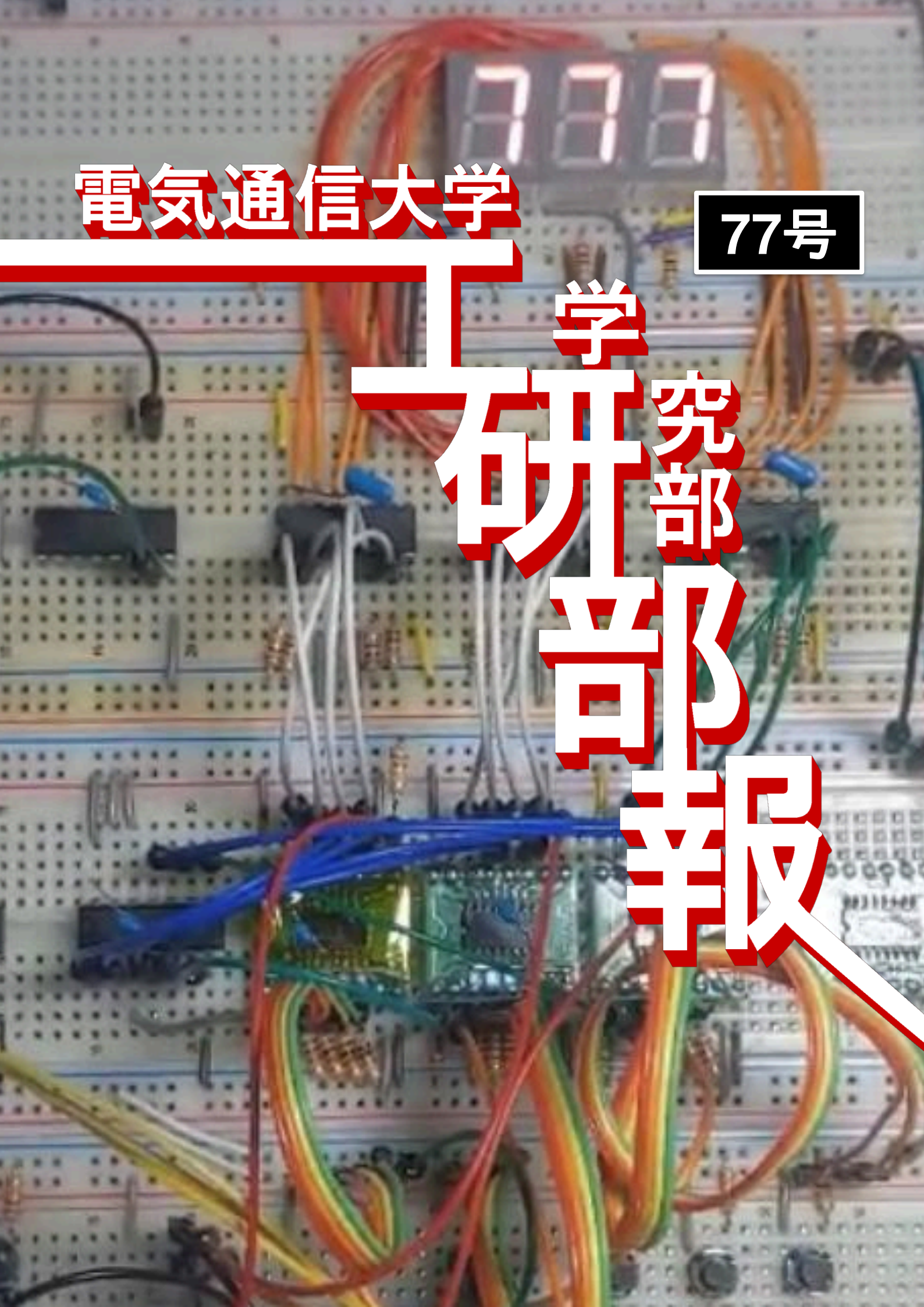


電気通信大学

77号

研究部報

学

目次

初めての Linux Kernel Module

21 hosi

「いつか特権命令を利用したプログラミングをしてみたい」という思いがあったので、初めて Linux Kernel Module を制作してみました。その際に苦労した点などを部報にしてみました。

1 Kernel Module とは

Linux Kernel Module は必要に応じてロードとアンロードができるバイナリファイルです。モジュールをロードすることによって Kernel の機能を追加することができます。モジュールの動作は特権モードで行われるため、ユーザーレベルでは禁止されている命令を使用することも可能です。

身近な例として、ハードウェアを利用する際にインストールするデバイスドライバなどが挙げられます。

2 開発環境

今回は Windows 11 上の WSL2 に環境構築を行い、開発を行いました。

- KernelVersion:5.15.167.4-microsoft-standard-WSL2+

3 環境構築

WSL 上の apt では Kernel ヘッダーをインストールすることができないようなので、自分でビルドを行います。実際に利用するカーネルのバージョンに合わせてモジュールを制作する必要があります。

Code 1 環境構築

```
#自分のカーネルのバージョンを確認する
$ uname -r
5.15.167.4-microsoft-standard-WSL2+
$ git clone https://github.com/microsoft/WSL2-Linux-Kernel.git
$ cd WSL2-Linux-Kernel
$ git checkout linux-msft-wsl-5.15.167.4

#以下でビルドを行う
$ sudo apt install build-essential flex bison libssl-dev libelf-dev
$ cp Microsoft/config-ws .config
$ make
```

4 ソースコード

Code 2 C Source Code

```
#include <linux/module.h>

MODULE_LICENSE("GPL");

static int hello_init(void)
{
    printk("hello Kernel world\n");
```

```
8       return 0;
9   }
10
11  static void hello_exit(void)
12  {
13      printk("Goodbye Kernel world\n");
14  }
15
16  module_init(hello_init);
17  module_exit(hello_exit);
18
19  }
```

Code 3　Makefile

```
1       obj-m := hello.o
2
3   all:
4     make -C ../path_to/WSL2-Linux-Kernel
        M=`pwd` modules
5   clean:
6     make -C ../path_to/WSL2-Linux-Kernel
        M=`pwd` clean
```

5　結果

make 後にモジュールのインストールを行います。

Code 4　結果

```
1   $ ls
2   Makefile  hello.c
3   $ make
4   $ ls
5   Makefile  Module.symvers  hello.c  hell
        o.ko  hello.mod  hello.mod.c  hello.
        mod.o  hello.o  modules.order
6   $ lsmod
7   Module Size Used by
8   $ sudo insmod hello.ko
9   $ sudo rmmod hello.ko
10  $ dmesg | tail -2
11  #moduleをロードした際のメッセージ
12  [ 9866.288659] hello Kernel world
13  #moduleをアンロードした際のメッセージ
14  [ 9871.345146] Goodbye Kernel world
```

これによって kernel から出力されたメッセージが確認できました。

6　まとめ

今回の記事では、WSL2 を利用した Linux Kernel Module 開発の初歩を説明しました。

これによって、ユーザー空間からは利用できない特権命令を利用することができるようになりました。

今後は実際に特権命令を利用したモジュールを開発するなどして、利用していきたいと考えています。

参考文献

[1] WSL2 でカーネルモジュールを作る 2022, https://qiita.com/mangos_/items/20a700fe6f494cc0fdba

[2] The Linux Kernel Module Programming Guide 2025, https://sysprog21.github.io/lkmpg/

バイクにスマートモニタを載せた話

21 らぼねこ

2025 年 3 月 31 日

1　檜原街道にて

私は普通二輪免許を取得後すぐにスズキのディーラーに駆け込んで現在も愛車としている GSX250R と運命的な出会いを果たし，初めての遠出の帰りに東京都の西部にある檜原街道を走っていた．12 月の夜のことであった．

土地勘のない場所であったので，私はインカム[*1]から聞こえるスマートフォンのナビアプリの音声を頼りに走行していた．しばらく走行していると不安が募ってきた．「ナビアプリの音声案内を聞き逃したのではないか？」「インカムとナビアプリの連携が切れたのではないか？」「スマートフォンの電池が切れたのではないか？」この不安が的中していたら，私はすでに道を間違え，自宅から遠ざかっているかもしれないし，もしかしたら帰れないかもしれない．数キロごとにバイクを降り，ナビアプリを確認した．

右左折をせず道なりに数十キロ走り続けるような場合，走っている間は特にナビからの音声案内がないので，何らかの原因で音声案内が途絶えても正常な場合と聴覚情報が変化しない．どうにか帰宅した私は，バイクにスマートモニタを取り付けることでこの問題を解決しようとした．

2　スマートモニタの動作

バイク用のスマートモニタはスマートフォンと連携し，運転中の経路確認などを可能にする．バイクにスマートフォンをマウントする方法もあるが，これはバイクの振動がスマートフォンに伝わって故障を招いたり，スマートフォンの電池が消耗するなどの問題もあるため，今回はスマートモニタを別途取り付けることとした．

今回取り付けるスマートモニタとして Kaedear 社製の KDR-D11 を選定した．付属していた電源アダプタからは赤色，黄色，黒色の 3 本の電源線が出ていた．黄色をバイクの ACC 電源 (アクセサリ電源) に接続し，赤色をバッテリーの正極，黒色を負極に接続すると，メインスイッチ[*2]が ON の場

[*1] ヘルメットに装着して使う無線スピーカーのような機器．スマートフォンと連携して走行中にナビの案内や音楽を聞いたり，通話ができる．

[*2] バイクの鍵を差して回すことで ON, OFF の切り替えができるスイッチ．

合に ACC 電源に負担をかけずバッテリーから電源をとれるようであった．私はバイクの電源回路について概ね図 1 のように理解していた．もちろん，実際のバイクではこの図にヒューズやセルモータ，灯火類などが加わることだろう．また，実際にトランジスタを利用しているかはわからないがモニタの電源線の役割については図 2 のように理解した．

そこで，黒色の線を GND とし，赤色，黄色の線にバッテリーの正極に相当する電圧を印加することで，動作確認を行うことにした．安定化電源で 12.6 V を印加したところ，モニタが動作し，スマートフォンとの連携も正常にできた．消費電力は 150 mA 程度であった．

3　ACC 電源の取り出し

モニタを取り付けるためには，ACC 電源およびバッテリーの正極と負極を露出させ配線する必要がある．まずは ACC 電源を取り出すことにした．ACC 電源は灯火類などバイクの様々な箇所に引き回されているので様々な取り出し方が存在するが，今回は原状復帰の容易さ等を考え既製品のハーネス[*3] を利用することにした．

