*Information technology researching society*
*of the Gunmer, by the Gunmer, for the Gunmer*

Vol. 03
Comic Market 89 版

# 群馬大学電子計算機研究会

# Lollipop

IGGG
http://www.iggg.org/

群馬大学電子計算機研究会　著

群馬大学電子計算機研究会 IGGG "Lollipop" Vol. 03 Comic Market 89 版

# 目次

# Haskellによるアルゴリズム入門

ひげ

## 概要

今，マイブームである Haskell をアルゴリズムと絡めて紹介しようと思います．ちなみに，Haskell 自体は昨年の春から勉強しています．内容としては，Haskell とアルゴリズムとして著名な書籍,「関数プログラミング 珠玉のアルゴリズムデザイン [1]」から，簡単な例題をとってきて，もっと初学者向けに説明したいと思います．

## 1　アルゴリズムと Haskell

### 1.1　アルゴリズムとは

そもそもアルゴリズムとは何かと言うと,「問題を解くための手順を定式化した形で表現したモノ」(wikipedia より) だそうだ．要するに，(コンピュータサイエンスの分野では) プログラミングテクニックの一つと言えるだろう．アルゴリズムの目的は，そのプログラムの速度を上げるコトである．アルゴリズムを工夫することで，本来は現実的な時間で解くことのできない問題でも解くことが出来るようになる．

### 1.2　計算量オーダー

アルゴリズムの評価の 1 つに計算量オーダーと言うものがある．これは，アルゴリズムが問題を解くのにかかる時間を表している．しかし，ただどれくらいかかるかでは無く，要素数 $n$ に対しどのように変化するのかを表している．例えば次のようなものがある．

| | | |
|---|---|---|
| $O(1)$ | 定数時間 | 実行時間が要素数に依存しない |
| $O(n)$ | 線形時間 | 実行時間が要素数に比例する |
| $O(n \log n)$ | 線形対数 | ソートアルゴリズムの限界 |
| $O(n^2)$ | 二乗関数 | 2 重ループを書くと大抵コレ |

### 1.3　Haskell とは

Haskell とは純粋関数型プログラミング言語で，関数プログラミング言語の代表として 1990 年に作成された．関数型プログラミングとは問題を数学的な写像 (関数) と，その関数合成として記述する．特徴として，強力な型システムやデフォルトで遅延評価であることが挙げられる．また，純粋が故に入出力のような副作用を持つ処理が多少苦手である．

### 1.4　アルゴリズムを Haskell で議論する利点

珠玉のアルゴリズムデザインには，主に以下の 2 つだと書いてあった．

- そもそもアルゴリズムは数学的であり関数型の方が素直に表現できる
- 運算を利用して単純だが効率の悪いプログラムから複雑だが効率の良いプログラムを導出できる

運算とは等式論証を用いて数式を変換していくコトである．Haskell ならばプログラムを 1 つの等式としてみなすことが出来るので，相性が非常に良い．

逆に欠点として Haskell が配列を扱いにくいというコトが挙げられる．Haskell は再代入を行うことが出来ない．故に，挿入・更新・変換が定数時間 (配列の大きさに依存しない) で行える配列を素直に表現することが出来ないのである．

## 2　最小自由数

### 2.1　最小自由数とは

この問題は珠玉のアルゴリズムデザインで最初に紹介されていた問題で，与えられた自然数の有限集合 $X$ に**含まれない**最小の自然数を求める問題である．この問題の解法と言うよりかは難度は $X$ の表現によってもちろん異なる．$X$ の要素に重複のない昇順リストであれば簡単で，単に最初の隙間を見つければ良いだろう．しかし今回は，要素に重複のないが $X$ に特定の順序が付いていないリストで与えられていると仮定する (要素は自然数なので順序を持つ，整列されてないだけ．)．例えば次のようなリストが考えられる．

$$[5, 20, 12, 3, 8, 15, 27, 9, 33, 4, 1, 17, 19, 0, 2, 22, 6, 11]$$

この場合の最小自由巣は 7 だ．最も簡単な方法は，一度ソートアルゴリズムを適用して整列してしまえば良い．しかし，ソートアルゴリズムは最速で $O(n \log n)$ かかる．つまり，その解法では線形時間で解くことが出来ない．にもかかわらず，この問題は線形時間で解くことのできる解法があることが知られている．それを Haskell を利用して紹介しよう．

## 2.2 素朴な解法

すごく簡単に考えると以下のような関数を考えれば解くことが出来る.

```
minfree :: [Int] -> Int
minfree xs = head ([0..] \\ xs)
```

Haskell になじみのない人のために補足すると,

- 1 行目は式名と式の型を定義している. ::以降が型.
- この場合,“ Int のリスト型から Int 型への関数型 ”となる.
- つまり,矢印は写像を意味し,各括弧はリスト型 (配列ではなく線形リスト) を表している.
- 2 行目以降に式の定義を書く.
- 左辺にある xs は引数を表現しており,右辺で束縛して利用できる.
- head はリストから先頭の要素をとる関数
- [0..] は 0 からの無限整数列
- as \\ bs はリスト as からリスト bs の要素を除いたリストを返す関数

無限リストを利用しているが,Haskell はデフォルトで遅延評価なため,この関数は実行可能である. しかし,最悪の場合の計算量は $O(n^2)$ である (\\でそれだけの時間がかかる).

## 2.3 今回の鍵

今回の問題を効率よく解くうえで鍵となる事実は,[0..(length xs)] の範囲にある数の内少なくとも 1 つは xs に含まれていないコトである (length xs は xs の要素数を返す. [0..n] は n を含むリストなので仮に xs が 0 から隙間なく並んでいても length xs が xs に含まれない. ). よって,最小自由数は filter (<= length xs) xs に含まれない最小の数となる (filter f xs は xs の要素のうち,述語 f が真となる要素のリストを返す関数. ).

## 2.4 配列を利用した解法

この解法では先述した鍵を利用して [0..(length xs)] に xs の要素が含まれているか否かのチェックリストを配列で作る. 要するにこんな感じである (これは定義済みと仮定して GHCi で実行した結果である)

```
*Main> :t makeChecklist
makeChecklist :: [Int] -> Array Int Bool
*Main> makeChecklist [3,8,4,0,6]
array (0,5) [(0,True),(1,False),(2,False),(3,True),(4,
    True),(5,False)]
```

Haskell の配列にはいくつか種類があり,今回利用するのは IArray と言われるものである. IArray は参照が定数時間で行えるが更新には線形時間かかるので,C 言語の配列とは異なる (今回は関係ないが...). 配列型は Array i a となっており,i はインデックスになる型を,a は要素の型を表す (但し,インデックス型は何でもよいわけではなく Ix 型クラスのインスタンスである必要がある. )

線形時間で配列を生成するために Data.Array にある生成関数を利用する. 今回は accumArray を利用する. accumArray は次のような型定義となっている

```
accumArray :: Ix i  -- i型はIx型クラスのインスタンス
  => (e -> a -> e) -- 初期化関数
  -> e             -- 初期値
  -> (i, i)        -- インデックスの範囲
  -> [(i, a)]      -- 配列化したいリスト
  -> Array i e
```

(恐らく) もっと基本的な関数で定義するとこんな感じだろう.

```
accumArray f init indexs xs
  = array indexs (map (\(a,b) -> (a,f init b)) xs)
```

- array は最も基本的な配列の生成関数で,インデックスの範囲とインデックスと要素のタプルのリストを与えれば配列に変換してくれる.
- (\x -> x) はラムダ式 (無名関数) である.
- map はリストの各要素に 1 引数目の関数を適用する高階関数である.

array も map も線形時間で終わる (らしい) ので accumArray は線形時間で終わる. makeChecklist は accumArray を利用して次のように定義できた.

```
makeChecklist :: [Int] -> Array Int Bool
makeChecklist xs = accumArray (||) False (0,n) xs'
    where
      xs' = zip (filter (<= n) xs) (repeat True)
      n = length xs
```

- where 以下は局所的な式定義である.
- zip は 2 つのリストを貰ってタプルのリストに変換する関数 (リストの短い方に丸める)
- repeat は引数を要素にした無限リストを返す
- ||は論理和

これを利用して minfree は次のようになる.

```
minfree =  length . takeWhile id . elems . makeChecklist
```

- . は関数合成である
- elems は配列の要素だけのリストを返す.
- takeWhile はリストに与えられた関数を適用して偽になるまでの部分リストを返す
- id は引数をそのまま返す関数

つまり takeWhile id で最初の False までのリストを返し，この要素数が最小自由数に等しい．そして，これは全て線形時間の関数の関数合成で出来ているので線形時間である．

この方法の利点の 1 つに重複する要素の場合でも簡単に実装できる点がある．次のようにすれば良い．

```
minfree3
  = length . takeWhile (/= 0) . elems . makeCountlist

makeCountlist :: [Int] -> Array Int Int
makeCountlist xs = accumArray (+) 0 (0,n) xs'
    where
      xs' = zip xs (repeat 1)
      n = maximum xs
```

## 2.5 分割統治法

今度は分割統治法を用いて実装する．分割統治法とは，問題をそれよりも小さいサイズの問題に分割して，それを解いた後に連結していくというような手法である．分割統治法で考えるべきなのは大きく 2 つで，分割の仕方と連結の仕方である．今回は，素朴な方法をベースにして分割統治法をステップバイステップに適用していく．

素朴の解法で肝になっているのは差集合である．そこでまず，差集合の性質を列挙した．

$$(as \cup bs) - cs = (as - cs) \cup (bs - cs)$$
$$as - (bs \cup cs) = (as - bs) - cs$$
$$(as - bs) - cs = (as - cs) - bs$$

さらに，$xs, ys$ が互いに素 (お互いに同じ要素を持たないコト) の場合

$$xs - ys = xs$$

である．以上の性質より，$as$ と $us$，$bs$ と $vs$ が互いにその場合次のことが成り立つ

$$\begin{aligned}&(as \cup bs) - (us \cup vs)\\ &= (as - (us \cup vs)) \cup (bs - (us \cup vs))\\ &= (as - us - vs) \cup (bs - us - vs)\\ &= (as - vs) \cup (bs - us)\end{aligned}$$

差集合演算子と和集合演算子をそれぞれ Haskell の関数に置き換えて，素朴な解法に代入すれば次のようになる．

```
minfree xs = head ([0..(b-1)] \\ us) ++ ([b..] \\ vs)
  where (us,vs) = partition (< b) xs
```

もちろん，これではまだ動作しない．b は length xs 中の任意の数である．partition p は与えたリストを述語 p を満たす要素と満たさない要素で分割してくれる効率的な関数である．よって，[0..(b-1)] と us，[b..] と vs は互いに素なので，素朴な解法と等価である．

