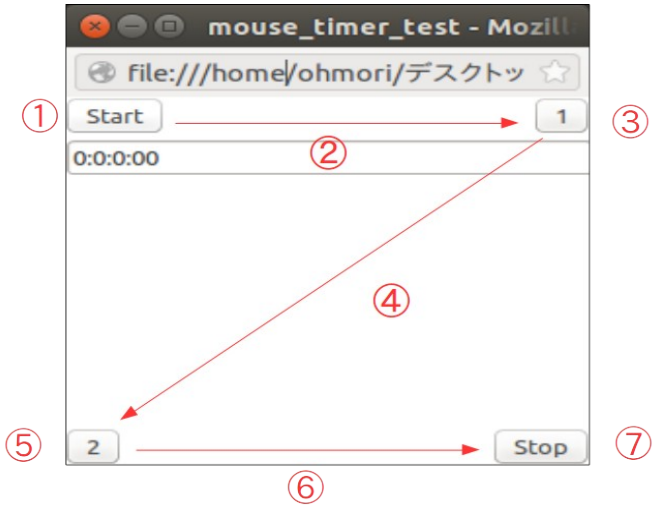

図 4.6.1.2 作業手順

① Start ボタンをクリックする(時間計測開始)。
② 1 ボタンまでポインタを移動させる。
③ 1 ボタンをクリックする。
④ 2 ボタンまでポインタを移動させる。
⑤ 2 ボタンをクリックする。
⑥ Stop ボタンまでポインタを移動させる。
⑦ Stop ボタンをクリックする(時間計測終了)。

4.6.2 被験者による主観的評価

4.6.1 節の実験を終了した被験者に対し、5段階評価で以下のアンケートを取り、主観的評価を行う。

・使いやすさ　5:使いやすい - 4:少し使いやすい - 3:どちらともいえない -
2 :少し使いにくい - 1:使いにくい

・なめらかさ　5:なめらかに動く - 4:まあまあなめらかに動く - 3:どちらともいえない -
2:あまりなめらかに動かない - 1: なめらかに動かない

4.7 実験結果

被験者 6 人に対し 4.6 節の実験を行ったところ、実験結果は以下のようになった。

4.7.1 十字キーメタファの実験結果

表 4.7.1a 十字キーメタファ実験の所要時間(秒)

被験者	1回目	2回目	3回目	平均値
A	7.17	5.27	8.87	7.10
B	11.52	5.73	5.21	7.49
C	9.62	10.26	8.73	9.54
D	10.19	10.69	7.22	9.37
E	10.31	11.5	9.9	10.57
F	12.16	9.33	10.67	10.72
平均値	10.16	8.80	8.43	9.13

表 4.7.1b 十字キーメタファの評価

被験者	使いやすさ	なめらかさ
A	5	4
B	4	4
C	2	3
D	1	4
E	2	4
F	2	3
平均	2.67	3.67

十字キーメタファについて表 4.7.1a を見ると、実験にかかった時間の平均は 9.13 秒であった 被験者 A 以外のどの被験者も実験1回目より3回目のほうが実験にかかった時間が短いことが分かる。また、被験者全員の 1 回目、2 回目、3 回目の所要時間の平均は回数を重ねるごとに短くなっていることが分かる。このことから、十字キーメタファは操作に慣れやすく、ある程度安定した操作が行えるのだと考えられる。次に、表 4.7.1b を見ると、使いやすさは 3 を下回ったが、なめらかさは 3 を上回った。

4.7.2 テーブルメタファの実験結果

表 4.7.2a テーブルメタファ実験の所要時間(秒)

被験者	1回目	2回目	3回目	平均値
A	20.27	15.38	28.93	21.53
B	17.92	17.68	38.78	24.79
C	15.6	33.2	16.87	21.89
D	27.96	40.02	23.73	30.57
E	39.23	30.8	21.43	30.49
F	47.93	28.25	42.23	39.47
平均値	28.15	27.56	28.66	28.12