タンデムシートの下にはテールランプにつながれたカプラがあり，いったんこれを外してハーネスを間に割り込ませることで ACC 電源を取り出すことができた．この後両端にそれぞれギボシ端子のオス，メスを成端したコードをこのハーネスにつなぎ，図 4 のようにメインシートの下まで伸ばしておいた．

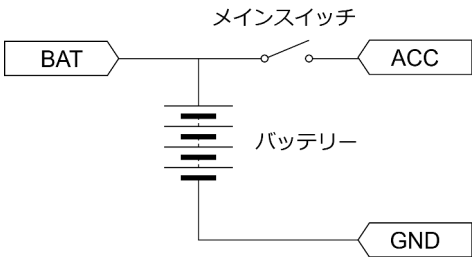


図 1　バイクの電源回路の模式図

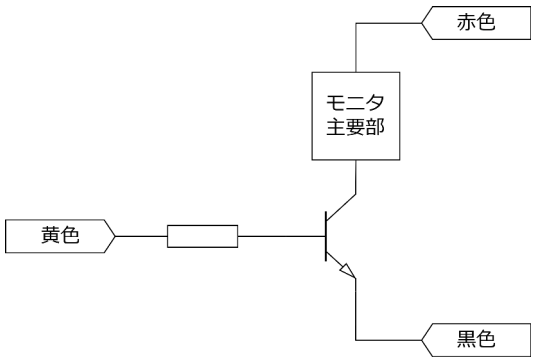


図 2　モニタの電源線の役割

図 3　安定化電源を用いた動作確認の様子

[*3] 電源取出しハーネス https://kitaco.co.jp/goods_detail/36088

4　メインシート下からハンドルまでの通線

スマートモニタはハンドル付近に取り付ける一方で，バッテリーはメインシートの下にあるので，この間の通線作業が必要である．

通線工具として不要になった VGA ケーブルの両端を切り落としたものを用意し，電源アダプタのコードの端にビニルテープで仮につないで延長した．VGA ケーブルを一度メインシート下からカウルの内側を通ってバイクの進行方向左側にあるカウルの隙間から引き出した．これを再度カウルの隙間に入れ，ハンドル付近から引き出した．通線を 2 段階に分けて行うことで，カウルの取り外しなどを省略し，長距離の通線も避けて容易に作業ができた．

5　動作確認と仕上げ

図 5 のように，アダプタの電源線をバッテリーと ACC 電源に接続し，もう一方の端にモニタ本体を接続してメインスイッチを入れると，モニタが動作した．うれしい！

その後，配線の接点を自己融着テープで絶縁し，配線が動かないよう結束バンドでバイクの各部に固定した．さらに，本体を別途設置したマウントバーに取り付け[*4]，作業を完了した．

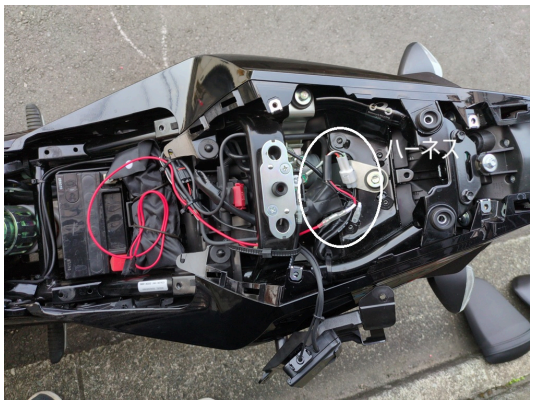


図 4　テールランプからの ACC 電源の取り出し

6　設置後

こうして運転中でも視覚的な情報を得られるようになり，遠出も快適にできるようになった．さらに，スマートフォンを出さなくてもナビアプリ等の操作が可能になったほか，付近の道路のおおまかな形状や渋滞予測等，運転中に得られる情報が大きく増加した．うれしい！ うれしい！

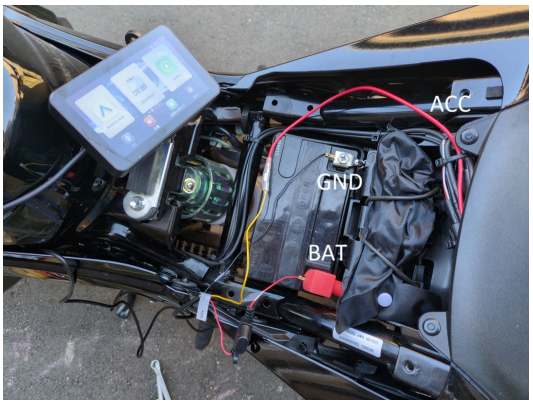


図 5　動作確認の様子

[*4] セパレートハンドルの車種の場合はマウントバーを使わない場合取り付け位置に困ってしまうことが考えられる．私はここまで取り付け位置を全く考えていなかったので作業途中で府中 2 りんかんにマウントバーを探しに行くことになった．

AVR の UPDI 書き込み機をつくったよ

21 Valkyrie

AVR マイコンの書き込み方式の一つである UPDI を使って書き込みを行うための書き込み機を作成した。

1　方法

回路図は以下のとおりです。

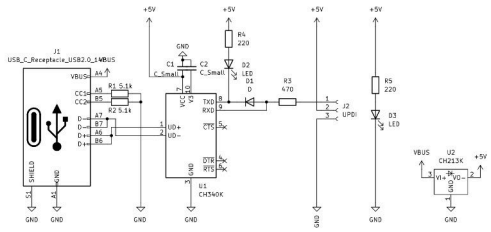


図 1　回路図

フットプリントは以下のとおりです。

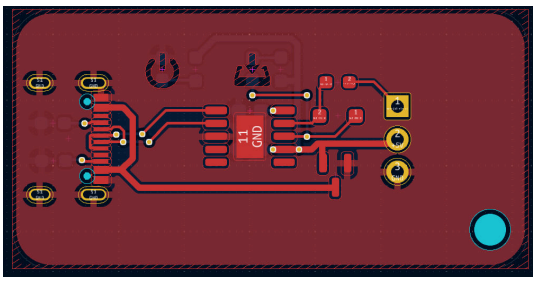


図 2　フットプリント

図 2 に電源マークとダウンロードマークがあります。このマーク、光ります。以下の図 3 に示すようにソルダーレジストと銅箔面をそれぞれ除去すると裏面の LED の光が表面に透けて見えるようになります。

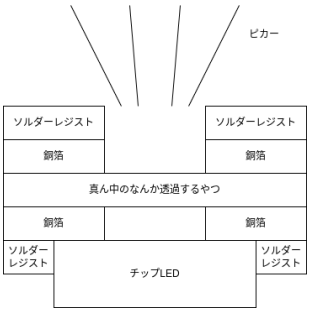


図 3　レイヤー

ソルダーレジストを消すには Mask を指定します。銅箔面を消すにはベタグラウンドの塗りつぶし禁止範囲に指定します。

2　結果

完成したものがこちらです。

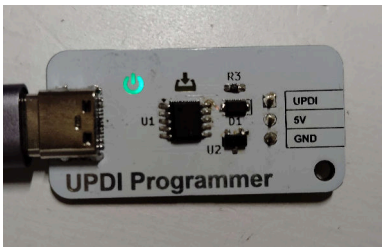


図 4　完成品

この部報がカラー印刷になっているかデータ配布になっているのかは不明ですが、光っています。完成！ ほっこり。

OCI を使って無料で k8s クラスタを運用しよう

21 ぼいど

21 のぼいどです。普段はインフラ系を主にやっていて、サークルのインフラ基盤の開発や運用などをしてます。今回は Oracle Cloud を活用して無料で Kubernetes クラスタを運用する方法についてご紹介します。

1 はじめに

Kubernetes って便利ですよね。でも、クラスタ運用するのは苦しい…と思ったことはありませんか？頻繁にあるアップデートやら、Ingress の LB の設定から、PVC の設定まで色々とやることが多いと思います。

AWS や GCP などのクラウドを使うと、EKS(Elastic Kubernetes Service) や GKE(Google Kubernetes Engine) を使うことで、マネージドな Kubernetes クラスタを使うことができます。しかし、こういったサービスは結構コストがかかります。

なんとか 0 円で Kubernetes クラスタを運用できないかなぁと試行錯誤して、いい感じに運用する方法を見つけたのでご紹介します。

2 Oracle Cloud

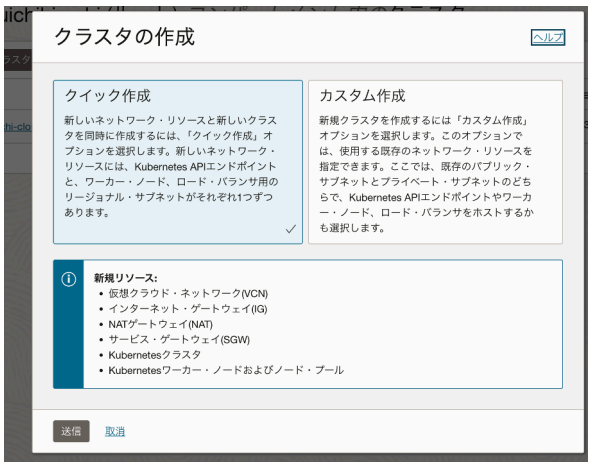
Oracle Cloud(OCI) は、言わずと知れたあの Oracle 社が提供しているクラウドサービスです。あまり有名ではないかもしれませんが、実は隠れた最強クラウドサービスです。OCI の良い点を 2 つご紹介します。

1. **k8s の Control Plane が無料**
   OCI にも EKS などと同様に Oracle Kubernetes Engine(OKE) というマネージドな Kubernetes クラスがあります。このサービスでは Control Plane が無料で利用でき (一部条件あり)、ノードにのみ課金されるようになっています。
2. **潤沢な ARM インスタンスが無料**
   OCI の特徴の 1 つは潤沢な無料枠です。なんと、4vCPU、24GB 相当の ARM インスタンスを無料で利用できます。先ほど OKE でノードに課金されると言いました。この無料枠の ARM インスタンスをノードとして使うことで、ほぼ無料で Kubernetes クラスタを運用できます。
   ここまでの量のリソースが無料で使えるクラウドサービスは他にないのではないでしょうか？