次に，head (xs++ys) を定義する．

```
head (xs++ys) = head (if null xs then ys else xs)
```

である (null は空のリストであれば真を返す)．よって

```
minfree xs = if null ([0..(b-1)] \\ us)
             then head ([b..] vs)
             else head ([0..] us)
  where (us,vs) = partition (< b) xs
```

である．ここで，null ([0..(b-1)] \\ us) に結局 $O(n^2)$ だけの時間がかかってしまうことがわかる．しかし，この判定用の式は次の式と等価である．

```
length us == b
```

length が線形時間かかるので，これは線形時間で解ける．

次に，再帰用の関数を定義する．上述の minfree のコードより，次のように一般化 (というか再帰関数を定義) できることがわかる．

```
minfrom :: Int -> [Int] -> Int
minfrom a xs = head ([a..] \\ xs)
```

これを利用して minfree は次のように定義できる．

```
minfree = minfrom 0
    where
      minfrom a [] = a
      minfrom a xs
        | length xs == b - a = minfrom b vs
        | otherwise          = minfrom a us
        where (us,vs) = partition (< b) xs
```

後は b を定義できればよい．b は a より大きくないといけないのく，できるだけ均等になると良いので

```
b = a + 1 + n `div` 2
```

となる (一番時間がかかるのは xs が密に詰まっていた場合である．その場合 length us を 1/2 ずつ行っていくので $O(n/2 + n/4 + n/8 + ...) = O(n)$ となる．)

vs の要素数を一工夫して，最終的に次のようになった．

```
minfree xs = minfrom 0 (length xs, xs)
    where
      minfrom a (_,[]) = a
      minfrom a (n,xs)
        | m == b - a = minfrom b (n-m,vs)
        | otherwise  = minfrom a (m,us)
        where
          (us,vs) = partition (< b) xs
          b = a + 1 + n `div` 2
          m = length us
```

## 2.6 実測

紹介した 2 つの方法が本当に線形時間で終わるのかを実測してみる．その為に，次のようなメイン関数を定義した．

```
main = do
  [op] <- getArgs
  xs <- fmap ((fmap read) . words) $ getLine
  start <- getCurrentTime
  case op of
    "1" -> putStrLn $ show $ minfree1 xs
    "2" -> putStrLn $ show $ minfree2 xs
```

```
    "3" -> putStrLn $ show $ minfree3 xs
  end   <- getCurrentTime
  putStrLn $ show $ (diffUTCTime end start)
```

これについては詳しく言及しない．`minfree1,2,3` がそれぞれ，素朴な解法，配列を利用した解法，分割統治法である．標準入力で空白区切りの数列を与えればよい．例えば次のような感じである．

```
$ cat random100000.dat | ./minfree 3
487
0.860682s
```

結果は次のようになった．

| 要素数 | 素朴な解法 | 配列の利用 | 分割統治法 |
|---|---|---|---|
| $10^1$ | 0.000395s | 0.000432s | 0.000396s |
| $10^2$ | 0.002536s | 0.002272s | 0.002202s |
| $10^3$ | 0.031683s | 0.007383s | 0.015467s |
| $10^4$ | 2.007103s | 0.193031s | 0.159733s |
| $10^5$ | 98.935995s | 1.103851s | 0.860682s |

今回は素朴な解法の結果が悪くなるように，乱択自然数列を総数 $n$ に対し $0 \sim n$ までにしている．素朴な解法はあくまでも最悪が $O(n^2)$ なので，平均は総数に対する最大値に依存する．また，今回は計算量だけなんとなく見たかったので，特に平均をとってはいない．グラフにすると次のようになる．

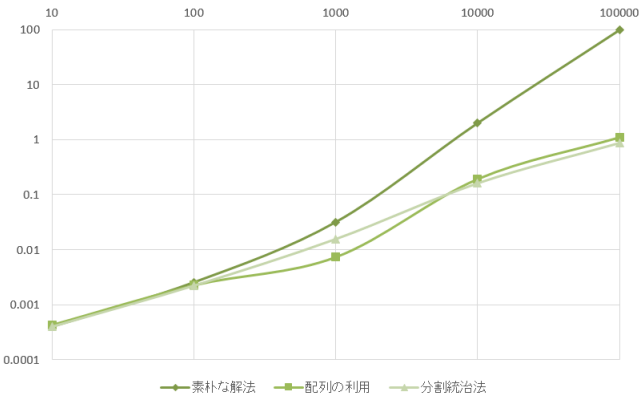



図 1: 実測結果

配列を利用した解法と分割統治法を利用した解法はどちらも (だいたい) 線形時間であることがわかります．素朴な方法は $O(n^2)$ とは言い難いですが，これは遅延評価のため最小自由数が何であるかに依存します (見つけ次第残りの計算を辞めてしまう．)．なので，こんなもんでしょう．

また，珠玉のアルゴリズムデザインでは配列を利用する解法より分割統治法を利用する解法の方が二割ほど速いと書いてあった．そこで，要素数が $10^5$ のときだけ，10 回の平均をとってみた結果次のようになった．

だいたい，三割ほど速かった (もちろんコンピュータのパフォーマンスに左右されるため目安程度の結果である)．

| | |
|---|---|
| 配列を利用 | 1.1652094 |
| 分割統治法 | 0.8680479 |

## 3 おわりに

想定以上に長くなってしまいました．すいません．しかも，だらだらと本の内容を書いただけで，対してオリジナル性もありません... 次はもう少しオリジナリティのあることをします．

結局のところ，今回の目的は Haskell もいろんな事できるんだよってコトを知っていただけたらなぁという感じです．Haskell を利用すれば，数式を利用した解法がそのまま適用できます．そこが手続きと型との大きな違いですね．このように，Haskell は他の言語には無い独特な味があります．ぜひ皆さんも Haskell を利用してみてください．

## 参考文献

[1] 山下伸夫 訳 Richard Bird. 関数プログラミング 珠玉のアルゴリズムデザイン, 2014.

# あぁ＾〜心がぴょんぴょんするんじゃぁ＾〜な人のための ティッピー合成マシンとこころぴょんぴょん識別器

ism67ch

## 概要

「あぁ＾〜心がぴょんぴょんすんじゃぁ＾〜」とは，例えば図 1 のような，ティッピーを頭に乗せたチノちゃんを見たとき，誰しもが自然と出てしまう言葉である（ちなみに筆者はシャロちゃん推しのため，図 2 でもぴょんぴょんする）．

図 1: ティッピーを頭に乗せたチノちゃん（可愛い！）

図 2: お仕事中のシャロちゃん（カワユス !!! ）

都会の喧騒にもまれる都会人，あらゆるアカハラに耐える大学生，空っ風に耐えるグンマーにおいて，この「こころがぴょんぴょんする」という感情は必要不可欠な癒しとなりつつある．そこで本稿では，ぴょんぴょんしない画像をぴょんぴょんする画像へと変換するシステムを紹介する．

また「こころがぴょんぴょんする」という感情に対し，これまで定量的な評価がなされてこなかった．これに対し，人間の持つ他の感情との関連性を用いた，Support Vector Machine（SVM）による線形判別の実例を紹介する．

## 1　はじめに

読者の皆様におかれては，一度は「ご注文はうさぎですか?」 [1] は既に見られていることと思われる（見てないよ〜って人は見てから読んでね）．見たことある人ならば，この「あぁ＾〜心がぴょんぴょんすんじゃぁ＾〜」という感情には心当たりがあるかと思う．筆者に限って言えば，毎週土曜日のぴょんぴょんタイムを原動力に，一週間頑張れている節がある．聞くところによれば，世の中には一週間働いてはラビットハウスに帰ってくるような人もいるようである．

さて，そもそもこの「こころがぴょんぴょんする」とは，どのような感情なのだろうか．ニコニコ大百科 [2] によれば，現代社会との関連性は「ストレスに対する休養地を発見した時に発する言葉」，哲学との関連性は「星そのものに生まれ変わって，神と最も近い場所で寄り添うのだという世界の真理を見つめる能力，第三の目を開くこと」と定義している．…なるほど，わからん．そこで筆者自身における「ぴょんぴょん」を「ごちうさ」を見ながら，整理してみることとした．

「ごちうさ」を見ていると，一話の中でも「ぴょんぴょん」の具合が違うように感じることに気が付いた．基本，見ている最中は「ぴょんぴょん」しているわけであるが，同じチノちゃんでもその度合いが違った．具体的には，図 3 と図 4 を見たときに感じるような違いである．

図 3: ティッピー無し　　図 4: ティッピー有り

どうだろう，ティッピー無しよりもティッピー有りの方が，「こころがぴょんぴょんする」という人が，多いのではないかと思う．裏付けとして，ティッピーの有無による「ぴょんぴょん」度合いの違いを図るため，被験者 10 人に対し，次のような超ローカルな調査を行った．

Q. 図 3 と図 4 を見て下さい．
　貴方はどちらの方がぴょんぴょんしますか？

A.