表 4.7.2b テーブルメタファの評価

被験者	使いやすさ	なめらかさ
A	1	1
B	1	1
C	1	1
D	1	1
E	1	1
F	1	1
平均	1	1

テーブルメタファについて表 4.7.2a を見ると、実験にかかった時間の平均は 28.12 秒であった。また、実験1回目より3回目のほうが実験にかかった時間が短い被験者は 6 名中 3 名であり、被験者全員の 1 回目、2 回目、3 回目の所要時間の平均は回数を重ねるごとに短くなることはなかった。このことから、テーブルメタファは操作は慣れにくく、安定しないのだと考えられる。次に、表 4.7.2b をみると、被験者全員が使いやすさ、なめらかさ共に最低評価1をつけている。

4.7.3 ディスプレイメタファの実験結果

表 4.7.3a ディスプレイメタファ実験の所要時間(秒)

被験者	1回目	2回目	3回目	平均値
A	11.67	23	21.73	18.80
B	12.84	29.2	18.19	20.08
C	33.26	31.49	27.35	30.70
D	20.91	29.72	15.98	22.20
E	14.79	10.68	15.72	13.73
F	29.77	22.6	36.48	29.62
平均値	20.54	24.45	22.58	22.52

表 4.7.3b ディスプレイメタファの評価

被験者	使いやすさ	なめらかさ
A	1	1
B	1	1
C	1	1
D	1	1
E	2	1
F	1	1
平均	1.17	1

ディスプレイメタファについて表 4.7.3a を見ると、実験にかかった時間の平均は 22.52 秒であった。また、実験1回目より3回目のほうが実験にかかった時間が短い被験者は 6 名中 2 名であり、被験者全員の 1 回目、2 回目、3 回目の所要時間の平均は回数を重ねるごとに短くなることはなかった。このことから、ディスプレイメタファは操作に慣れにくく、安定しないのだと考えられる。次に、表 4.7.3b をみると、被験者 E 以外のどの被験者も使いやすさ、なめらかさ共に最低評価1をつけている。

4.8 考察

表 4.7.1a, 表 4.7.2a, 表 4.7.3a より、各メタファの実験にかかった時間の平均を比較すると、十字キーメタファは平均 9.13 秒で実験が終了したのに対し、テーブルメタファは平均 28.12 秒と約 3.08 倍、ディスプレイメタファは平均 22.52 秒と約 2.48 倍もの時間がかかっていることがわかる。これは、テーブルメタファとディスプレイメタファが移動し始めの加速度を利用してポインタを動かしているため、「常に」動かし続けるということができないため、細かな位置調整や斜め方向の移動が安定して行うことが困難だと考えられ、その結果、「なめらかさ」を欠いてしまっていると考えられる。表 4.7.2b, 表 4.7.3b を見ても、テーブルメタファとディスプレイメタファは、被験者による主観的評価における「なめらかさ」の評価で全員が最低評価 1 をつけており、「使いやすさ」においても、ほとんどの被験者が使いにくいと答えた。

一方、十字キーメタファは、表 4.7.1a, 表 4.7.1b を見ると、実験にかかった時間は前述した通り、他のメタファに比べ 2 倍以上早く終えることができ、被験者による主観的評価においても、「使いやすさ」の平均は評価 3 を下回ったが、「なめらかさ」の平均は評価 3 を上回っており、それなりの評価を得ることができた。これは、十字キーメタファは、加速度センサを傾けることで、傾きに応じた値を検出し続けることができるため、傾けた方向へ傾けている限り「常に」動かし続けることが可能である。したがって、ある程度なめらかに動かすことができたと考えられる。また、表 4.7.1b より、使いやすさについて見ると、どうしても既存のマウスとの操作性と比較してしまうため、あまり良い結果にならなかったのだと考えられる。さらに、被験者ごとの「使いやすさ」の評価が 5 から 1 まであり、評価で個人による操作性の差が見られた。しかし、表 4.7.1a より、被験者ごとに 1 回目の実験でかかった時間と 3 回目の実験でかかった時間を比較すると、どの被験者も 1 回目より早く実験を終えることができたことがわかる。また、被験者全員の 1 回目、2 回目、3 回目の所要時間の平均は回数を重ねるごとに短くなっていることが分かることから、十字キーメタファによるポインティングは慣れが見て取れた。そのため、もっと実験回数を増やすことで、実験後の評価も上がるのではないかと考えられる。