3　k8s クラスタの構築

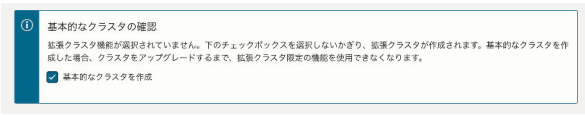
では早速、OCI で Kubernetes クラスタを構築してみます。アカウント作成などは他のクラウドとほぼ同じですので、割愛します。

1. クラスタの作成
   今回はクイック作成でやっていますが、細かいネットワークの設定などをしたい場合はカスタムを選んでください。



2. ワーカーノードの設定
   ここが大事なポイントです。必ずノード・シェイプを「VM.Standard A1.Flex」にしてください。また、クラスタ全体で OCPU とメモリー数の 4 OCPU、24GB 以下になるようにします。今回は「2 OCPU、12GB」で 2 つのノードを作成しました。

3. クラスタ作成前の確認
   作成前に確認画面が出るので、ここで必ず「基本的なクラスタの作成」にチェックを入れてください。これにチェックを入れていないと拡張クラスタと呼ばれるものになり、追加で料金が発生してしまいます。



   最後に「クラスタの作成」をクリックするとクラスタが作成されます。作成には数分かかりますので、気長に待ちましょう。

kubectl の設定などは少し複雑なので、OCI の公式ドキュメントを参考にしてください。以下のように無事にノードが 2 つ作成されれば成功です。

```
$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
10.0.10.146 Ready node 38d v1.31.1
10.0.10.85 Ready node 38d v1.31.1
```

4　無料運用のための注意点

OCI でクラスタを利用するためには、いくつか注意するポイントがあります。無料

だと思っていても思わぬ課金が発生することがあるので注意が必要です。

- Ingress Controller
  Ingress Controller を使う場合、デフォルトで設定させている OCI の LB を使うと課金が発生します。無料で運用するためには、NodePort を使うか、他の Ingress Controller をセットアップする必要があります。
- PVC
  PVC を使う場合、デフォルトでは OCI の Block Volume が使われます。Block Volume の無料枠は 200GB なので、その範囲に収める必要があります。ただし、そのまま Block Volume を PVC として使うのはお勧めしません。なぜなら、Block Volume の最小容量が 50GB なので、それ以下の容量で PVC を要求しても自動的に切り上げて 50GB の Block Volume が作成されてしまいます。これでは実質的に 4 つの PVC しか使えなくなってしまいます。

5 Longhorn の活用

PVC の問題を解決するために、Longhorn という Kubernetes のストレージプラグインを使うことをお勧めします。方法としては、ノードに予め 100GB の Block Volume をマウントしておき、これを Longhorn のストレージとして利用し、PVC は Longhorn の CSI を使って作成するようにします。

注意点として、無料枠に含まれるのは VM にデフォルトでアタッチされるブートボリュームも含めて 200GB な部分です。ブートボリュームの最小値は 48GB なので、VM が 2 つある時点で既に自由に使えるボリュームは 200-48*2=108GB です。これを超えると課金されてしまうので注意してください。ブロックボリュームの最小値は 100GB なので、今回は 100GB のブロックボリュームを 1 つ作成して Longhorn のストレージとします。

簡単にインストール方法を紹介します。

1. VM に Block Volume をマウント
   予め 100GB の Block Volume を作成します。次に k8s のワーカーノードで、作成した Block Volume をアタッチします。

2. ストレージをマウントする
   一般的な VM のストレージをマウントする時と同様に、一度 VM に入って自動マウントの設定を追加します。ノードに SSH する方法はいくつかありますが、kvaps/kubectl-node-shell などの krew プラグインを使うと簡単に SSH できます。
   手順としては、mkfs などを使ってボリュームを ext4 でフォーマットして、`/etc/fstab` にマウント設定を追加します。以下のような設定です。複数設定する場合、ここでマウントパスはノー

ドで同じなるようにしてください。

```
UUID="01b12f6d-280c-46a0-a2b0-332
    fd739ab76" /mnt/volume ext4
    defaults,nofail 0 2
```

3. Longhorn のインストール
   Longhorn 自体は Helm を使って簡単にインストールできます。以下のコマンドを実行してください。

```
$ helm install longhorn longhorn/
    longhorn --namespace longhorn-
    system
```

4. Longhorn の設定
   そのままでは、Longhorn で利用するストレージがないので、先ほどマウントしたストレージを Longhorn で使えるように設定します。以下のコマンドを実行して、設定ファイルを開きます。

```
$ kubectl edit nodes.longhorn.io <Node
     Name> -n longhorn-system
```

開いたファイルで以下のようにディスクの設定を追加してください。ここで path は先ほどマウントしたパスを指定してください。

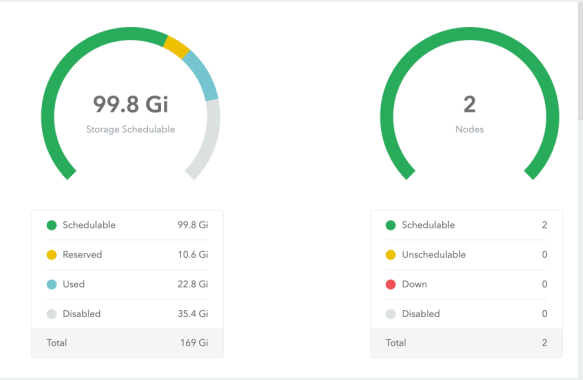
```
spec:
  disks:
    custom-disk:
      allowScheduling: true
      diskType: filesystem
      path: /mnt/volume
      tags: []
```

100GB を超えて複数作成したい場合 Node Name を変えて、他のノードに対して同じ設定を行ってください。

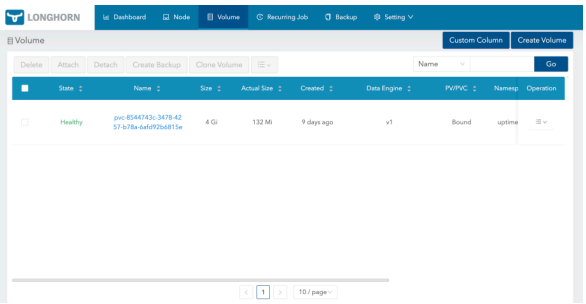
5. UI での確認
   Longhorn の UI にアクセスして、ディスクが正常に認識されているか確認してください。open-svc などのプラグインを使って以下のようにアクセスできます。

```
$ kubectl open-svc longhorn-frontend -
    n longhorn-system
```

正しく設定できていれば、約 195GB ぐらいのストレージが利用可能になっていると思います。



Longhorn を利用するには、PVC を作成する際の StorageClass を longhorn で指定するだけです。PVC を作成すると、以下のように Volumes に PVC が作成されているのが確認できます。



Longhorn にはこの他にも S3 にバックアップをする機能など色々と便利な機

能がありますので、ぜひ使ってみてください。

6 まとめ

今回のクラスタは HA ではないのでしっかりとしたサービスの運用は難しいかもしれませんが、学習用や個人開発などには十分使えると思います。これを機に是非 Kubernetes に入門してみてはどうでしょうか？

3D プリンタは日用品

22 あづみ

工学研究部の特徴的な機材として、3D プリンタが挙げられる。謎の船のオブジェを作って満足するとか、ビックリドッキリメカを作るくらいしか用途がないように思われがちだが、実用的な日用品を作ることにも大いに役立つ。今回は、自分が 3D プリンタで作った日用品を紹介する。

1. 3D プリンタとは

3D プリンタとは、比較的手軽に立体物を CAD データをもとに現実世界に召喚することができる~~魔道具~~工作機械である。その手法には色々あるが、現在個人用途で主流なのは溶かした樹脂を積み重ねて造形する FDM 方式。

FDM 方式では、下から上に積み重ねる都合上、下から支えられていない形状(特に片持ち梁)を作りにくいという制約がある。サポート材と呼ばれる支えの部位を作ることで対処できるが、取り除く手間や材料の無駄が発生する。今回紹介する物品も FDM 方式の 3D プリンタで造形したものであり、この点を考慮して設計している。

2. ケーブル収納

初めて 3D プリンタを使って作ったのは、使っていないケーブルを百均のワイヤーネットに掛ける形で保管するための部品である。まっすぐ掛けることで、絡まらず、出し入れしやすく、一覧性よく保管できる。

当初はコネクタ部をひっかけるようにしていたが、形状によってはうまく掛からなかったり、重いものには耐えられなかったりした。次の世代では、ケーブルにフックをはめ込む形にしたが、ケーブルの太さを正確に測定し、さらに交差を考慮して設計する必要があり、手軽さが犠牲となった。最終的には、次に紹介するような配線整理に使うものと兼ねる形にした。

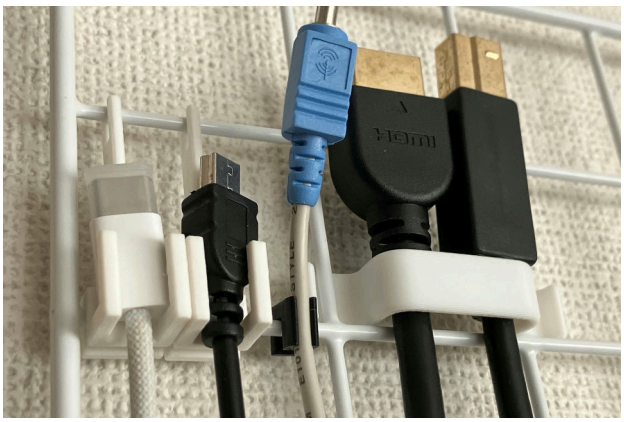


図 1: 左から初代 2 つ、二代目、三代目

3. 配線整理

使用中の電源コードや LAN ケーブルなどの整理にも活用できる。こういったケーブルは針金や可逆結束バンドなどでワイヤーネットにくくりつけることで整理して

いたが、脱着が面倒だったりしてあまり快適でなかった。最初は使っていないケーブル同様ひっかけるようにしていたが、ケーブルには上方向に力がかかる場合もありあまり適切でなかった。その次は穴の両側にかみこんで固定するようにしたが、脱着が面倒になった。最終的にはかぶせてはめ込む形状とし、これは使っていないケーブルの整理にも使えるものとなった。