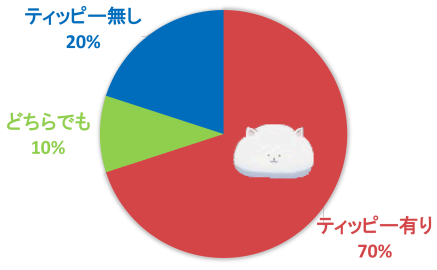



図 5: 調査結果

図 5 の調査結果より，筆者の周囲ではティッピーが有る方が「ぴょんぴょんする」という調査結果が得られた．ちなみに，無しと答えた人に話を聞くと，「無くてもティッピーが見える」と回答していた．

以上の調査より，ティッピーが頭に乗ってることにより「ぴょんぴょん」度合いが向上するという知見を得られた．筆者はこの事実を真摯に受け止め，多くの人の「ぴょんぴょん」度合いを向上させたいという思いから，ぴょんぴょんしない画像から，ぴょんぴょんする画像へと変換するシステムを開発しようと考えた．具体的には，図 3 から図 4 のように，ティッピーを画像中の人物の頭に合成することによってこれを実現した．このシステムに関しては，本稿 2 節で説明する．

また，このシステムを評価するにあたり，「ぴょんぴょん」度合いを定量的に評価する必要性を感じた．具体的には，人間の持つ他の感情との相関関係をデータセットとして用いることにより，疑似的に「ぴょんぴょん or Not ぴょんぴょん」識別器を SVM で設計する．このぴょんぴょん識別器に関しては，本稿 3 節で説明する．本稿 4 節で簡単な実験を示し， 5 節でまとめを示す．

本稿の目的は「多くの人のこころをぴょんぴょんさせること」である．長文になってしまったが，最後まで手に取って読んでいただき，読者の皆様に少しでも「ぴょんぴょん」を提供できれば筆者としては幸いである．

## 2　ティッピー合成マシン

ティッピー合成マシンのフローは図 6 のように至ってシンプルである．まず入力画像に対し，OpenCV の Haar-Like 特徴量を用いて顔認識を行う．そして次に，顔のディテクション情報を元に，顔の矩形上底における水平座標の中心にティッピーを合成するといった流れである．

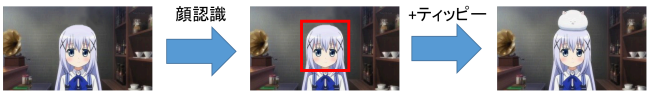



図 6: ティッピー合成マシンの流れ

Haar-Like 特徴量とは，顔の局所的な明暗差を捉えることによって記述される特徴であり，顔認識などで広く用いられている特徴である．実際にティッピーを合成した画像の一例を図 7， 8 で示す．

Fate/Zero 第 11 話でギル様が王の財宝から酒と酒器を取り出すシーンの画像である．どうであろうか，図 7 では張り詰めた臨場感溢れるシーンであり，カッコイイという感情が先立たないだろうか．一方，図 8 になると，どことなく空気が緩む．以上のように，適切に合成されるときちんと頭上にティッピーが乗る．

図 7: 合成前

図 8: 合成後

## 3　SVM によるぴょんぴょん識別器

「ぴょんぴょんする」という感情を評価するための識別器を設計した．被験者 20 人に「Plutchik の感情の輪」 [3] における基本感情 8 つの感情（嫌気，怒り，期待，幸せ，信頼，恐れ，驚き，悲しみ）を 5 段階で評価してもらい，アンケート時において，ぴょんぴょんしてるか否かも回答してもらった．それらを訓練データとして用い，線形識別器として SVM を用いて学習した．

線形識別器について簡単に説明すると，線形識別器とは与えられたデータに対し，それが正例か負例かを判定するものである．今回の実験では，正例がぴょんぴょんしている，負例がぴょんぴょんしていないとして捉えて頂きたい．SVM はこの線形識別器の 1 つであり，近年割と広く用いられている識別器である．

ぴょんぴょん識別器の流れを，図 9 に示す．

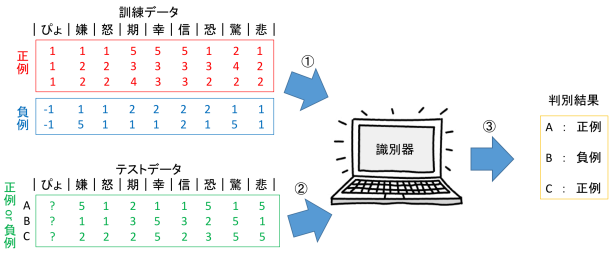



図 9: ぴょんぴょん識別器の流れ

図 9 の流れをステップ毎に説明する．

1. 訓練データとして，8 つの感情とその時にぴょんぴょんしていたか否かのデータを用いて，識別器を学習する．
2. 図 7，図 8 のような before と after の画像を見せた後，8 つの感情についてのアンケートを行い，それをテストデータとして識別器にかける．(今回 before の画像には極力ぴょんぴょんしない画像を選択した．)

3. 入力されたテストデータに対し，そのデータ（感情）が正例（ぴょんぴょんする）か負例（ぴょんぴょんしない）かを判別する．

# 4　実験

ティッピーが頭に乗ることにより，こころぴょんぴょんすることを確認するために取り行った実験について示す．4.1 項では実験条件を示す．4.2 項では実験結果と考察を示すとともに，いくつか上手くいかなかった例について考察する．

## 4.1　実験条件

before 画像：筆者が独断と偏見により選んだ，
ぴょんぴょんしない 15 枚の画像．

before 画像に対し，今回提案するティッピー合成マシンを用いてティッピーを合成し，合成後の画像を after 画像とする．被験者 10 人に対して before 画像を見せた後，after 画像を見せる．その後，8 つの感情（嫌気，怒り，期待，幸せ，信頼，恐れ，驚き，悲しみ）を 5 段階評価でつけてもらい，そのアンケート結果をテストデータとしてぴょんぴょん識別器にかけ，判別結果が正例（こころぴょんぴょんしている）となった割合を集計する．

## 4.2　実験結果

今回行った実験においては，正例（ぴょんぴょんする）の割合が 86 %をマークした．時間の関係上 before 画像の選定が甘かった部分もあるのが，今回選定した before 画像に関しては，概ねぴょんぴょんする画像に変換できたことがわかる．

今後の性能向上のため，今回の実験において被験者をぴょんぴょんに出来なかった例を考察していきたいと思う．

### 4.2.1　事案 1（合成ミス問題）

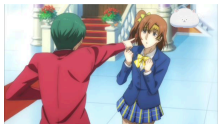


図 10: 事案 1

図 10 は被験者をぴょんぴょんさせられなかった画像の一枚である．この画像に対しては，顔認識が正確になされず，その結果ティッピーを頭の上に上手く合成出来なかったと考えられる．こういった事案は，顔が全て写っていなかったり，著しく顔全体の輝度が明るかったりすることに起因するものと思われる．ただし，そもそも元画像自体にぴょんぴょんする人もいるかもしれない．

### 4.2.2　事案 2（アンマッチ問題）

図 11: 事案 2

図 11 も被験者をぴょんぴょんさせられなかった画像の一枚である．この画像に対しては，ご覧の通りティッピーによるぴょんぴょん成分付加の限界を示している良い例である．いくらティッピーが可愛いと言えども，ゾンビ化したゆき姉に対してはあまりにもアンマッチである．ゾンビ萌えの人にとっては，もしかしたらぴょんぴょんするかもしれない．

# 5　まとめ

今回の実験において，合成ミス問題とアンマッチ問題が残った．合成ミス問題に関しては，原因が割と明白なので時間があるときにでも対処したい．アンマッチ問題に関しては，適材適所な部分が否めないため，なんとも難しい．しかしながら，そういった画像に対しても，ティッピー以外の何かを合成することにより，ぴょんぴょんさせることができるかもしれないという可能性は残しておきたい．

また，今回は締め切りが切迫した中だったため，そもそも before 画像においてぴょんぴょんするのではという可能性に対しては，筆者が極端にぴょんぴょんしない画像を選択してしまったので，性能改善を測る際には，その部分気を付けて行きたいと思う．今後は，ぴょんぴょんを多段階でも評価出来るようにブースティング等々でも試したいと思う．

最後に，本稿は原作を否定するものでなく，むしろ筆者が「ごちうさ」が好きな感情に起因していることを強調する．また，本誌を手に取って読んで下さった読者の皆様には，誠に感謝したいとともに，「ごちうさ」を愛し，日々「こころぴょんぴょん」しながら生活して欲しいと願う．

**「あぁ＾〜心がぴょんぴょんすんじゃぁ＾〜」**

# 参考文献

[1] ご注文はうさぎですか？:http://www.gochiusa.com

[2] ニコニコ大百科:http://dic.nicovideo.jp/a/あぁ%5E〜心がぴょんぴょんするんじゃぁ%5E〜

[3] Plutchik の感情の輪:https://ja.wikipedia.org/wiki/%E6%84%9F%E6%83%85%E3%81%AE%E4%B8%80%E8%A6%A7#cite_note-1

[4] 出典：ごちうさ画像，Fate/Zero 画像，おそ松さん画像＠本編，ゆき姉＠ロー＋さん

# うますぎる闇鍋の作り方

大宮・テイラー

## 風味を変えた食材

今回，おいしい闇鍋の作り方ということで，入れたら味と風味が独特になってしまった食材を紹介したいと思う．私が今まで闇鍋という名の迷路で遭遇した一癖も二癖もある食材である．どんな鍋でも最初から闇に染まっているわけではない．闇の鍋に落ちる原因となる物が必ず存在する．私の知る「闇鍋落ち」する要因を以下に紹介する．

### とろけるチーズ

危険度：★★
味自体はチーズ風味が増す．鍋を突いていたはずなのにいつの間にかチーズフォンデュをしている気分になれる．食材に絡みつくチーズと煙はチーズフォンデュの故郷であるアルプス山脈を彷彿とさせる ･･･ のだが，チーズの香りが残り，数日間はアルプスと闇鍋の思い出が部屋全体に広がり続ける．部屋に匂いが残る点は少し危険．

### チョコレート

危険度：★★★★★★★★
2 月といえばチョコレート．そんな安易な考えから第一回の「始まりの闇鍋」で板チョコを入れたところ鍋の素が一瞬にして浸食された．食材の味は（チョコ : 本来の食材 : 謎な味＝ 3:5:2）でどこかの民族料理でありそうな味と当時は思った．立ち上るあまいかおりは鍋の食材だけでなく部屋一面に充満し，参加者の思考を低下させた．そして参加者全員が無心状態で映画「アバター」を見る異様な光景になった．そして，この甘ったるい独特な匂いは三日三晩闇鍋会場と玄関を占領すると言う大きな爪痕と現実を残した．

### 粉状のスポーツサプリメント

危険度：★★★★★★★★★★
多くの栄養素を含んでいるバランス栄養食である闇鍋と相性はいいかと思われたが違っていた．板チョコほどは甘ったるい匂いはしなかったが，修正が効かないほど鍋の味が激変した．チョコのようなストレートな味ではなく,「ぬるく甘いなんともいえない未知のあまりオススメできない味」だった．味の修正のため味噌などの調味料を入れる，希釈するなど行ったが効果はなくさらに意味不明な味になった．

### パイナップル

危険度：★★
パイナップルはブロメライン（通称パパイン）と呼ばれる酵素によって肉を柔らかくすると予想していたが，大した効果はなく闇鍋の酸味に変わった．レモンほど強烈な酸味ではなく軽め，気持ち程度の酸味だった．パイナップル，みかん，グレープフルーツなどのフルーティな食材を加えるとホットフルーツが食べられる．酢豚にパイナップルを容認している人は案外いけるかも．

### レモン

危険度：★★★★★★
レモンは唐揚げにかける調味料として用いられることが多いが闇鍋に入れると徐々に鍋の酸味が強くなっていった．味は酸味と苦味を兼ね備えた暖かいレモンといった感じで苦味の割合の方が多い．闇鍋にスライスしたレモンを入れて以降，私は闇鍋にはレモンを入れない派になった．