これらの結果より、今回作成した加速度センサを利用したグローブ型ポインティングデバイスにおけるポインティングメタファとして、傾きを利用した十字キーメタファによる操作が最も適したポインティングメタファであると判断した。また、残り 2 つのメタファは、加速度センサを用いてのポインタ制御は不向きだと判断し、今後、この2つのメタファでポインティングを行う場合は、別のセンサを利用するべきであると考える。

第 5 章　まとめ

本論文では、工事現場のような並行作業の多い環境下で使用しやすいポインティングデバイスとして、ポインタの制御に加速度センサを利用したグローブ型ポインティングデバイスを提案し、クリック処理に曲げセンサ、制御装置として Arduino Leonardo を用いて、カーソル移動とクリックができる程度のグローブ型簡易ポインティングデバイスを作成した。そして、作成したポインティングデバイスに合った操作法の検討として、手をゲームパッドの十字キーに見立て、手を傾けることでポインタを操作する十字キーメタファ、マウス・タッチパネルディスプレイを操作するモーションを空中で行い、手の移動による加速度を用いてポインタを操作するテーブルメタファ・ディスプレイメタファの計 3 種類のポインティングメタファをを用意し、どのメタファが最適な操作法であるか検討を行った。その結果、作成したポインティングデバイスは、手の傾きを利用した十字キーメタファが最も操作しやすいことがわかった。

今後の課題として、本論文で提案する加速度センサを利用したグローブ型ポインティングデバイスを実用的に使用するには、デバイスの小型化・軽量化や、無線化、操作性のさらなる向上が考えられる。

謝辞

本論文を作成するにあたり、ご指導いただきました三好力教授に深謝致します。また、多忙の中、本研究において多くの御助言をいただきました研究室の皆様に深く感謝いたします。

参考文献

[1]. マウスの構造と仕組み
http://www.kazumin.org/kazumi/exercise/system/mouse.html

[2]. マウスの動作原理
http://www.sugilab.net/jk/joho-kiki/1302/index.html

[3]. 防水・防塵マウス MightyMouse 5
http://www.attainj.co.jp/ja/prd/hdw/might5.html

[4]. ジャイロセンサとは
http://www5.epsondevice.com/ja/information/technical_info/gyro/

[5]. 空中に掲げてマウスの操作をフルに行えるトラックボール「airm01w」
http://www.oshiete-kun.net/archives/2015/06/airm01w.html

[6]. 現場用にいかが?指に取り付ける超小型マウス
http://kenplatz.nikkeibp.co.jp/article/it/column/20101118/544298/

[7]. サンワダイレクト、指に装着してジェスチャー操作できる指マウス
http://yukan-news.ameba.jp/20141121-156/

[8]. windows7 新時代：第 5 回　タッチ・インターフェイスで使う Windows7
http://www.atmarkit.co.jp/ait/articles/0911/19/news107_2.html

[9].【続報：wii 分解】「センサーバー」にはセンサがなかった – 家電・モバイル - 日経テクノロジーオンライン
http://techon.nikkeibp.co.jp/article/NEWS/20061127/124506/?rt=nocnt

[10]. モーションセンサーで組み込み機器はどう変わる?- MONOist
http://monoist.atmarkit.co.jp/mn/articles/1402/24/news086.html

[11]. Kinect プログラミングの道しるべ, 橋本 直
http://www-ui.is.s.u-tokyo.ac.jp/~hiraoka/media2011/pdf/Kinect_111118.pdf

[12]. しっかり分かる「センサーの活用法」- EDN Japan
http://ednjapan.com/edn/articles/1205/16/news110.html

[13]. 曲がり具合を測る（曲げセンサ使用）
www.hiramine.com/physicalcomputing/arduino/bendsensor.html

[14]. Arduino リファレンス
http://garretlab.web.fc2.com/arduino_reference/language/functions/leonardo_specific/index.html

[15]. 加速度センサ, 角速度センサのしくみ, 上田智章
http://www.cqpub.co.jp/dwm/contents/0117/dwm011700700.pdf

[16]. MEMS 加速度センサの選び方,使い方 , 大内篤
www.cqpub.co.jp/dwm/contents/0126/dwm012600970.pdf

[17].「Arduino で計る,測る,量る」, CQ 出版社, 2012.