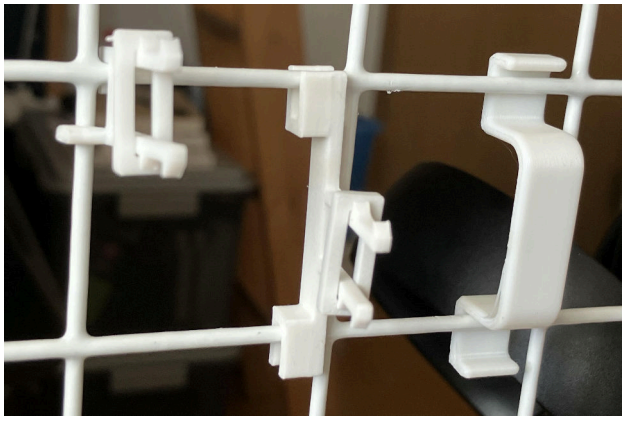


図 2: 左から初代 2 つ、二代目、三代目

また、スイッチングハブを縦向きにワイヤーネットにくっつけたかったが、ギリギリ重力に負けてしまったので吊るして支える部品も作った。

4. ケーブル携帯

ケーブルには家で常設するもののほかに持ち運ぶものもある。束ねて鞄に入れるだけでは跡がついたり引っかかったりするので、ケーブルを丸く格納できる容器を作った。外壁を一層として連続的に造形する方法を用いてみたところ、収縮してべこべこになったり少し使うと裂けたりしたが、使えないことはないくらいになった。

5. ピンセット立て

工研部室ではピンセットを普通のペン立てに立てていたが、ピンセット同士が絡まって取り出しにくいことが多々あった。そこで、仕切りのついたシンプルなピンセット立てを作った。先が曲がったピンセットのために 2 コマの領域も用意した。

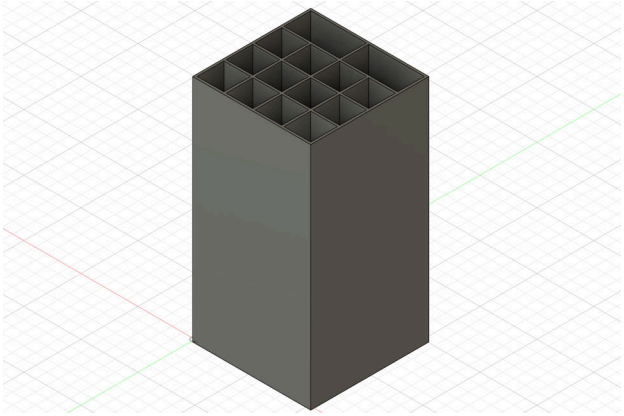


図 3: ピンセット立て

6. リモコン掛け

エアコンのリモコンがよくあるネジ山にひっかける穴が空いているタイプなのだが、このためだけにネジを壁に打ち込むのは面倒である。画鋲のように容易に設置できるフックが百均に存在するので、それにリモコンをかけられるようにした。

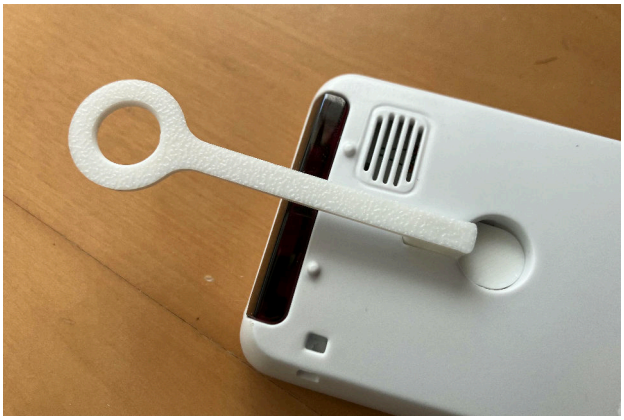


図 4: リモコン掛け

7. ティッシュ箱、AC アダプタフック

箱ティッシュをロフトベッドの梁部分に吊るすためのフックのようなものを作った。ぴったりのサイズにでき、取り出すときに動かないのでとても使いやすい。同様にモニタの AC アダプタの箱部分を吊るすためのものも作った。

図 5: 吊るされたティッシュ箱

8. キートップ引抜工具

キーボードのキートップを引き抜くための工具を作った。市販のものには金属ワイヤが使われている[1]が、細い樹脂でも意外と大丈夫だった。サポート材が不要となるようパーツを分けたところ、可動域が広がるという利点もあった。

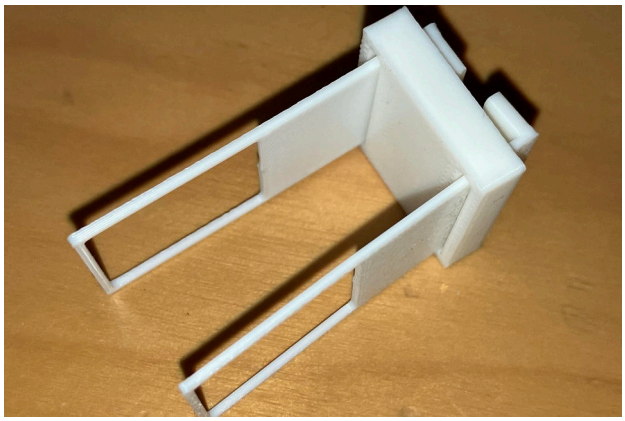


図 6: 3D プリンタ製キートップ引抜工具

9. LAN ケーブルのツメ

LAN ケーブルのツメが折れて固定できなくなることがまれによくある。そんな LAN ケーブルに被せてツメの代わりになることで対処するパーツがある[2]が、3D プリンタでも作ることができる。モデルは自作ではなく配布されているもの[3]を用いた。

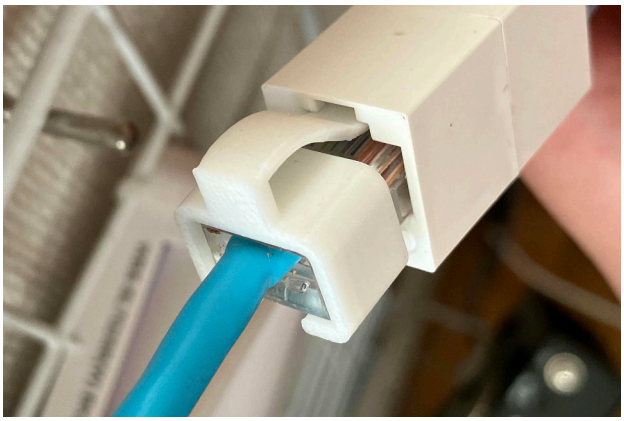


図 7: 修復した LAN ケーブル

10. 一家に一台 3D プリンタ

工研では、FDM 方式の 3D プリンタとしてBambu Lab の P1S と A1 mini を導入している。自分の入部目的の一つはこの 3Dプリンタを使用することであった。もちろん工研の 3D プリンタを借りてもよいが、日用品は自宅で欲しいときに作れるのが理想である。したがって、人類は一家に一台 3D プリンタを持つのがよいとされる。自分も途中から A1 mini を導入している。

でも、お高いんでしょう？　難しいんでしょう？　と思われるかもしれないが、技術の進歩により、3D プリンタは安価かつ特別な知識なしで使えるようになってきている。Bambu Lab の製品はその最たるもので、各種調整が自動化されていることによ

り、使用するための手順が大幅に簡略化されている。従来の世話の焼ける代わりにカスタマイズ性の高い 3D プリンタを好む人も存在するが、道具として使うのであれば Bambu のようなものが適している。お値段はセールであれば最安 3 万円から、通常時でも 5 万円からで、激安と言わないまでも十分現実的である。

Bambu Lab には比較的安価な A1 シリーズと、「囲い」があり使える材料の多い P1 シリーズがあり、後者が上位機種とされている。しかし、A1 シリーズは後発なぶん改善されている点が多々ある。たとえば、ベッドレベリング(原点出し)が高速化されているうえ、必要な部分に絞って行う機能まで追加されている。P1S では時間がかかるのを嫌ってベッドレベリングを無効にし、結果的に失敗して時間を無駄にすることがあったが、A1 シリーズでは時間を気にせず常に有効にしておける。それ以外の造形前準備の時間も圧倒的に短縮されている。さらに、流量キャリブレーションの自動化により、非純正や湿気るなどしたフィラメントでも造形品質の向上が期待できる。ついでに、ゴミが造形領域上に投げ出されることも減っている。基本的な材料である PLA しか用いない場合には A1 シリーズがおすすめ~~で、知らずに P1S を選んでいる人を見ると正直哀れ~~である。

A1 シリーズには P1S と同じ造形領域の A1 と、小型の A1 mini が存在する。ほかの機能にほぼ違いはない。上位機種に小型版はないので、A1 mini であれば P1S を選ばなかった言い訳も立つといえる。大型が欲しくなるころには P1S の後継機が出ているかもしれない。

参考文献

[1] ダイヤテック株式会社, 「FILCO KeyPuller 製品情報 | ダイヤテック株式会社」. 参照: 2025 年 3 月 22 日. [Online]. 入手先: https://www.diatec.co.jp/products/det.php?prod_c=706

[2] サンワサプライ株式会社, 「ADT-RJ45SOS-10【RJ-45 プラグ SOS（LAN ケーブル補修カバー・爪折れ修復)】つめが折れても安心。プラグのツメを復元する簡単取り付けの後付けラッチ。10 個セット。｜サンワサプライ株式会社」. 参照: 2025 年 3 月 22 日. [Online]. 入手先: https://www.sanwa.co.jp/product/syohin?code=ADT-RJ45SOS-10

[3] STECO3D, 「Ethernet RJ45 repair Remixed by STECO3D - MakerWorld」. 参照: 2024 年 12 月 14 日. [Online]. 入手先: https://makerworld.com/en/models/446389-ethernet-rj45-repair

3D プリンタを使ってみた

鳩鴨 (Hatogamo)

22 生の鳩鴨です．今回は大学生活最後の活動として 3D プリンタに挑戦してみました．

1　事の始まり

今回 3D プリンタに手を出したのは、マウントに不満があったからです．私はサバイバルゲームをしており、図 1 の遊戯銃 (以下、エアガンと呼ぶ) を所有しています．このエアガンを使う際、的を容易に狙えるようにする機器 (照準器と呼ばれる) を図 2 の様に取り付けています．この図では、図 3 の照準器が取り付けられています．照準器の取付では、図 4 のマウントレール (以下、マウントと呼ぶ) を用います．このマウントを用いて取り付けた場合、図 2 のようにエアガン本体から浮いてしまい、照準器を通して的を狙うのが難しくなってしまいます．私はこの問題に直面し、マウントより低い位置で照準器を取り付けられるパーツを欲していました．