## まとめ

最後に，★ 4 以下は鍋に入れてもほとんど問題ない．それ以上の物を入れると，世界が激変してしまう．それ以外の食材であれば比較的うまい闇鍋が食べられるかもしれない．もし，忠告を無視し挑戦しようと言う勇者が現れた時，どんな状態でも最後まであきらめずルールを守って楽しく闇鍋を行うことができることを切に願う．ぜひ鍋料理を闇の色で染め上げてくれ．

**「ファイトだよ！」**

# “動く”半導体、MEMSデバイスの話

れぽ @gokinaka

## 1　はじめに

どこのお宅にも 1 台はあるであろうインクジェットプリンター．MEMS 入ってます．
群馬県民は持っていないと正直しんどい自動車のエアバッグ．MEMS 入ってます．
某イカのゲームとかやるハード用の棒形コントローラ．MEMS 入ってます．
多分初めて聞く単語だと思いますが，結構生活で便利に使われている半導体デバイスのいちジャンルです．今回は半導体業界では，あと 30 年で一番儲かると噂の MEMS デバイスの話をしようと思います．

## 2　MEMS の概要

MEMS は Micro Electro Mechanical System の略です．日本語なら微小電気機械システムとか書きますが，ほとんど MEMS と呼ばれています．微細集積回路を作る技術使って他にも色々作れない?という発想から始まった技術です．大体 1980 年位にアメリカの Kurt Petersen などが言い出しました [4]．MEMS 関連の研究が進んだおかげで微細回路と複雑な機械構造が混ざったマイクロオーダーの形状が大量に安定して作れるようになりました．

## 3　製品 MEMS デバイスの種類

### 3.1　加速度センサ

加速度センサは MEMS デバイスの中でも一番作られています．すごく小さいのに (ST のデバイスだと 2 × 2 × 1mm など)，振動や重力，衝撃などを検出できます．スマホを横にすると画面も横になるのはこれのおかげ．プロジェクタの台形補正機能も，車が急停止するとエアバッグが開くのもこれのおかげと，色々と入ってます．色々なメーカーが作っていますが，工具でお馴染み BOSCH とかが有名です．

図 1: 某ゲーム内の加速度センサ (約 5mm 角)

### 3.2　インクジェットプリンタヘッド

インクジェット方式は各色インク滴を紙の上に着地させて印刷します．これを行うには，pL オーダーのインクを 25,000 発毎秒で正確に飛ばす必要があります．この飛ばす機構は MEMS で作っています．インクを押し出すアクチュエータ (可動部分) を小さく安く作れるメリットが活きるのでほぼ全てのインクジェットプリンタのインクノズルに採用されています．家庭用プリンタが普及したブレークスルーの一つです．[2]

### 3.3　DMD(Digital micromirror Device)

プレゼン等に使うプロジェクター，10 年前と比べてすごく小さくなりました．最近はタブレットにも内蔵されるなど，更に小型化が進んでいます．これらには DMD と呼ばれる 10um 四方の鏡が平面上にずらっと並んだ部品が入ってます．(図 2) この鏡は± 10 度程度で角度を変えることができます．これに光源を当て，鏡の方向を変えて制御することで，映像を描画します．[3]

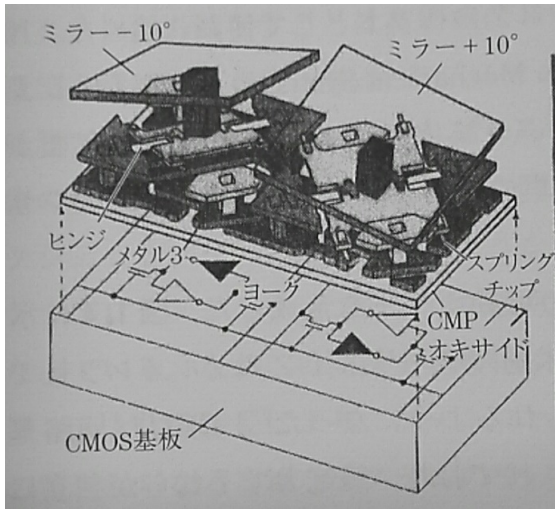



図 2: DMD チップ模式図)

## 4　研究開発段階のデバイス

いままで挙げたデバイスは既に普及しているデバイスです．しかし MEMS が儲かると言われる所以は，考えられる

応用範囲がすごく広いから．アイディア一つで儲かりそうなデバイスがいっぱいあります．個人的に面白くて儲かりそうなものを 2 つ紹介します．

### 4.1 臓器チップ (Organs on a chip)

MEMS は半導体プロセスを端に始まったものなので，材料はシリコン (Si) が主です．しかし最近は高機能金属や高分子材料など，様々な材料が使えるようになりました．よってバイオ分野でも活発に研究が行われています．その中の一つが臓器チップです．臓器の正確なミニチュアを作って実験する試みです．例えば肺は周期的に動く臓器なので，単純に構造を模するだけでは不十分でした．MEMS で作った肺は血液，空気や肺の動きを再現できるので，[ウイルス感染した肺内の血液変化」などを観察出来ます [1] ．動物愛護の点からも注目されています．詳しい事は TED で臓器チップと検索すると開発者の話が聞けます．

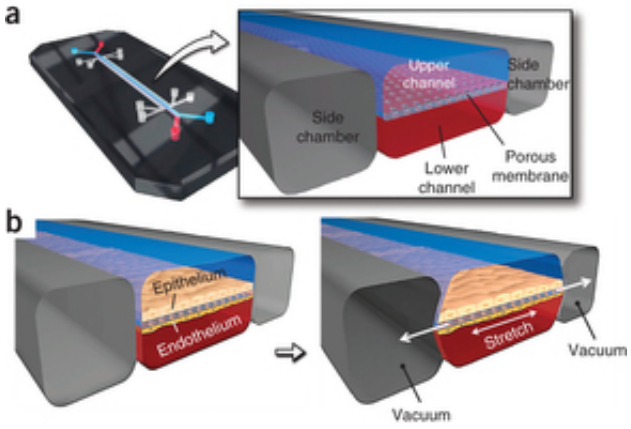



図 3: 肺チップ (Lung on a chip))

### 4.2 エネルギーハーベスト

MEMS デバイスの特徴として低消費電力が挙げられます．また，微小構造を形成し振動や電磁気から発電する研究が進んでいます．これにより，自家発電して単独で動作しつづけるデバイスが作れます．現在も住宅内に設置するセンサが市販されています．また，配線の必要がないので，回転体に容易に付けることが出来ます．タイヤの空気圧監視や，プロペラの変化等を持続的に測定出来るシステムが実用化されつつあります．他にも体内に埋め込むセンサなど医用分野などにも応用がきく等，今後 10 年で大きく成長しそうです．

## 5 メーカーと開発の話

先ほど，MEMS はアイディア勝負と申しましたが，アイディアを実現するためにはお金がかかります．半導体プロセスは使用する装置が高価で維持費もかかるので，現状開発しているのは研究機関や大手部品メーカーがほとんどです．しかし，近年ベンチャー企業などが様々なデバイスの試作を始めています．その背景にあるのは，ファウンドリです．ここ 10 年で「半導体業界は集積回路を設計するファブレスメーカー」と，「受注した設計から製造量産を専門に行うファウンドリ」というふうに分業化が進みました．現在，スマホの CPU などはほとんどこの形態で生産されています．最近，MEMS も作ってくれるファウンドリが増えてきました．日本では MEMS デバイス専門ファウンドリのメムス・コアをはじめ，受託開発を得意とするメーカーが増えました．よってベンチャー企業も設計のみに集中できる環境が広まり始めています．

## 6 まとめ

技術系のお話が多い中，空気を読まない記事を書きました．情報系からするとほぼ確実に関わることのない低レイヤー層の話です．でも便利そう，儲かりそうだなと思ってくれると嬉しいです．

## 参考文献

[1] D.Huh and et al. Microfabrication of human organs-on-chips. In *Nature Protocols 8*, pages 2135–2157, 2013.

[2] デザインウェーブマガジン編集部. *MEMS* **開発 活用スタートアップ**, 2004.

[3] 江刺正喜. **はじめての** *MEMS*. 森北出版, 2011.

[4] 前田龍太郎 and et al. *MEMS/NEMS* **の最先端技術と応用展開**. フロンティア出版, 2006.

# ROS(Robot Operationg System) を勉強し始めたよ

トム

## はじめに

一年前くらいに ET ロボコンからロボットとかに興味を持って、研究室も制御やってるし、メカトロ関連の勉強したいなーと思い始めてはや半年。今回は ROS というロボット用ミドルウェアを勉強し始めたのでそれをまとめました。

## 1 Robot Operating System

”OS”というと Windows・Mac を思い浮かべると思いますが、この ROS はその”OS”ではなく、ロボットシステムを構築することに特化したミドルウェアです。なので OS というよりも、名前の通り「ロボット開発において異なる開発環境下でも使える共通のライブラリとアプリケーションをまとめたツール」というイメージです。しかもオープンソースなのでこれから紹介する ROS のアプリケーションはすべて無料で使えます。対応している環境は基本的に Ubuntu です。Ubuntu のバージョンは 14.04、ROS は「indigo」を使用しました。

## 2 ロボット用通信方式としての ROS

通信が必要なロボットシステムを開発するときにめんどうなのはハードウェアごとの API や言語などの開発環境の違いですね。ROS はそれらの異なる環境やロボットに共通のライブラリを提供してくれます。IP アドレスでお互いを識別し、ロボットやセンサーを一つの Node として扱い、メッセージ (msgs) を送受信します。Publisher で情報を発信し、Subscriber で受け取る、この仕組のため、送受信のための型さえ守れば同じソフトウェアでいくつものロボットを動かせるのです。