付録

○十字キーメタファ

```
const int xAxis = A0;         // X 軸用アナログセンサ(7pin)
const int yAxis = A1;         // Y 軸用アナログセンサ(6pin)
const int vol_pin = A3; //曲げセンサ

int range =12;               // X/Y の移動の出力範囲
int responseDelay = 10;       // マウスの反応の遅延。単位はミリ秒
int threshold = range/12;     // 静止状態を表す閾値
int center = range/2;         // 出力幅の中心
int minima[] = {
  280, 275};                 // analogRead()の最小値（X 軸=280, Y 軸=275）
int maxima[] = {
  744, 740};                   // analogRead()の最大値（X 軸=744 , Y 軸=740）
int axis[] = {
  xAxis, yAxis};             // X 軸・Y 軸を制御するためのアナログピン番号



void setup() {
 Mouse.begin();
}

void loop() {

// 2 つの軸の値を読み取る
  int xReading = readAxis(0);
  int yReading = readAxis(1);

// クリック判定
  int vol_value = 0;
  vol_value = analogRead( vol_pin );

  if(vol_value > 620){
    Mouse.click();
    delay(400);
  }

// マウスを動かす
    Mouse.move(xReading, yReading, 0);
    delay(responseDelay);
}

/*加速度センサ読み取り*/
int readAxis(int axisNumber) {
  int distance = 0;   // 出力幅の中心からの距離
  // アナログ入力を読み取る
  int reading = AnalogReadAve(axisNumber);

  // アナログ入力値を出力幅に変換する
  reading = map(reading, minima[axisNumber], maxima[axisNumber], 0, range);

 // 出力値がスレッショルドよりも大きければその値を使う
  if (abs(reading - center) > threshold) {
    distance = (reading - center);
  }

  // 軸の移動距離を返す
  return distance;
}


/* 加速度センサ読み取り*/
int AnalogReadAve(int axisNumber){
  int Areading =0;
  int result = 0;

  for(int count=0 ; count<10 ; count++){ //10 回測定して加算する
    Areading += analogRead(axis[axisNumber]);
  }

  result = Areading / 10;

  return result;
}
```

○テーブルメタファ

```
const int xAxis = A0;         // X 軸用アナログセンサ(7pin)
const int yAxis = A1;         // Y 軸用アナログセンサ(6pin)
const int vol_pin = A3; //曲げセンサ

int range =12;               // X/Y の移動の出力範囲
int responseDelay = 3;       // マウスの反応の遅延。単位はミリ秒
int threshold = range/12; // 静止状態を表す閾値
int center = range/2;         // 出力幅の中心
int minima[] = {
  256, 255};                 //analogRead()の最小値（X 軸 256= , Y 軸=255）
int maxima[] = {
  756, 755};                   // analogRead()の最大値（X 軸=756 , Y 軸=755）
int axis[] = {
  xAxis, yAxis};             // X 軸・Y 軸を制御するためのアナログピン番号
/*table_meta_line*/
int UtoD = 550; //動作ライン
int DtoU = 460; //動作ライン
int Flag[] = {0,0} ; //flag 初期値
int Flag_count[] = {0,0}; //count 初期値

void setup() {
 Mouse.begin();
}

void loop() {

// 2 つの軸の値を読み取り縮小する
  int xReading = readAxis(0);
  int yReading = readAxis(1);

// クリック判定
  int vol_value = 0;
  vol_value = analogRead( vol_pin );

  if(vol_value > 620){
    Mouse.click();
    delay(400);
  }

// マウスを動かす
    Mouse.move(xReading, yReading, 0);
    delay(responseDelay);
}

/*加速度センサ読み取り*/
int readAxis(int axisNumber) {
  int distance = 0;   // 出力幅の中心からの距離

  // アナログ入力を読み取る
  int reading = AnalogReadAve(axisNumber);

if(Flag[axisNumber] == 0){

  //動き出し加速度判定
  if(reading >= UtoD){
    Flag[axisNumber] = 1; //up
  }

  if(reading <= DtoU){
   Flag[axisNumber] = 2; //down
  }

}

  // アナログ入力値を出力幅に変換する
  reading = map(reading, minima[axisNumber], maxima[axisNumber], 0, range);


  //利用する加速度処理
  switch(Flag[axisNumber]){

    case 1:
     // 出力値がスレッショルドよりも大きければその値を使う
      if ((reading - center) > threshold) {
        distance = (reading - center);
        Flag_count[axisNumber] = 0;
      }
      else{
        Flag_count[axisNumber] += 1;

      }
      break;

    case 2:
       // 出力値がスレッショルドよりも大きければその値を使う
      if ((center - reading) > threshold) {
       distance = (reading-center);
        Flag_count[axisNumber] = 0;
      }

      else{
        Flag_count[axisNumber] += 1;
      }
```

```
    break;

    default:
      break;
  }

  //静止状態が 50 回続いたとき
 if(Flag_count[axisNumber] >= 50){
   Flag[axisNumber] = 0;
   Flag_count[axisNumber] = 0;
 }

 // 正負を逆転させる
   if(distance != 0){
   distance = -distance;
  }

 // 軸の移動距離を返す
 return distance;
}

/* 加速度センサ読み取り*/
int AnalogReadAve(int axisNumber){
 int Areading =0;
 int result = 0;

 for(int count=0 ; count<10 ; count++){ //10 回測定して加算する
   Areading += analogRead(axis[axisNumber]);
  }

 result = Areading / 10;

 return result;
}
```