図 1: 自宅の遊戯銃

図 2: マウントレールを用いた照準器の取付

図 3: 自分が所有する照準器

図 4: 照準器の取付に必要なマウントレール

2　作成したもの

私は、既存のマウントに満足できなかったため、部室の 3D プリンタを用いてより低いマウントを作ることにしました．作成したマウントは図 5 の様で、図 4 のマウントより照準器を低い位置に固定することができます．また、既製品と同じ方法でエアガンに取り付けることができます．

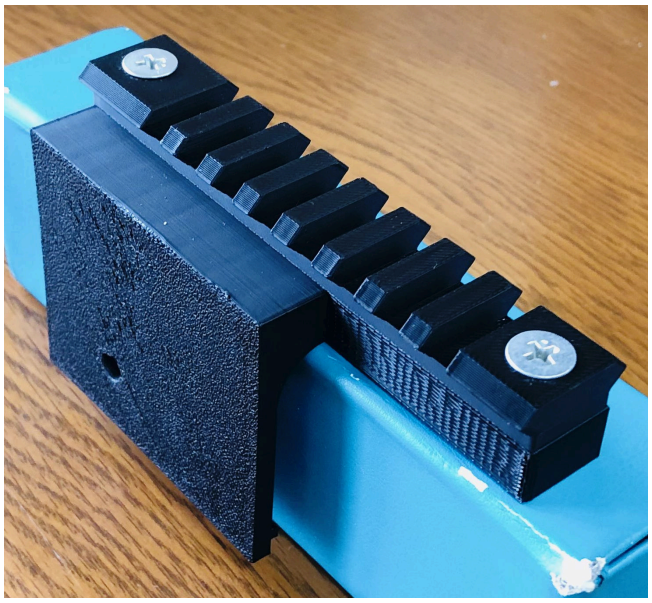


図 5: 作成したマウント

このマウントをエアガンに取り付けた様子は図 6 のとおりです．取付では既製品のネジを流用できるようにしたため、新しいネジを用意する必要がありません．

図 6: 作成したマウントの取付

3　作成物の外観

自作マウントのモデルは、Free CAD [1] を用いて作成しました．作成したモデルの外観は 7、図 8 の様です．前者は照準器の固定台で、台の凸凹した箇所はピカティニーレールと呼ばれることがあります．後者は固定台をエアガンに取り付けるための土台で、既製品に類似した構造となっています．マウント全体をこのように分けたのは、3D プリンタによるモデルの印刷精度を良くするためです．

図 7 の制作では、 ピカティニーレールの規格 [2] に従って、照準器の固定場所を作成しました．また、図 8 との結合に皿ネジを使うため、皿ネジの穴を両端に開けました．

図 8 の制作では、既製品の寸法を元に、エアガンに固定される部分を作成しました．特に、図 8b の穴と突起部分の位置関係と、突起部部分のサイズが、既製品と同じになるようにしました．これによって作成した土台を、エアガンにしっかりと固定することができました．照準器の固定台を載せる部分は既製品より厚くしました．これは、固定台と土台の結合に用いるナットの厚みに合わせたためです．

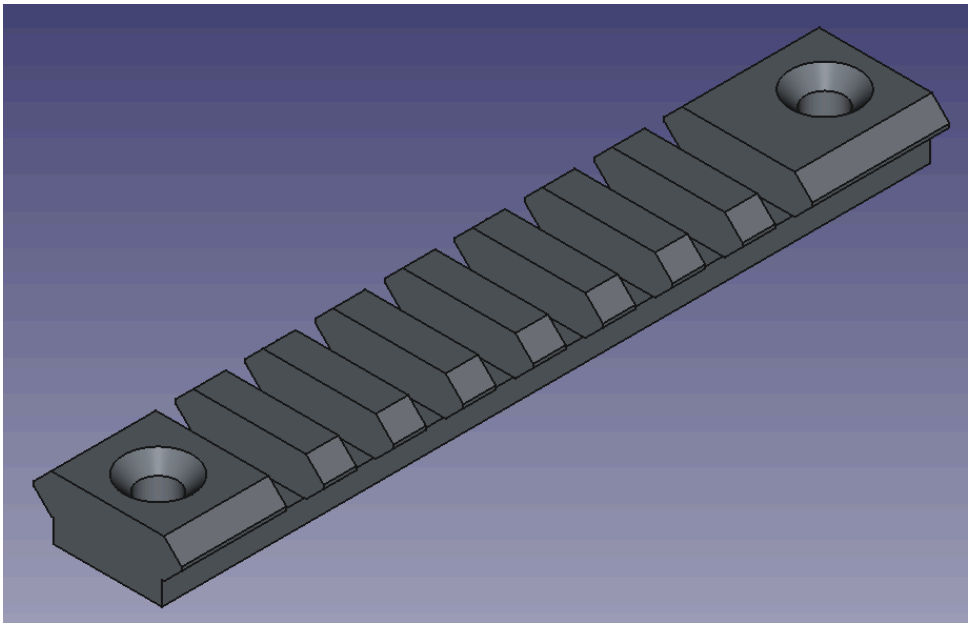
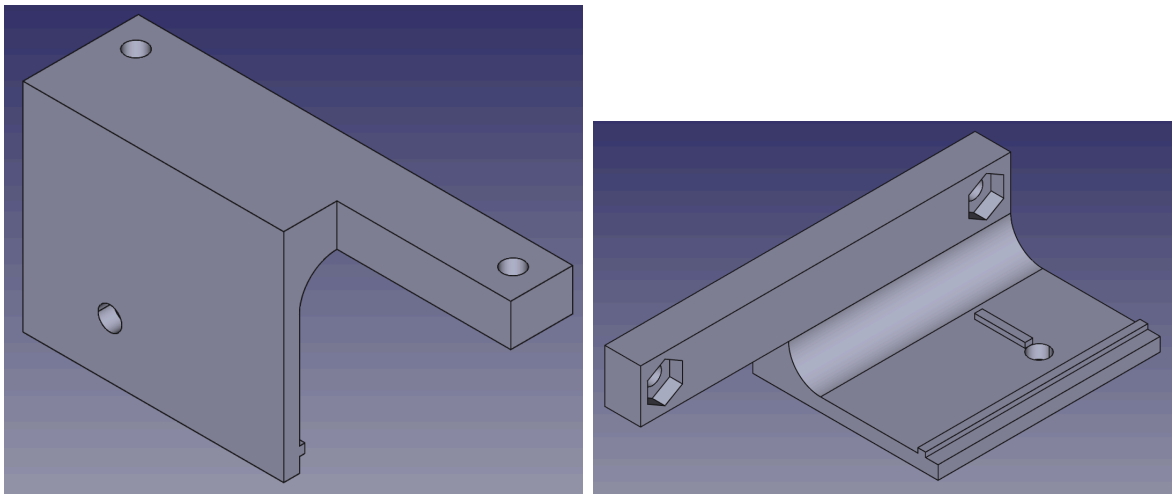


図 7: 自作マウントの照準器固定台



(a) 前　　(b) 後

図 8: 自作マウントの照準器固定台の土台

4 使用感

作成したマウントをエアガンに取り付け、その上に図 3 の照準器を載せてみたところ、既製品より低い位置で的を狙うことができました．また、手持ちのスコープも同様でした．しかし、マウントの固定台とエアガンの可動パーツが干渉してしまい、内部メンテナンスに手間が掛かるようになりました．また、図 8 の固定台を載せる部分のそこがエアガン本体と接触してしまい、接触部が傷ついてしまう恐れがありました．

5 おわりに

今回は 3D プリンタを用いて、既製品の改良版の作成に挑戦してみました．作成を通して部品計測の大変さやモデリングの難しさに何度も直面しましたが、実際に自分が作成したパーツが理想通りに使えた時は、その苦労を忘れるほど喜びました．

今後は、モデルの改良と、既製品のネジや工具への依存を無くそうと思います．また、他のパーツも作成してみたいと思います．

参考文献

[1] The FreeCAD Team. FreeCAD: Your own 3D parametric modeler. https://www.freecad.org/.

[2] United States Department of Defense. Dimensioning of accessory mounting rail for small arms weapons. https://quicksearch.dla.mil/qsDocDetails.aspx?ident_number=115317.

おみくじ Bot「Hakurei Shrine Bot」

22 へるくん

唐突に御神籤を引きたい時もある.

1　背景

電気通信大学には「バーチャルライブ研究会 (VLL)」というサークルがあり, そこの Discord サーバーには初音神社というおみくじを引けるチャンネルが存在する. さて, 筆者は VLL に加えて「東方 Project 同好会 電々。通信 (電。)」というサークルにも所属しており, 東方 Project は神社を題材としたストーリーやキャラクターが一定程度存在する. 神社と多少関連性がある電。にこそ, おみくじを引く Bot が必要であると考え, 制作に至った.

2　初音神社をはじめとした先行御神籤 Bot の課題

初音神社には以下の課題が挙げられると考える.

- ありきたりなおみくじの内容となっており、使用するにつれて飽きが生じてしまう
- 過去自分が引いたおみくじの内容を見返すことができない
- 連続でおみくじを引きたい際に一回ずつコマンドを送信するのは煩雑

これらの課題をできるだけ解決するようなおみくじ Bot を作成した.

3　作成したおみくじ Bot

名を「Hakurei Shrine Bot」と名付けて電。の Discord サーバーに導入した。図は 1 となっている。また、ソースコードは Github 上に掲載されており、以下のリンクから閲覧することが可能である。

`https://github.com/hel-kun/HakureiShrineBot`

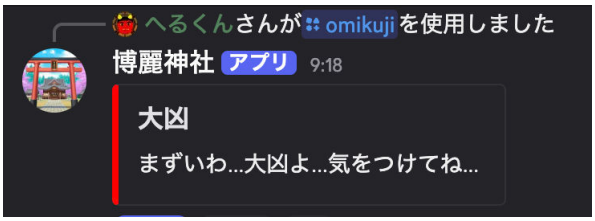


図 1　Hakurei Shrine Bot

3.1　各コマンドの紹介

- `/omikuji`：おみくじを引く
- `/omikuji_x10`：おみくじを 10 回引く
- `/history`：自分が引いたおみくじの履歴を確認する

3.2 技術面

この Bot は TypeScript で書かれており, Discord.js ライブラリを使用している. おみくじの内容は JSON 形式で保存されており、その内容が出る確率などが設定されている. この json ファイルの中身を変更することで, おみくじの内容を決めることができる.