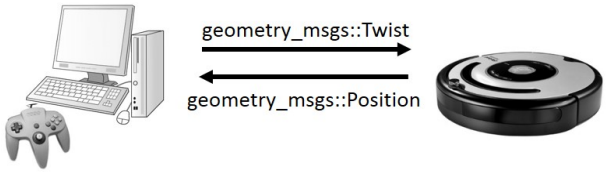



図 1: ROS のメッセージの型。Twist で目標速度を送り、Position でロボットの自己位置を返している。

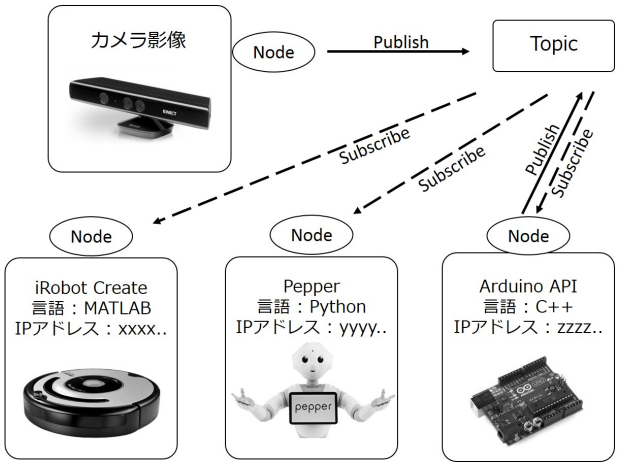



図 2: ROS の通信方式。最近 Pepper にも対応した。

## 3 ROS を使った arduino との通信

さっそく ROS を使って Ubuntu の Terminal と arduino で通信をして L チカをしてみます。図 3 のようなプログラムを書きました。

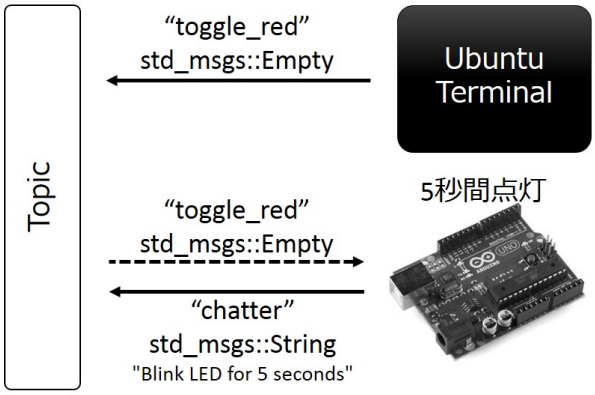



図 3: Terminal から空メッセージの Empty を Publish し、arduino が Subscribe すると 13 番ピン LED が 5 秒間点灯し、”Blink LED for 5 seconds”と Publish する

Ubuntu で arduino IDE を起動し、下記のコードを arduino に書き込みます。言語は C++です。

```
#include <ros.h>
#include <std_msgs/Empty.h>
#include <std_msgs/String.h>
//ROSのライブラリ

ros::NodeHandle  nh;
//arduino用のNodeを作る
std_msgs::String str_msg;
//ROSのStringクラスのインスタンス定義
ros::Publisher chatter("chatter", &
    str_msg);
//Publisherクラスのインスタンス定義
char messageLED[] = "Blink LED for 5
    seconds";

void messageCb( const std_msgs::Empty&
    toggle_msg){
  str_msg.data = messageLED;
  chatter.publish( &str_msg );
//str_msgをPublish(送信)する
  digitalWrite(13, HIGH);
  delay(5000);//5秒間LED点灯
  digitalWrite(13, LOW);
}

ros::Subscriber<std_msgs::Empty> sub("
    toggle_led", &messageCb );

void setup()
{
  pinMode(13, OUTPUT);
  nh.initNode();//ROSのNodeの初期化
  nh.subscribe(sub);
  nh.advertise(chatter);
}
void loop()
{
  nh.spinOnce();//Subscribe(受信)待ち
  delay(1);
}
```

それではこのプログラムを動かしていきます。Ubuntu の terminal を 4 つ立ち上げ、1 つめに

```
roscore
```

2 つめに

```
rosrun rosserial_python
    serial_node.py /dev/ttyACM0
```

3 つめに

```
rostopic echo chatter
```

と打ち込みます。roscore は Node 間で通信をするために ROS を立ち上げるコマンドで、2 つめのコマンドは arduino とシリアル通信を始めるときのコマンド (私の arduino は /dev/ttyACM0 にポートがあるよう) です。3 つめのコマンドは図 3 の topic に”chatter” という名前で Publish(送信) されたものを表示するコマンドです。

そして最後の 4 つめの terminal を使って LED を光らせます。arduino との通信は 2 つめのコマンドで始まっているので、4 つめの terminal に以下のコマンドを打ち込みましょう。

```
rostopic pub toggle_led std_msgs/
    Empty --once
```

これは「topic に”toggle_led” という std_msgs/Empty 型の空メッセージを一度 Publish(送信) する」というコマンドです。先ほどのソースコードより、arduino は nh.spinOnce() で Subscribe(受信) 待ちの状態となっていて、USB のシリアル通信でこの空メッセージを arduino が Subscribe(受信) するとコールバック関数 messageCb() が呼ばれる仕組みとなっています。

コールバック関数 messageCb() では、chatter.publish() が呼ばれ”Blink LED for 5 seconds” と topic に Publish(送信) します。その後 13 番ピンの LED を点灯させて 5 秒後に消灯し、関数から抜けて再び nh.spinOnce() の受信待ち状態に戻ります。

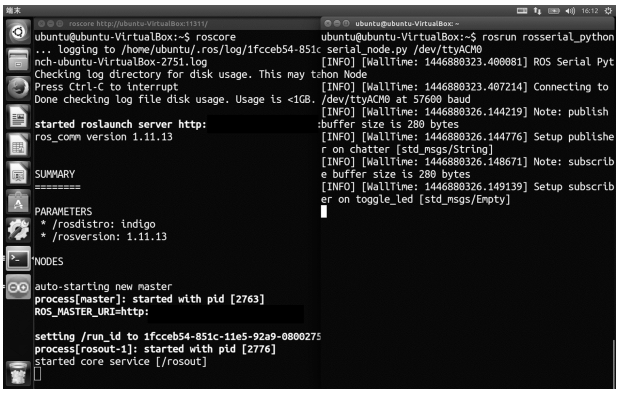



図 4: ROS の立ち上げ

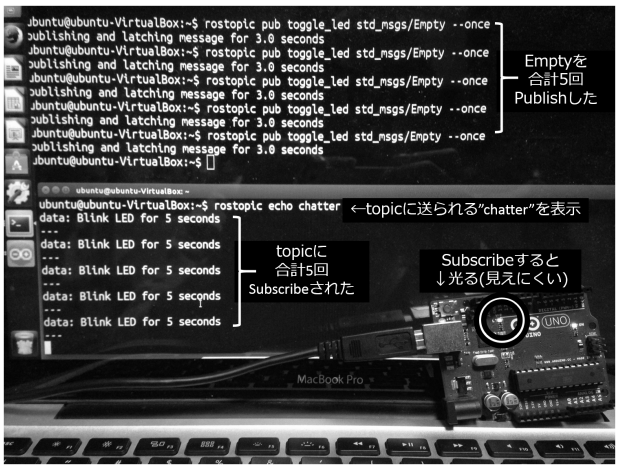



図 5: terminal で LED 点灯司令の Publish(上)

# 4　ロボットシステム用 GUI アプリケーションとしての ROS

前章では実際に ROS で arduino と通信をして L チカをしました。ただ、通信のシステム解説なんてものはやっぱり地味です。なのでこの章からは ROS で使うことのできる便利な GUI アプリケーションの一部を紹介していきます。

## 4.1　物理シミュレータ Gazebo

Gazebo はフリーの物理シミュレータです。多くのロボットのモデルが Gazebo で使用でき、ROS と同じ (?) ところで製作されているため ROS に対応しているという特徴があります。

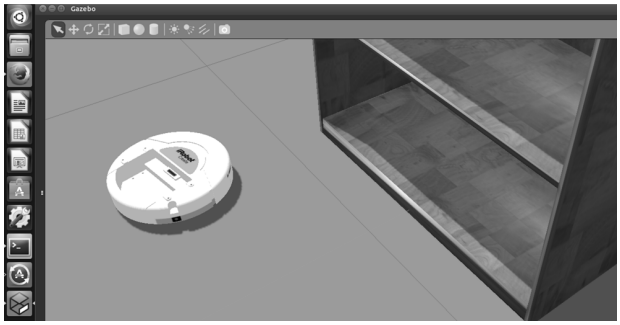



図 6: Gazebo 内の本棚と iRobotCreate(ルンバ的なロボ)

## 4.2　3D 可視化ツール Rviz

Rviz はロボットから Publish(送信) されてきた情報からの身の回りの状態を擬似的に可視化します。多くの自由度を持つロボットは特に可視化が重要になります。図 7 では Turtlebot という iRobotCreate に kinect を乗っけた (?) ロボットから PCL を使い奥行き情報を取得して前方の障害物を可視化しています。また、カメラの影像を直接表示したり事前に作ったマップと重ねて表示させることもできます。

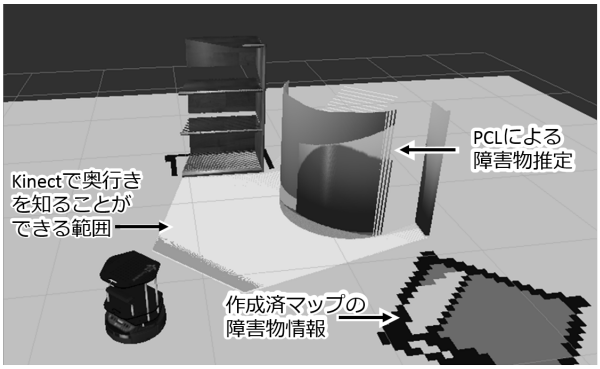



図 7: kinect の奥行き情報から前方の障害物を視覚化

## 4.3　Robot + Qt

Qt はマルチプラットフォームに対応した GUI 作成に便利なフレームワークです。ただ、私は Qt 単体では使ったことはありません。ROS にもこの Qt が対応していて比較的簡単に図 8 のようなボタンやスライダーの GUI をおいたり、グラフ化を行うことができます。

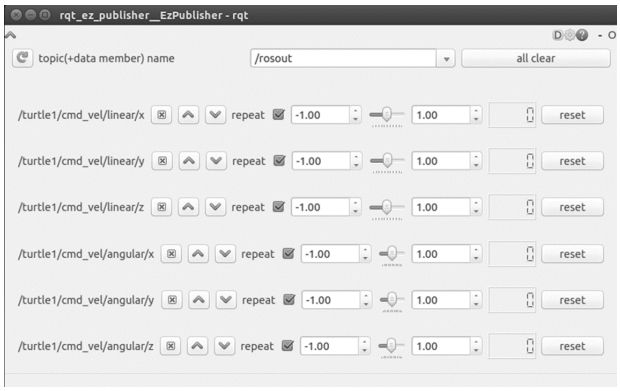



図 8: rqt で作成されたコントローラ

# 5　ROS と wii リモコンで シミュレータ内のルンバを動かす

今回は Gazebo を使ってシミュレータ内のルンバを wii リモコンで操作してみます。プログラムの概要は図 9 のように、wii リモコン傾斜角度から PC で目標移動速度を算出し、それをルンバに送信します。