○ディスプレイメタファ

```
const int xAxis = A0;        // X 軸用アナログセンサ(7pin)
const int yAxis = A1;        // Y 軸用アナログセンサ(6pin)
const int vol_pin = A3; //曲げセンサ

int range =12;               // X/Y の移動の出力範囲
int responseDelay = 3;       // マウスの反応の遅延。単位はミリ秒
int threshold = range/12;    // 静止状態を表す閾値
int center = range/2;        // 出力幅の中心
int minima[] = {
 256, 490};                  // analogRead()の最小値（X 軸=256 , Y 軸=490）
int maxima[] = {
 756, 990};                  // analogRead()の最大値（X 軸=756 , Y 軸=990）
int axis[] = {
 xAxis, yAxis};              // X 軸・Y 軸を制御するためのアナログピン番号
/*水平移動(x 軸)*/
int UtoD = 570; //動作ライン
int DtoU = 430; //動作ライン
/*垂直移動(y 軸)*/
int wUtoD = 790; //動作ライン(display)
int wDtoU = 690; //動作ライン(display)
int Flag[] ={0,0}; //flag 初期値
int Flag_count[] = {0,0}; //count 初期値


void setup() {
 Mouse.begin();
}

void loop() {

// 2 つの軸の値を読み取り縮小する
 int xReading = readAxis(0);
 int yReading = readAxis(1);

// クリック判定
 int vol_value = 0;
 vol_value = analogRead( vol_pin );

 if(vol_value > 620){
   Mouse.click();
   delay(400);
 }


// マウスを動かす
   Mouse.move(xReading, yReading, 0);
   delay(responseDelay);
}

/*加速度センサ読み取り*/
int readAxis(int axisNumber) {
 int distance = 0;   // 出力幅の中心からの距離

 // アナログ入力を読み取る
 int reading = AnalogReadAve(axisNumber);

if(Flag[axisNumber] == 0){

 //動き出し加速度判定
   if(axisNumber == 0){ //x 軸
    if(reading >= UtoD){
     Flag[axisNumber] = 1; //up
    }

     if(reading <= DtoU){
     Flag[axisNumber] = 2; //down
    }
   }

  if(axisNumber == 1){ //y 軸
    if(reading >= wUtoD){
    Flag[axisNumber] = 1; //up
    }

  if(reading <= wDtoU){
   Flag[axisNumber] = 2; //down
    }
  }
}



 // アナログ入力値を出力幅に変換する
 reading = map(reading, minima[axisNumber], maxima[axisNumber], 0, range);


 //利用する加速度処理
 switch(Flag[axisNumber]){

  case 1:
   // 出力値がスレッショルドよりも大きければその値を使う
   if ((reading - center) > threshold) {
     distance = (reading - center);
      Flag_count[axisNumber] = 0;

   }
   else{
     Flag_count[axisNumber] += 1;
   }
   break;

  case 2:
    // 出力値がスレッショルドよりも大きければその値を使う
   if ((center - reading) >threshold) {
    distance = (reading - center);
     Flag_count[axisNumber] = 0;
   }
   else{
     Flag_count[axisNumber] += 1;
   }
   break;

   default:
     break;
 }


  /*静止状態が 50 回続いたとき*/
if(Flag_count[axisNumber] == 50){
  Flag[axisNumber] = 0;
  Flag_count[axisNumber] = 0;
}

 // 正負を逆転させる

  if(distance != 0){
  distance = -distance;
 }


 // 軸の移動距離を返す
 return distance;
}


/*加速度センサ読み取り*/
int AnalogReadAve(int axisNumber){
 int Areading =0;
 int result = 0;

 for(int count=0 ; count<10 ; count++){ //10 回測定して加算する
   Areading += analogRead(axis[axisNumber]);
 }

 result = Areading / 10;

 return result;
}
```