おみくじの履歴も JSON 形式で保存されており, それを元に履歴を確認することができる. 各 Discord サーバごとに json ファイルを分けており, json のファイル名をサーバーの ID にしている. json の key はユーザの ID にしており, それを元に履歴を取得することが可能である.

3.3 今後の展望

Discord を動かしているサーバーを維持しつつ, より良い機能を思いつき次第追加していく予定である.

Raspberry pi 4B RaspberryPi OS Bookworm WiFi に繋ぎながら WiFi を飛ばす

22 もやし

アクセスポイントにしつつ WiFi クライアントとしても動作させる方法をまとめました。
環境：Raspberry Pi 4 Model B Rev 1.2, RaspberryPi OS Lite (bookworm)

1 手順

1.1 デバイスの確認

認識しているネットワークの一覧を取得するコマンド

```
iw dev
```

実行すると以下のように返ってくる。

```
phy#0
Unnamed/non-netdev interface
  --省略--
Interface wlan0
  --省略--
```

wlan0 がデフォルトの WiFi インターフェイスを指している。また、1 行目が phy0 であれば内蔵、phy1 であれば外部のデバイスであると分かる。ここの数字が変わることがあるので注意。

1.2 仮想 WiFi インターフェイスの追加と ap モード

以下のコマンドで仮想 WiFi インターフェイスを追加する。

```
sudo iw phy phy0 interface add
ap0 type __ap
```

2 つ目の引数は先程の phy に続く数字と揃える。
次のコマンドで先程追加した ap0 に MAC アドレスを割り当てる。

```
sudo ip link set ap0 address
"ここが MAC アドレス"
```

iw dev をもう一度実行し、Interface ap0 が追加されていれば OK。ap0 は再起動すると消えてしまうので以下の設定を行う。

1.3 起動時に ap0 を自動設定する

上記のコマンドをラズパイが起動したタイミングで実行する。Web には udev の rules を設定する方法が紹介されているが、

電気通信大学 工学研究部
http://www.koken.club.uec.ac.jp
ueckoken@gmail.com

何度もカーネルがクラッシュしたので使わない。/etc/rc.local に書かれた内容は起動時に root 権限で実行されるので、

```
#!/bin/sh
/usr/sbin/iw phy phy0 interface
add ap0 type __ap
/usr/bin/ip link set ap0 address
"ここが MAC アドレス"
exit 0
```

を/etc/rc.local に書いて

```
sudo chmod 755 rc.local
```

で実行権限を付与する。権限は 700 でも良さそう (未確認)。

```
which iw
which ip
```

などして実行ファイルのフルパスを調べて書くこと。再起動して ap0 が確認できれば良い。

1.4 WiFi アクセスポイントの設定

nmcli を用いて発信する WiFi を設定する。

```
sudo nmcli connection add type wifi
con-name "ここがネットワークの識
別名"
ssid "ここが SSID"
802-11-wireless-security.psk
"ここがパスワード"
ifname ap0 mode ap
802-11-wireless.band bg
ipv4.method shared
ipv4.addresses "例)192.168..."
ipv6.method disabled
802-11-wireless-security.key-mgmt wpa-psk
802-11-wireless-security.proto rsn
802-11-wireless-security.pairwise ccmp
802-11-wireless-security.psk
"ここがパスワード"
autoconnect yes
```

改行を挟んだがこれで一つのコマンド。全てオプションなので行ごとに半角スペースで区切って入力すること。パスワードが.bash_history などに残るので注意!!
再起動してホットスポットが確認できればOK。ipv4.method shared としているのでホットスポットに接続したデバイスもラズパイが属するネットワークにアクセスできる。

3D プリンターで遊ぶ

22 ゆい

2025 年 3 月 23 日

1 はじめに

22 のゆいです。Bambu Lab の A1 という 3D プリンターを購入したのでその話を書きます。

2 3D プリンター A1 Combo を買った

Bambu Lab のブラックフライデーセールで、定価約 10 万円のマルチカラー対応 3D プリンターが 7 万円弱で手に入るというのに釣られて買ってしましました。AMS lite という自動フィラメント切り替え機を外付けすることで、最大 4 色のフィラメントを使い分けて印刷できます (多色刷りは切り替え時にゴミを大量に出すのであまり使っていない)。基本的に静かで早くて安定していて文句ないのですが、いくつか嫌な部分があります。

- 印刷開始時の共振測定 (？) がうるさすぎる
- 毎回の起動時にフィラメントのゴミが出る
- 印刷開始までが遅い (準備時間が長い)
- Z 軸のモーターがうるさい

3 G-code で遊ぶ

3.1 G-code の変更

G-code は 3D プリンター等の加工機を制御する言語です (たぶん)。Bambu のスライサー、Bambu Studio を使うとマシンの開始や終了のタイミングで実行される G-code を編集することで処理を任意に変更できます。これを使うと嫌な部分を自分好みに改造できます。ただ、G-code は加工機により各メーカーの独自拡張がされて機能が追加されているので情報が全くない部分があり、完全な解読は難しいです。公式ドキュメントがない部分は実際の動作を見たり予想したりの試行錯誤が必要です。共通の部分は情報が多いのでそれを参考に [1]、Bambu 独自拡張の部分は他のプリンターを解読している人の情報 [2] を参考に読んでいます。G-code 内の Placeholder はリストが公開されています [3]。

図 1　G-code の編集画面

3.2　印刷開始時の共振測定をしない

デフォルトでは、印刷開始時に毎回共振測定が走ります。毎回ブーブーうるさくて家が吹き飛びそうになるのでこれをやめます。プリンター開始 G-code の”mech mode fast check”で囲まれた部分をコメントアウトするか削除することで共振測定を無効にできます [4]。

```
;===== mech mode fast check start =====================
;M1002 gcode_claim_action : 3
;G1 X128 Y128 F20000
;G1 Z5 F1200
;M400 P200
;M970.3 Q1 A5 K0 O3
;M974 Q1 S2 P0
;M970.2 Q1 K1 W58 Z0.1
;M974 S2
;G1 X128 Y128 F20000
;G1 Z5 F1200
;M400 P200
;M970.3 Q0 A10 K0 O1
;M974 Q0 S2 P0
;M970.2 Q0 K1 W78 Z0.1
;M974 S2
;M975 S1
;G1 F30000
;G1 X0 Y5
;G28 X ; re-home XY
;G1 Z4 F1200

;===== mech mode fast check end ========================
```

3.3 毎回フィラメントを捨てるのをやめる

デフォルトでは、

- 印刷が終了するとフィラメントを切り、引き戻す
- 次の印刷の印刷が開始すると前の残ったフィラメントを捨てて新しいフィラメントを入れる

が毎回行われます。連続して同じ色で印刷したいときはフィラメントが無駄になるし開始時間が長くなる原因の 1 つです。これをやめるためにプリンター開始 G-code と終了 G-code を編集します [5]。

まず、プリンター終了 G-code の次の部分をコメントアウトします。これでフィラメントの引き戻しが行われなくなります。

```
; pull back filament to AMS
;M620 S255
;G1 X267 F15000
;T255
;G1 X-28.5 F18000
;G1 X-48.2 F3000
;G1 X-28.5 F18000
;G1 X-48.2 F3000
;M621 S255
```

次はプリンター開始 G-code の次の部分をコメントアウトします。これでフィラメントを捨てなくなります。

```
;M109 S{nozzle_temperature_range_high[initial_no_support_extruder]} H300
;G92 E0
;G1 E50 F200 ; lower extrusion speed to avoid clog
;M400
;M106 P1 S178
;G92 E0
;G1 E5 F200
```

ただしこれを使っていると、印刷開始時に入っているフィラメントと別のものを使うとき、なぜかフィラメントのパージが行われなくなりました。いろいろ検証していましたが部報の締め切りを突破してしまったので、今後の課題としておきます。

3.4 印刷開始までの時間を短縮する

- ノズル掃除の前半の、ノズルをベッドに連打する動作をしない
- ベッド予熱中にベッドレベリングを行う (なぜか実現できなかった)

等を調整すると印刷開始までの時間を短縮できました。

4 印刷して遊ぶ

実際に A1 で作ったものを紹介します。

4.1 MakerWorld のモデル

MakerWorld は Bambu Studio 内からもアクセスできる 3D モデルのプラットフォームです [6]。世界中の人が 3D モデルをアップロードしており、自由にダウンロードして印刷できます。さらに自分の使っている 3D プリンターで印刷できるプロファイルを配布しているものもあり、数回クリックするだけで最適化された設定を使って印刷できます。特にプリンターの拡張パーツは便利で、ノズルから出るごみ入れやケーブルを支えるパーツなどを印刷して使っています。

4.2 CPU ストラップ

Bambu のスライサーではモデルの印刷を好きなタイミングで一時停止することができます。これを使うと、閉じた造形物の中に別のものを埋め込むことができます。磁石を埋めこみ、磁石にくっつくものを印刷している人をみかけたので、自分は CPU を埋め込んでみました。

4.2.1 モデル

埋め込む CPU 本体のモデルを作成後、枠のモデルから CPU 本体をオフセットを付けて抜き出すことで作成しました。

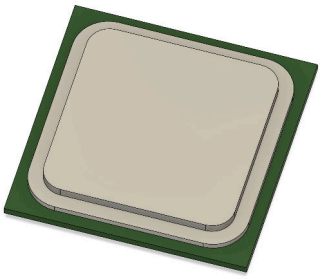


図 2　CPU のモデル

図 3　CPU ストラップのモデル 表

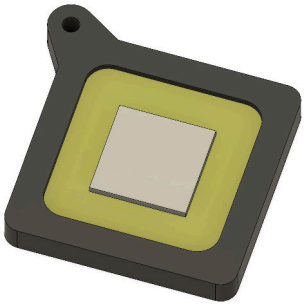


図 4　CPU ストラップのモデル 裏

4.2.2 印刷

一時停止後に印刷を再開することを考えると、なるべくノズルが引っかからない方向で印刷すべきです。今回は CPU のヒートスプレッダが下向きになる向きで印刷しました。印刷再開後は、CPU のパッドの上に下側部分を積み重ねていきました。スライサーはデフォルトから次のように変更しました。