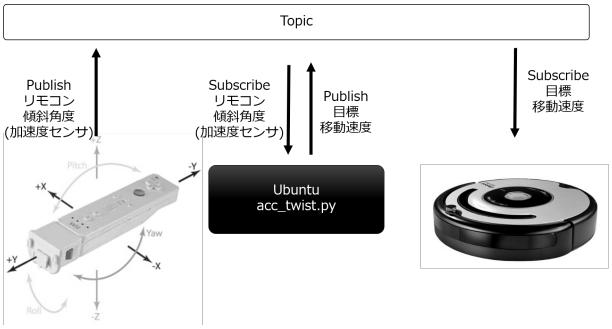



図 9: wii リモコンをコントローラに

まず、wii リモコンを ROS でつなげてどんな情報が Publish されているか確認します。wii リモコンのドライバを github からダウンロードしてインストールします。その後下記のコマンドを terminal に打ち込み、3 秒以内に wii リモコンの 1 と 2 ボタンを同時に長押しします。

```
rosrun wiimote wiimote_node.py
```

同期が完了したら別の terminal を起動し、下記のコマンドを打ち込みます。

```
rostopic echo /wiimote/state/
    linear_acceleration_zeroed
```

rostopic echo は topic に Publish(送信) されている情報を見るコマンドで、/wiimote/state/linear_acceleration_zeroed は wii リモコンから Publish されている各軸の加速度を表示するコマンドです。

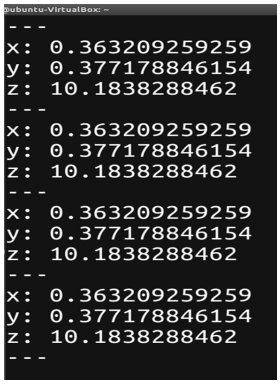



図 10: wii リモコンから得られた加速度

図 10 をみると三軸加速度ベクトルの絶対値は重力加速度の 9.8 くらいになっています。今回はどうやら geometry_msgs の型で Publish(送信) されている加速度を Subscribe(受信) してその値を適当な速度に変換し geometry_msgs 型で Publish(送信) するプログラムを作成すれば良いようです。

```
1  import rospy
2  from wiimote.msg import State
3  from geometry_msgs.msg import Twist
4  #ROSと各型のライブラリをインポート
5
6  class AccTwist(object):
7      def __init__(self):
8          self._acc_sub=rospy.Subscriber
               ('/wiimote/state',State,self
               .acc_callback, queue_size=1)
9      #State型で'/wiimote/state'という名前
           のSubscriber(受信者)
10         self._twist_pub=rospy.Publisher
               ('/mobile_base/commands/
               velocity',Twist,queue_size
               =1)
11     #Twist(速度)型で'/mobile_base/
           commands/velocity'という名前の
           Publisher(発信者)
12
13     def acc_callback(self,acc_msgs):
14             twist=Twist()
15             twist.linear.x=acc_msgs.
16             linear_acceleration_zeroed.x
                   *(-0.02)
17             twist.angular.z=acc_msgs.
18             linear_acceleration_zeroed.y
                   *(0.08)
19 #Twist(速度)型に加速度を定数倍して代入
20             self._twist_pub.publish(
                   twist)
21             #twistをPublish(送信)する
22
23 if __name__=='__main__':
24     rospy.init_node('acc_twist')
25     acc_twist=AccTwist()
26     rospy.spin()
```

今回のプログラムは python で書きました。プログラムは上記のとおりです。ROS で誰がどんな通信をしているか可視化する rqt 図を図 11 に示しました。wii リモコンから Gazebo のルンバにどんな過程を経てデータが送信されているかがわかると思います。そして実際に wii リモコンでルンバを動かしている様子が図 12 です。静止画だとわかりにくいですね … 。とにかく ROS を使うと案外簡単にハードウェアや開発環境の差を感じにくくシステムを構築しやすいということがわかっていただけたらと思います。

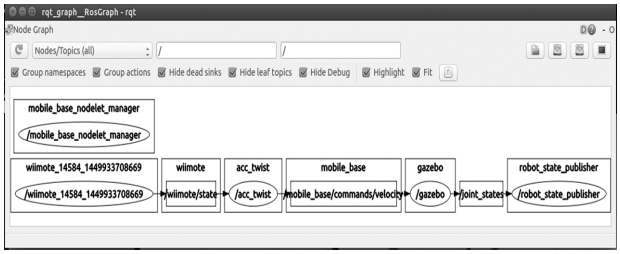



図 11: qrt 図


図 12: 実際に動かしている様子

## さいごに

今回は私自身学び始めのことをまとめたため、まあまあ勘違いやごまかしが入っていると思いますがそこは薄目で見ていただけたらと思いますし、私の専門がプログラミングではないためコードもあまり綺麗さとか考えないでいただけると嬉しいです (汗)。私が考える ROS を使っていて便利だなーと思うところは、やっぱりいろいろなツールのインターフェイスを型合わせだけで使えるところです。arduino も wii リモコンも簡単に ROS で操作できるのはとても心強いし、こういったライブラリが豊富なのも ROS がオープンソースなゆえだと思います。研究で使うまで行かなくても趣味として物理シミュレータで遊ぶのも楽しいですし、既存のロボットモデル以外に自分でロボットモデルを作成することができるのでそこそこ性能がある PC があれば自作ロボットをシミュレータ内で走らせることもできます。ROS は無料ですし便利なツールであることは間違いないのでこれからもっともっとこの ROS についてさらに学んでいきたいと思っています。

# もしAviUtl動画製作者がMMDerになったら

風土

## 概要

2015 年 8 月 14 日，以前筆者も参加したことのあるGunma.web(web 勉強会 in 群馬　WEB サイト作成に関わる人が集まる勉強・研究・交流の地域コミュニティ:公式サイトより引用) のメンバーの H 様から「Gunma.web のオープニングで流す動画」の制作依頼を受けた．普段は AviUtl による動画制作を行っている筆者だが,「今まであまり触れてこなかったジャンルの動画を作るチャンスだ」と考え，MMD による動画制作を行った．以下は，原案決定から実際に MMD モデルを使用して動画を完成させるまでの記録である．

## 1　動画のコンセプト

動画のコンセプトとして依頼されたのは「apple 社 CM のようなカッコイイ動画」である．もちろんこれだけでは動画は作れず,「細かい内容は任せる」とも言われていたので，独自の解釈で「カッコイイ」を突き詰めることにした．オープニング動画を作るにあたり，まず以下の 2 つを基礎とした．

1. MMD を用いること．
2. ストーリー性があること．

次章以降でその内容について述べていく．

## 2　MMDerとは

MMD によって作品を作っている人を指す．MMD は様々な要素で構成されている．動画作品を作るだけでは無く，素材を作る人達もまた MMDer である．

### 2.1　MMD

MMD とは MikuMikuDance の略であり,「樋口 M」こと樋口優氏が個人で開発し，自身のウェブサイト「VPVP（Vocaloid Promotion Video Project）」で無償公開しているフリーの 3DCG ムービー製作ツールである．(ニコニコ大百科【MikuMikuDance】の項より一部引用)

### 2.2　モデル

MMD には，ユーザーによって作られた数多くのモデルが配布され，その多くが無償である．アニメやゲームのキャラクターを模したもの，ユーザーオリジナルのものなど種類は様々である．人型だけでなく，動物や小物，建物や背景のモデルも存在する．モデルにはそれぞれ物理演算を適用することができ，髪の毛や服の動きなども重力に沿った動きを再現することが可能である．

### 2.3　モーション

モデルには「ボーン」と呼ばれるものが存在し，それを動かすことによってアニメーションを作ることができる．パラパラマンガのように各フレームごとにボーン位置を決定する必要はなく，0 秒のフレームに A というポーズ，10 秒のフレームに B というポーズを設定すれば，10 秒かけて A から B のポーズに変遷するアニメーションを作ることができる．これを更に短い間隔で繰り返し続けていくことになる． また「モーショントレス」を呼ばれる技法もある．人間のダンサーが踊っている動画からモーションを写す（トレース）ことである．通常，ダンスの動画は正面からの 1 方向のみから撮影されているため，自然なモーションを作るのは非常に今期の要る作業となる．

### 2.4　エフェクト（MME）

ぼかしやブラー，発光などのエフェクトを追加できる．エフェクト作成専門の MMDer もいる．

## 3　ストーリーとモーション解説

大まかなストーリは「とあるカフェにて問題に直面した主人公が，Gunma.web の存在を知って会場に駆けつける」である．以下にストーリー一覧を示す．また，使用したモデルは「慟哭のナイトメア」シリーズのものをお借りした．

### 3.1　悩む主人公

カフェでノート PC を操作する主人公，だがため息をつき，浮かない表情をしている． 主人公モデルの右手首ボーンとマウスの親ボーンをリンクさせ，右腕と一緒に動くようにしている．肩の上下運動によって「ため息」を表現した．

### 3.2　PC を操作するカフェ店員

奥のカウンターから出てきたカフェ店員．主人公に話しかけ，脇から主人公の PC を操作する．ディスプレイに表示されていたのは，Gunma.web の公式ページであった． 店員の「歩くモーション」を作るのに苦心．人間は足先ではなく膝・ももを動力として歩くため，足先の動きに膝を連動させる MMD との動き方の差異が顕著に現れてしまう．今回制作した動画ではその検証を生かせていないため，次回以降の MMD 動画で活かせるようにしたい． PC モデルのディスプレイは差し替え可能だったので，筆者の PC 画面のスクリーンショットをそのまま画面にした．

### 3.3　カフェから駆け出し会場へ走る主人公

ノート PC を畳み，カフェをあとにする主人公． ノート PC の開閉は PC のモデルのモーフで行った．PC の動きに合わせて主人公の腕をシンクロさせるモーションを打つことで，両手で PC を閉じる様子を表現した． ここの主人公の「走るモーション」の作成が一番苦労した．いわゆる「女の子走り」を目指したのであるが，筆者が男性であることなどから作業は難航した．Twitter のフォロワーの皆様からの意見を募り，最終的に納得できるものを作ることができた．なお，この女の子走りモーションはニコニコ動画にて配布動画を公開している．(http://www.nicovideo.jp/watch/sm27268982【MMD】女の子走りモーション配布)

### 3.4　既に発表が始まっている会場

男性 1 名がホワイトボードの前で発表中． 会場のモデルとして会議室を使用．カメラを右から左に移動させて会場を広く見せた．(実際の会議室モデルは非常に狭いものとなっている．) ホワイトボードに描かれている文字については，Gunma.web 側と何度も話し合いが行われた．最終的に「内容の詳細はわからないが，何かを説明している意図がはっきりとわかる『UML 図』」を用いることにした．

### 3.5　会場に到着した主人公

会場のドアから顔を出し，キョロキョロする主人公． 首を伸ばして顔だけ見えるようにしたかったのだが，ボーンの関係上うまくいかなかったので，扉の向こう側で主人公は完全に宙に浮いている（地面と設置していない）．見えないところは何をしていても自由なのである．

### 3.6　ロゴとキャッチコピー

ホワイトボードが反転すると Gunma.web のロゴ.「新しいナルホドを Gunma から」のキャッチコピー.