○作業用ページ HTML

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=utf8">
<meta name="GENERATOR" content="Microsoft FrontPage Express 2.0">
<title>mouse_timer_test</title>
</head>
<body>
<script type="text/javascript"><!--
window.resizeTo(300, 300);
var num = 0;
function aaaaa(a){

  if (a==1 && num == 0){      // Start ボタンを押した

      myStart=new Date();  // スタート時間を退避
//    document.myForm.myFormButton.value = "Stop!";
      myInterval=setInterval("myDisp()",1);

      num = 1;

  }else if(a == 2 && num == 1){
      num = 2;
  }else if(a == 3 && num == 2){
      num = 3;
  }else if(a == 4 && num == 3){                // Stop ボタンを押した
   // document.myForm.myFormButton.value = "Start";
    myDisp();
    clearInterval( myInterval );
    num = 0;
  }else{
    //alert("手順違う");
  }
}

function myDisp(){
    myStop=new Date();  // 経過時間を退避
    myTime = myStop.getTime() - myStart.getTime(); // 通算ミリ秒計算
    myH = Math.floor(myTime/(60*60*1000));         // '時間'取得
    myTime = myTime-(myH*60*60*1000);
    myM = Math.floor(myTime/(60*1000));            // '分'取得
    myTime = myTime-(myM*60*1000);
    myS = Math.floor(myTime/1000);                 // '秒'取得
    myMS = myTime%100;                             // 'ミリ秒'取得

document.myForm.myFormTime.value = myH+":"+myM+":"+myS+":"+myMS;
}

// --></script>
<form name="myForm">
<input type="text" size="28" value="0:0:0:00" name="myFormTime" style="position:
absolute; left: 0px; top: 30px" / onclick="aaaaa()">
<input type="button" value="Start" name="myFormButton_start" style="position:
absolute; left: 0px; top: 0px"/ onclick="aaaaa(1)">
<input type="button" value="1" name="myFormButton1" style="position: absolute; right:
0px; top: 0px"/ onclick="aaaaa(2)">
<input type="button" value="2" name="myFormButton2" style="position: absolute; left:
0px; bottom: 0px"/ onclick="aaaaa(3)">
<input type="button" value="Stop" name="myFormButton_end" style="position: absolute;
right: 0px; bottom: 0px"/ onclick="aaaaa(4)">

</form>
</body>
</html>
```

○作業用ページサイズ設定

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf8">
<meta http-equiv="Content-Script-Type" content="text/javascript">
<title>window_resize</title>
<script type="text/javascript">
<!--
function openwin() {
                window.open("./mouse_timer_test.html", "", "width=500,height=400");
}
// -->
</script>
</head>
<body>
<a href="javascript:openwin();">500*400_resize</a>
</body>
</html>
```