項目	設定値	設定理由
ブリッジ方向	45	パッドの上の印刷角度を縦横部分で揃えるため
インフィルから造形	true	パッドの上にいきなり外枠の印刷はできず、内側から印刷したかったため
壁上の移動を迂回	true	CPU 裏のチップ部品を避けるため
厚いブリッジ	true	なんか強そうだったため

表 1　CPU ストラップのスライサー設定

初回は壁上の移動を迂回を false にしていたため、ノズルがチップ部品に衝突して破壊してしまいました。これを true にすることでノズルが壁を跨がなくなり、結果的にチップ部品がある中央部分を避けるようになります [7]。

4.2.3 完成

実際に完成したものがこちらです。わりといい感じに作れました。

図 5　CPU ストラップ 表

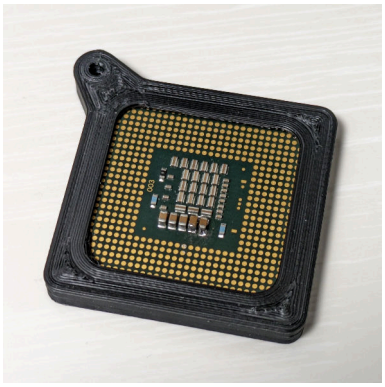


図 6　CPU ストラップ 裏

5　おわりに

最後まで読んでいただきありがとうございました。家に 3D プリンターがあると、好きな時に使いたいだけ使えるので便利です。特に、作りながら数値やスライサーの設定などを調整するのに毎回部室に行く必要がなくなりました。

今自分が使っている開始 G-code と Bambu の独自 G-code の早見表を最後に載せておきます。まだ十分 G-code を理解していないので大体は公式のものをコメントアウトしているだけなのと、検証途中で不正確な部分があります。

参考文献

[1] ＮＣプログラム基礎知識, ”G コード”, https://nc-program.s-projects.net/g-code.html, (参照 2025-03-18)

[2] Bambu Lab Community Forum, ”Bambu Lab X1 Specific G-Code”, https://forum.bambulab.com/t/bambu-lab-x1-specific-g-code/666, (参照 2025-03-18)

[3] Bambu Lab WiKi, ”Placeholder List”, https://wiki.bambulab.com/en/software/bambu-studio/placeholder-list, (参照 2025-03-18)

[4] reddit, ”How to turn off xy mech mode sweep”, https://www.reddit.com/r/BambuLab/comments/1acau76/how_to_turn_off_xy_mech_mode_sweep/, (参照 2025-03-18)

[5] Bambu Lab WiKi, ”Disable unloading and flushing to save filaments when printing the same material”, https://wiki.bambulab.com/en/ams/manual/ams-not-unloading-to-save-filament, (参照 2025-03-18)

[6] MakerWorld, https://makerworld.com/en, (参照 2025-03-18)

[7] Bambu Lab WiKi, ”Bambu Studio advanced quality settings”, https://wiki.bambulab.com/en/software/bambu-studio/parameter/quality-advance-settings#avoid-crossing-wall, (参照 2025-03-18)

付録 A　開始 G-code

全部載せようと思っていましたが思ったより行数が多く、部報のページ数がとんでもなく増えてしまうので Github へのリンクとします。https://github.com/Yuki-Yui/Bambu/blob/main/A1/StartGcode.gcode

付録 B　G-code 早見表

自分が使いながら書いているものです。まだ不正確な部分が多くあります。A1 の G-code をもとにしているので他のプリンターでは違う場合があります。

G-code	機能	説明
G0	高速移動	指定された位置に高速で移動する。
G1 X{x} Y{y} Z{z} E{e} F{f}	線形移動	指定された位置に移動する。 x, y, z は座標、a は押出量、b は移動速度。
G2 I{i} J{j} X{x} Y{y} F{f}	円弧移動	指定された円弧を移動する。I, J は中心座標、X, Y は終点座標、F は速度。
G4 P{p}	待機	p ミリ秒待機する。
G4 S{s}	待機	s 秒待機する。
G28	ホーミング	指定された軸をホームポジションに移動する。
G29.1 Z{z}	ABL オフセット設定	自動ベッドレベリングのオフセットを設定する。
G29.2 S{s}	ABL 有効化	自動ベッドレベリングを有効化する。s=0: 有効化, 1: 無効化
G29.4	不明	詳細な機能は不明。
G39.4	不明	詳細な機能は不明。
G90	絶対位置モード	絶対位置モードに切り替える。
G91	相対位置モード	相対位置モードに切り替える。
G92	原点設定	現在の位置を原点として設定する。
G380	プローブ？	G38 と同じ？ヒートベッドに当たるまで動かす？
M17	モータ有効化	ステッピングモータを有効化する。
M17 X{x} Y{y} Z{z}	電流制御	各軸のステッピングモータの電流を制御する。
M83	押出相対モード	押出を相対位置モードに切り替える。
M104 S{s}	ノズル温度設定	ノズル温度を s°C に設定する。
M106 P1 S{s}	ファン制御	部品冷却ファンを指定速度で回す。0≤s≤255
M106 P2 S{s}	ファン制御	補助ファン (big fan) を指定速度で回す。0≤s≤255
M106 P3 S{s}	ファン制御	チャンバーファンを指定速度で回す。0≤s≤255
M109 S{s}	ノズル温度待機	ノズルが s°C に達するまで待機する。
M109 S{s} H{h}	ノズル温度待機	ノズル温度が s〜h°C の範囲に達するまで待機する？
M140 S{s}	ベッド温度設定	ベッド温度を設定する。S[bed_temperature_initial_layer_single] で初層のベッド温度に設定する。
M190 S{s}	ベッド温度待機	ベッドが s°C に達するまで待機する。

M204 S{s}	加速度設定	プリンタの加速度を設定する。
M211 X0 Y0 Z0	ソフトエンドストップ無効化	ソフトエンドストップ (各軸の移動範囲制限) を無効化する。
M220 S{s}	フィードレート調整	フィードレートを s% に設定する。
M221 S{s}	フロー調整	フローを s% に設定する。
M392 S{s}	不明	詳細な機能は不明。
M400	動作完了待機	すべての動作が完了するまで待機する。
M412 S{s}	フィラメント検出	フィラメントの有無を検出する。
M500	設定保存	現在の設定を保存する。
M620 M	リマップ有効化	リマップ機能を有効化する。
M620.1 E F{f} T{t}	フィラメント速度設定	フィラメントの最大速度を設定する。
M620.3 W{w}	フィラメント絡み検出	フィラメントの絡みを検出する。
M621	フィラメント切り替え	フィラメントを切り替える。
M622 S{s}	不明	詳細な機能は不明。
M630 S{s} P{p}	モード設定	特定のモードを設定する。
M900 S{s} C{c}	高度な押出設定	高度な押出設定を行う。
M960 S5 P{p}	ロゴランプ	ロゴランプを制御する。s=1: オン, 0: オフ
M960 S{s} P{p}	レーザー制御	(キャリブレーション用？) レーザーを制御する。s=1: オン, 0: オフ
M970.3 Q{q} A{a} K{k} O{o}	不明	詳細な機能は不明。
M970.2 Q{q} K{k} W{w} Z{z}	不明	詳細な機能は不明。
M974 Q{q} S{s} P{p}	不明	詳細な機能は不明。
M975 S1	機械モード抑制	バイブレーションをオフにする？
M982.2 S{s}	ノイズ低減	ノイズ低減を有効化する。
M983 F{f} A{a} H{h}	押出補正	押出の動的補正を行う。
M984 A{a} E{e} S{s} F{f} H{h}	押出キャリブレーション	押出キャリブレーションを実行する。
M1002 gcode_claim_action : {x}	画面にメッセージ表示	x に指定されたメッセージを表示する。
M1006	プリンタ音制御	プリンタの音を制御する。
M1007 S{s}	質量推定有効化	質量推定を有効化する。
M2814 Z{z}	Z 軸調整	Z 軸の微調整を行う。
M9833.2	不明	詳細な機能は不明。
T[tool]	ツール変更	フィラメントの変更？フラッシュは手動でする必要がある？

自作アンプ (ノイズキャンセリングも自作したかった)

Donn

電子回路の授業でオペアンプを使った増幅回路というものをお勉強しました。反転増幅回路を使えばノイズキャンセリング自作できるのでは？？？ と思い手を出してみたものの、僕の怠惰な性格が祟ってただの増幅回路ということで妥協してしまいました。ところがどっこい、これが案外使い勝手がよかったんです。ということで、稚拙な作品ではありますがここに書かせていただきます。

1　理論

マイクで拾った音を電気信号に変えるわけですがこの電気信号めっっちゃ弱いです。我々が普段耳にするレベルの音の空気の振動自体そもそも小さいので。スピーカーでこの信号を我々にも聴こえる大きさの音でもって出力するには、もっと信号を強くする必要があります。そこで今回はオペアンプという素子を用いた非反転増幅回路で強い信号にしています。

オペアンプは電気信号を増幅させるよくわかんないけどすごいやつだと思ってください。非反転増幅回路は、入力信号がそのまま強くなって出力される回路だと思ってください。オペアンプにこんな風に抵抗とかつけると非反転増幅回路になるらしいです。