## 4　動画を作った感想と雑記・まとめ

1 年ぶりに動画の製作依頼を受けたのは良かったものの，「果たしてこの動画はクライアントの意図するものに沿っているのか」という自問自答の無限ループであった．今回の記事はあくまで MMD に関することに絞ったが，動画制作に当てた 31 日間のうち，ストーリーを完成させるのに 25 日を要した．実際に MMD で動画を作ったのは 5 日程度である．もっと MMD にかける時間を増やすべきだったと反省すると同時に，今まで一切触れてこなかった「ストーリーを作る」という作業ができたことに意味があると感じたい．

# 寝坊しない目覚まし時計

UTSUTSUKU

## 概要

寒い朝、お布団の囁きが聞こえる.
「この楽園から逃さないよ」
しかし, いつまでも楽園にいるわけにはいかない. 現実の世界に戻らなければならない.
こうして, 現実世界に戻るための目覚まし時計を作ることにしたのだ.

## 1　はじめに

季節は冬. 寒くて朝起きられなくなり, 休日の午前中を久しく経験していない読者の方も多いだろう. 朝起きるために目覚まし時計を複数セットしても起きられない場合や, 目が覚めても布団から出られずに二度寝してしまう場合が多いと思われる. 筆者も図 1 に示すようなポテンシャルの井戸を超えるのに苦労している.

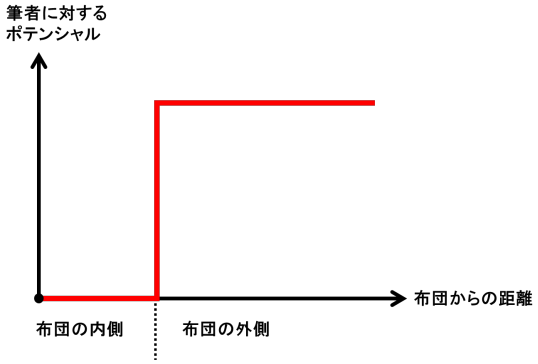



図 1: ポテンシャル概要図

以下に朝起きられない場合の行動パターンをいくつか挙げておく.

1. 布団に体の一部を入れた状態で, すべての目覚まし時計を止めて二度寝してしまう.
2. 離れた場所に置いた目覚まし時計を無意識のうちに止め, 布団に戻って二度寝する.
3. そもそも目覚まし時計の音に気づかない.

目覚まし時計の音に気がつかないのはどうしようもないが, 上述の 1 のパターンと 2 のパターンについては, 以下の条件を満たすことで克服できると考えられる.

- 布団から出る.
- 立ち上がる.
- 明かりを付ける.
- 部屋から出る.

以上の条件を満たしつつ経済性・汎用性・拡張性に富み, 容易に実現できる目覚まし時計の機能拡張方法について考察したところ, 目覚まし時計にカギを掛けるコトを思いついた.

## 2　準備

目覚まし時計に鍵をかけるために必要な物を以下に列挙する. 殆どのものは 100 円ショップで入手できるだろう. それぞれの写真を載せたかったのだが, 失念してしまった. そのため, 次のセクション中の写真を参考にしてほしい.

- 任意の目覚まし時計 (音に気づくもの)
- プラスチックのカゴ (自分の目覚まし時計に合った大きさのもの)×2
- 結束バンド
- 南京錠

## 3　作り方

1. 2 つのカゴを結束バンドでつなぎ, 蝶番の様にする（図 2,3). この時, スムーズに開閉できカゴとカゴの間が開き過ぎないように注意する.

図 2: 接合部の外側

図 3: 接合部の内側

2. 蝶番の反対側の面に結束バンドの環を取り付け, 南京錠を掛けられるようにする（図 4,5). この際にも, カゴとカゴの間が開き過ぎないように注意する.

図 4: 結束バンドの環

図 5: 南京錠

3. これで完成.

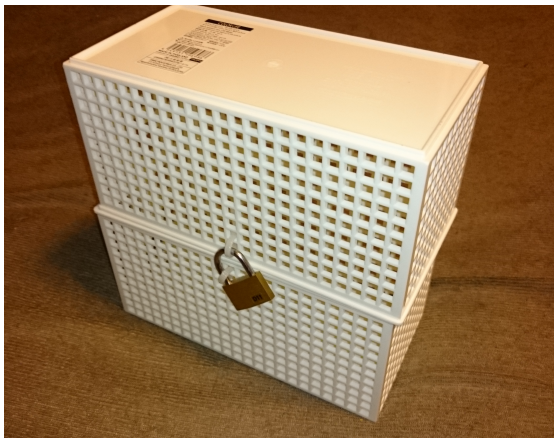


図 6: 完成図

# 4　使い方

## 4.1　基本

多くの読者の方はすでに感づいていると思われるが, 使い方を説明しておく. はじめに, 目覚まし時計をカゴの中にセットする（図 7). 次に, 南京錠を掛けてカゴをロックする (図 8).

図 7: セットした時計

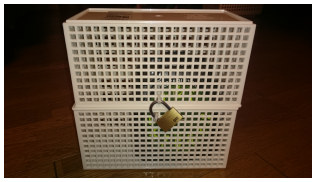


図 8: ロックしたカゴ

その後, カギをどこかに置くのだが, 置き場所をよく考える必要がある. 一度行けば二度寝しない場所に置かなければならない. 例えば, 壁の高めの場所に引っ掛けておけば, 布団から出る必要があり二度寝の防止になる. 筆者の場合は, 部屋の外の壁にカギを引っ掛けている.

## 4.2　機能拡張

南京錠 1 つだけでは, 寝ぼけた状態でも薄明かりの中でカギを開けてしまう方もいるだろう. そういう方はカゴに掛ける錠の数を増やしたり, 錠の種類を変えたりすることで, 自分に合った目覚まし時計の解除条件とすることができる. 例えば, 南京錠を複数取り付けることでカゴを開ける難易度を上げる事ができ, 南京錠ではなくナンバーキーにすることで明かりをつける必要性を付与することもできる. また, スマートフォンなどをカゴの中に入れておくことで布団の中でそれらを弄る時間を減らすことができる.

# 5　使った感想

ここでは寝坊しない目覚まし時計を実際に使用した筆者の感想や結果を述べる. 率直にいうと, 南京錠一つだけで使用する場合は効果が長続きしなかった. 使用する度に慣れて「無意識のうちにカギを取りに行く程度の能力」が強化されてしまい, 目覚まし時計を解除した後に二度寝するようになってしまった. しかし, カギを取りに行く過程で明かりをつけることができた場合には, そのまま起きることが出来た. また, 一番の難関である「布団から出る」ことは出来たため, 大きな一歩だといえる. 今後は上述の通りナンバーキーを利用したいと考えている.

# 6　まとめ

今回は目覚まし時計にカギを掛けることで寝坊を防止する方法を紹介した. 殆どの材料が 100 円ショップで入手できる経済性, カゴの大きさを変えることで多くの目覚まし時計に対応できる汎用性, 錠の数や種類を容易に変えられる拡張性が実現できた. また, 使用する度に慣れてしまう問題も, 上述の様に自分に合った錠の組み合わせにすることで解決できると思われる. ぜひ, 読者の方も自分にだけの組み合わせを見つけて休日の午前中を有効に過ごしていただきたい.

# 7　今後の展望

今回は目覚まし時計に「カギ」という機能を追加することで, 寝坊を防止しようとした. そのため, 非常にアナログなものとなり,IGGG らしくなかったかも知れないと少しだけ反省している. 今後は来年に発売される「Arduino 101」を活用してスマートフォンと連携する照明などを製作し, さらなる寝坊の対策を行って休日の午前中を守りたいと考えている.

# ぶうまんが Vol.1

Buu[†]

## 概要

今は昔、およそ <censored /> 年前。そこには HTML を手で入力し、<BLINK>、<MARQUEE> などのタグや GIF アニメ、カラフルなアクセスカウンターで華やかに装飾した自作のホームページを、50MB ほどの無料スペースに FTP で転送して公開する文化があったという。本研究グループではその文化について調査を進めるうちに、とある文献に辿り着いた。本稿ではその文献の登場人物について述べ、文献の一部を抜粋して掲載する。

なお、文献の登場人物と筆者の名称は無関係である。

## 登場人物紹介

ぶう

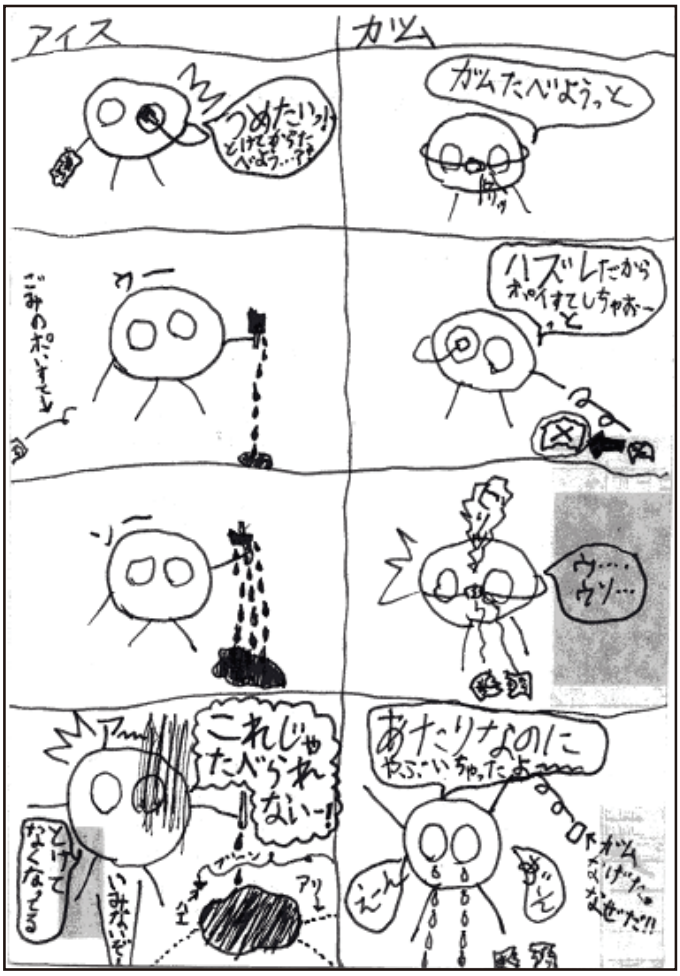

主人公と思われる、謎の生き物。
全体的に残念な雰囲気を漂わせているが、実は何かと良い奴かもしれない。

ぶう子

リボンがとてもキュートな、
花も恥らう乙女かもしれない。

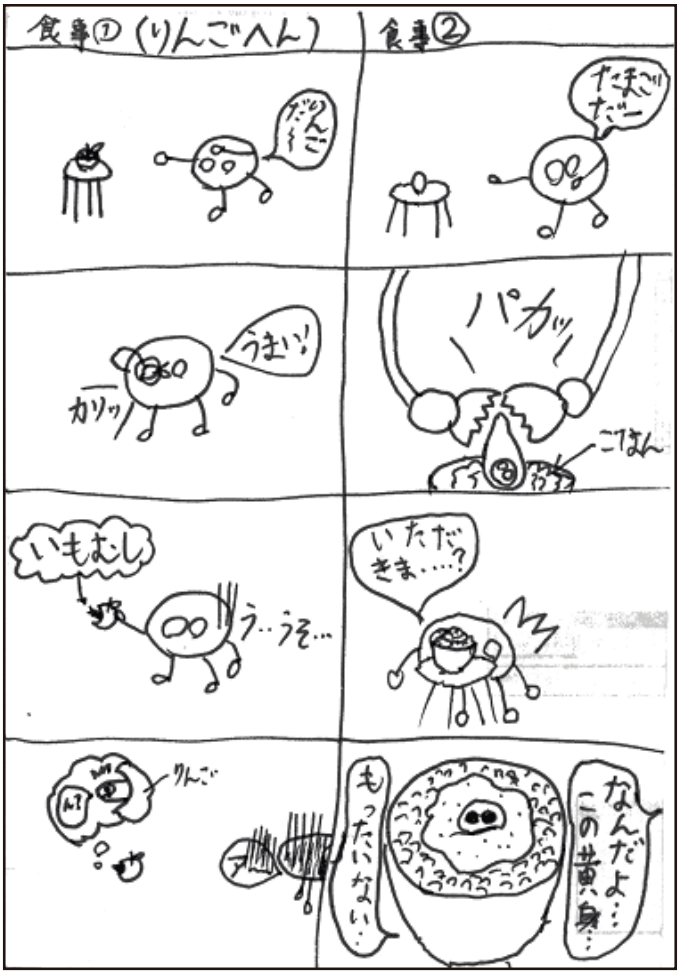
？

主人公の友人かもしれない。
<censored /> 年前はもっと詳細な設定があったが、その真相は忘却の彼方。

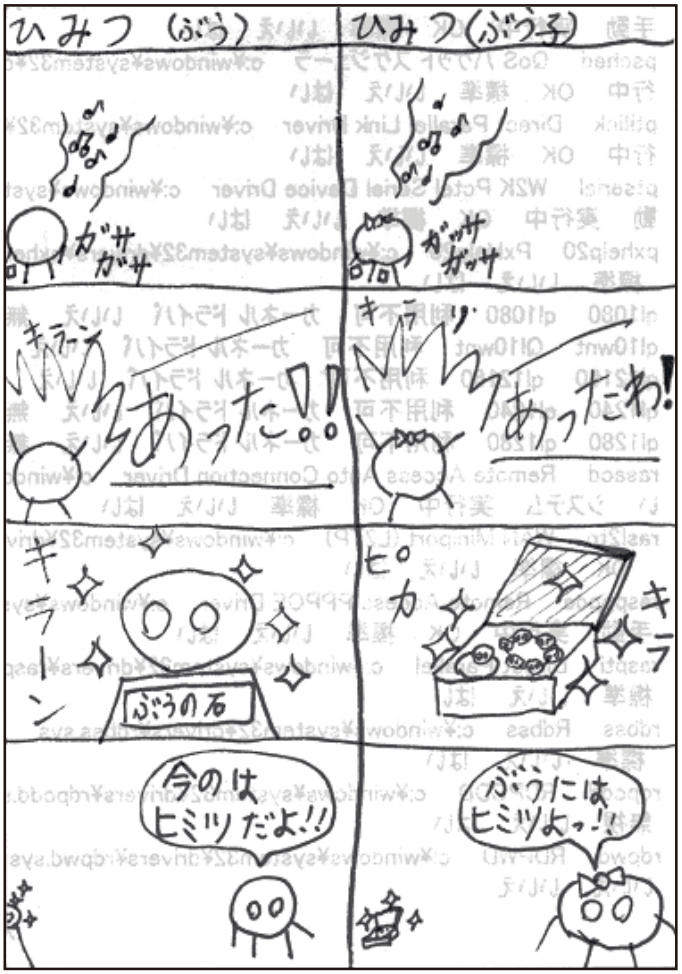


[†] Twitter: @buu0528


食事① (りんごへん)
だんご〜
食事②
だんごだー
うまい!
カリッ
パカッ
こなん
いもむし
う..うそ...
いただきま...?
りんご
もったいない
なんだよ...この黄身...


ぶうの大発明①
なんだ?これ?!
こうかな?
やったーできたー
うわ〜!!
ブン
ぶうの大発明②
なんだ、このどうぐ
こうかな??
できたー
ピーッ
タスケテ...


Aか?Bか?①
B
A
Bに入った。
A
B
D
Dに入った。
X
Y
Z
D
A
Xに入った。
A
B
C
D
E
F
G
H
I
J
K
X
Y
Z
Aか?Bか?!②
ギギギ...
A
B
D
E
F
G
D!
Dに入った。
うわ〜死ぬ〜...


ひみつ (ぶう)
ガサガサ
キラーン
あった!!
キラーン
ぶうの石
今のはヒミツだよ!!
ひみつ(ぶう子)
ガサガサ
キラリ
あったわ!
ピカ
キラ
がうにはヒミツよっ!!

# あとがき

### ひげ

「IGGG に Haskell を浸透させよう委員会」会長のひげです(会員は約 1 名). IGGG は Python のシェアが高いので日々奮闘していますいつかIGGGからPythonを追い出してHaskell一色にするのが夢です．うそです. Python も大好きです．共存の道を歩きましょう．ちなみに Ruby も好きですよ．

### ism67ch(@ism67ch)

今回は最近ハマってる「ごちうさ」で記事を書かせて頂きました．研究会の予稿提出とダダかぶりしてしまい，執筆が遅れて noob には迷惑かけちゃいました（＞＜） パッチ，絶対領域の次はティッピーでしたねｗ 次回は何にしようかな～

### 大宮

執筆するのは 3 回目，あとがきを書くのは 2 回目の大宮なのです！IGGG では突撃担当と闇鍋担当なんかをしているかも．日々の部活の方で闇鍋！闇鍋！言っていたら IGGG Meetup の扱うテーマで「闇鍋」が例として取り上げられました！！！（どうしてこうなったし）今回もいつも通り闇鍋について書いているかと思います．冬といえば，「鍋」．鍋といえば「闇鍋」と闇鍋がおいしい季節になりました．寒い季節は闇鍋に挑戦するのはどうでしょう？！もちろん闇鍋はオールシーズン行えます．最後に闇鍋は非常に安全な食べ物です．ルールを守って楽しく闇鍋！！してみてください．

### れぽ(@gokinaka)

今回はネタのなさの余り，自分の専門の内容がっつりやってしまいました．反省しています．
次の部誌では何か作って載せたいなと思います．

### トム

こんにちは．トムです．今回は ROS について書きました．ROS でシステム設計からシミュレーションまで簡単に，コストが掛からずに目に見えるものができるので結構楽しみながら勉強しています．昨日国際ロボット展にも行ってきました．そこでも ROS を使ったデモが多数出展されていてロボットに興味がある人は ROS を勉強して損はないのではないかと勝手に思っています．

### 風土

3 度目まして，風土と申します．動画作ってる人です．なので今回も動画作りの思い出というか，日記のようなものを書きました．参考なったでしょうか？ならないですよね．参考になるような文章を書きたい人生だった.....

### UTSUTSUKU

はじめまして，初投稿の UTSUTSUKU です．今回は，すごくローテクな目覚まし時計について紹介しました．IT とは全く関係ない内容でしたが，実用性はあると思います．次回は IGGG らしい目覚ましデバイスを紹介したいです．

### buu

docker が楽しいので「めんどくさがりのための docker 」って感じで書きたいなーと思っているうちに気づけば締め切り前日でした．最近 Windows Server 2016 でもコンテナ型仮想化がサポートされ，ますます Agile とか DevOps という単語を目にする機会が増えそうな今日このごろですが…… 今回は 10 年ほど前に遡ってしまいました．中 2 といえばみんなも黒歴史サイト作るよね！ねっ…！？そしてこのような歴史がネットの海の深いところに沈んでいると思うと，深層ウェブは表層で検索エンジンにインデックスされた静的ページのおよそ 500 倍存在する（ Wikipedia より）と言われているのもうなずけます．深層ウェブの非構造化ビッグデータを分析したら，新たな知見が得られそう．ところでラブライブ！劇場版 BD 最高でしたね．劇場で 9 人 x2 周ぶん観たのに飽きません．いまが最高！！


| | |
|---|---|
| 発行日 | 2015/12/30 |
| 発行元 | 群馬大学電子計算機研究会 |
| 連絡先 | contact@iggg.org |
| 印刷 | 西岡総合印刷株式会社 |

*IGGG Journal "Lollipop" Vol.03, Comic Market 89 Edition*

# 群馬大学電子計算機研究会 "Lollipop" Vol. 03 C89 版

2014 年 5 月に突如立ち上がったサークル,「群馬大学電子計算機研究会 (IGGG)」のメンバーが, 相も変わらず計画性なく C89 に応募してしまったがために締め切りギリギリでヒーヒー言いながら執筆した部誌第 3 号です. 次から計画的に作りましょう ( 前回も前々回も言ってた ). IGGG は Information technology researching society of the Gunmer, by the Gunmer, for the Gunmer の略称の模様. グンマーの、グンマーによる、グンマーのための IT 研究会, ということでアルゴリズムや機械学習といったソフトな分野から制御理論や目覚まし時計 (?) といったハードな分野まで, 幅広く集めた一冊となっております. ぜひお楽しみください。

IGGG

発行／群馬大学電子計算機研究会
@IGGGorg http://www.iggg.org/