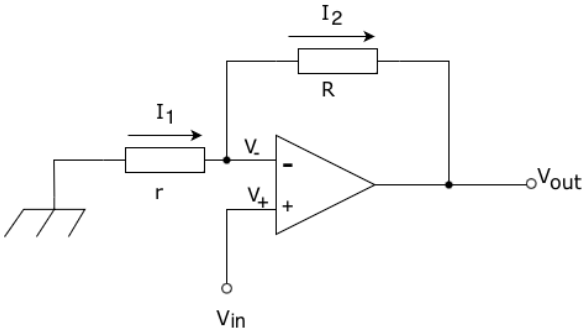


図 1　オペアンプによる非反転増幅回路

オペアンプの配線の仕方を変えると反転増幅回路という回路になって、入力信号がひっくり返って増幅されて出力されます。

本当は、この回路も実装して外からの音を打ち消すノイズキャンセリングを実装しようとしました。しかし、

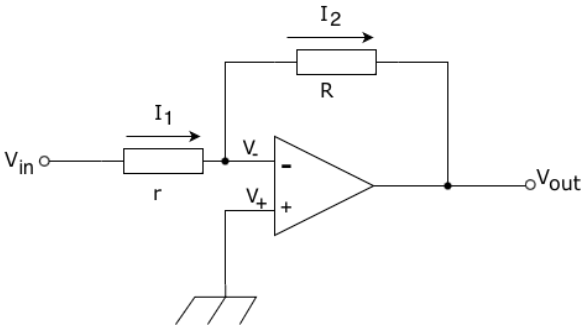


図 2　オペアンプによる反転増幅回路

時間もなかった上に、そもそもこの時点で想像以上に便利な道具であることに気づいてしまったので反転増幅回路を実装するのが面倒くさくなってきて実装していません。

2　実装

表 1　材料

NJM4850DD(オペアンプ)
JL-032C(エレクトレットコンデンサーマイク)
スピーカー
抵抗
コンデンサー

全体として図にするとだいたいこんなかんじです。正直見よう見まねです。特にマイクのことろはどこかに書かれてた推奨回路をほぼまんまパクりました。今回使ったコンデンサーマイクはエレクトレットコンデンサーマイクといって、電界効果トランジスタが内臓されているみたいだったので 6V の電源電圧でもって印可された負荷抵抗を繋げる必要があるとのことでした。トランジス

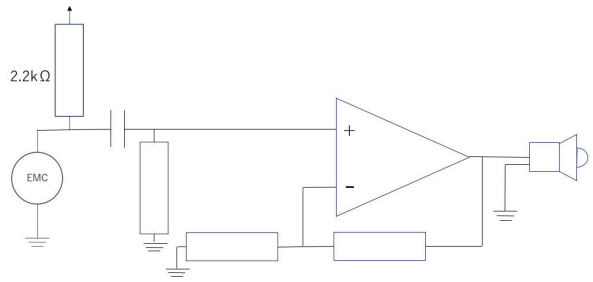


図 3　全体の回路の概略図

タを使うときの基本のようですね。

オペアンプの帰還とかの抵抗は接続させるスピーカーに合わせて適宜変える感じです。

なんでこんなところにコンデンサをはさむのかとか、ムズカしいことはよくわかりません。でも動いたのでOK です。

気が向いたらノイズキャンセリングにでも改造するつもりです。いつかきっと多分やります。

3　感想

スピーカーとかマイクを初めて電子回路で扱いましたが、定格電圧とか入力インピーダンスって結構大事なんだなあって思いました。これを気にしないでテキトーにやってたら思いっきり音がひずんだりして、わりと痛い目に遭いました。

参考文献

[1] マイクの種類と使い方 | CQ 出版社 オンライン・サポート・サイト CQ connect, https://cc.cqpub.co.jp/system/contents/2474/

[2] JL-032C データシート, https://www.marutsu.co.jp/contents/shop/marutsu/datasheet/JL-032C.pdf

[3] NJM4580D データシート, https://www.nisshinbo-microdevices.co.jp/en/pdf/datasheet/NJM4580_E.pdf

C++ で二次元の有限要素法を解いて可視化した話

23 Hogaraka

有限要素法をご存じだろうか。名前くらいは知っている人もいるだろう。これはもともと構造を解析するために使われた数値解析手法であり、今は連続体に広く応用されている。そして何より複雑な形状に対応できるのがこれの長所だ。今回は 2 次元ポアソン方程式を C++ で Qt と OpenGL のライブラリである SFML を用いて可視化することを記す。

1 性質 & 導出

数値解析とは手で解いて求める解析的な解を得るのが困難である場合に代わりに関数列 (定数値の配列を関数とみたもの) を求める手法。

その中でも有限要素法を重み付き残差法で解析する。さらにその中でもガラーキン法を用いてポアソン方程式を求める。

$$r(x) = \Delta u(x) + f(x)$$

$$\int_\Omega \omega(x) r(x) d\Omega = 0$$

上記の残差 r が空間 Ω 内でいつも 0 になる u がある場合、それが厳密解になる。しかし、それを見つけるのが困難な場合、代わりに r の平均が 0 である u を解とする。これを重み付き残差法という。

それを離散化していく。Γ_1 上で $\omega = 0$ とすると、

$$\int_\Omega \omega(x)(\Delta u + f) d\Omega - \int_{\Gamma_2} \omega(\frac{du}{dn} - q) d\Gamma = 0$$

となり、これをガウスの恒等式で変形する。そして、重み関数を Ψ とする。

そして、離散化されているマークに上付き文字 h を導入し、解 u^h を関数列で表わすと、

$$u^h = u_g + \Sigma u_i \phi_i$$

となる。ここで ϕ は重み関数で ϕ 関数を用いる。つまり、$\phi_i(x_j) = \delta_{ij}$ である。また Ψ にも δ 関数を用いる。

すると

$$AU = \mathbf{b}$$

となり ($u_i \neq 0$ のとき) ガウスジョルダン法で計算可能になる。

ここで

$$A_{ij} = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j d\Omega$$

$$\mathbf{b_i} = \int_\Omega \phi_i f d\Omega + \int_{\Gamma_2} \phi_i p d\Gamma - \int_\Omega \nabla\phi_i \cdot \nabla u_g d\Omega$$

$$A' = \begin{pmatrix} A^e_{kk} & A^e_{kl} & A^e_{km} \\ A^e_{jk} & A^e_{jl} & A^e_{jm} \\ A^e_{mk} & A^e_{ml} & A^e_{mm} \end{pmatrix}$$

$$A^e_{ij} = \Delta_e \left(b^e_i b^e_j + c^e_i c^e_j \right)$$

ここで k, l, m は構成する三角形の拡張点の座標に該当する配列番号

$$b_i = \frac{y_j - y_k}{2\Delta_e}$$

$$c_i = \frac{x_k - x_j}{2\Delta_e}$$

ここで i, j, k は A' における添え字で $j = (i+1) \mod 3$, $k = (i+2) \mod 3$

$$A = \Sigma A'$$

$$\mathbf{b_e} = \begin{pmatrix} \mathbf{b_k} \\ \mathbf{b_l} \\ \mathbf{b_m} \end{pmatrix} = \frac{\Delta_e}{12} \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} f_k \\ f_l \\ f_m \end{pmatrix}$$

尚、基本境界条件では $i = j$ の場合上書きし、直接 $A_{ij} = u_g$

$i \neq j$ で $A_{ij} = 0$ とする

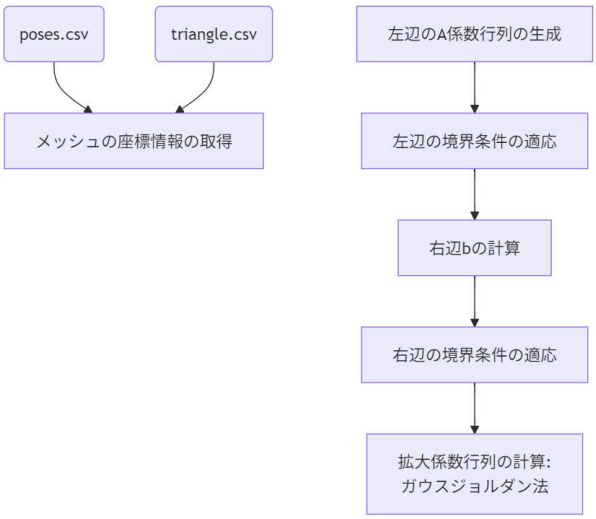
2 方法

メッシュ生成 poses.csv(点の座標ベクトル (pos) に番号を振り、pos を成分に持つ行列 poses として保存したもの) と triangle.csv(三角形の座標に対応する 3 点を poses の配列番号を用いて triangle に保存し、それの配列の triangle 行列) を保存また、時計回りに統一する。

要素係数行列を宣言し、要素の重み付けを Δ, b, c から計算する。これらを重ね合わせる。

左辺の拘束条件で自然拘束条件 (境界方向 1 階微分) なら重ね合わせ、基本境界条件 (0 階微分) なら上書きする。右辺も同様に内部の計算や境界条件を計算する。今回は直接法で解ける形なので、出来上がった係数行列に対し直接法で解を求める。今回はガウスジョルダン法を用いた。

3 方法 (有限要素法)



上記の手順で解を生成する。なお今回メッシュ分割には python を用いて numpy で計算した。

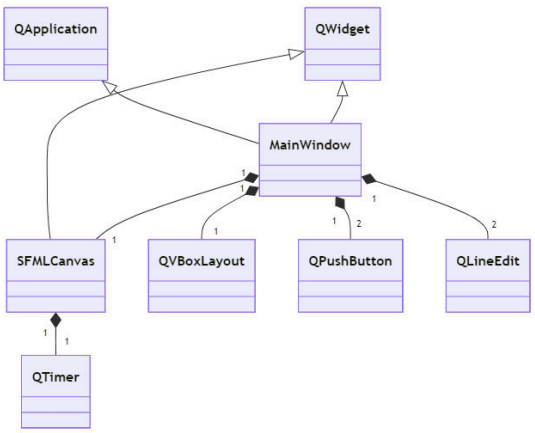
4 Qt, SFML について

Qt は C++ で動くクロスプラットホームで動作する GUI 開発ライブラリである。これによって日本語表記やテキストおよびボタン処理などが実装しやすくなる。さらに

SFML という OpenGL のラッパーライブラリを用いる。このライブラリもクロスプラットホームであり、またこれを用いることでグラフィック処理が簡便になる。また OpenGL よりもレンダリングコンテキストや入出力の点で OS 依存性が低いというメリットがある。

5　アーキテクチャ

Qt と SFML を同時に使うための継承関係。



特に MainWindow クラスに SFMLCanvas クラスをコンポジットする形で実装することで上記の機能を実現した。

6　実際の問題

領域

$$[0,1]^2$$

において境界内部で

$$\Delta \mathbf{u} = \sin(\pi x)\sin(\pi y)$$

境界境界条件は

$$u_g = 0 (x = \pm 1, y = \pm 1)$$

解析解は

$$u = -\frac{\sin(\pi x)\sin(\pi y)}{2\pi^2}$$

である。

7　出力結果

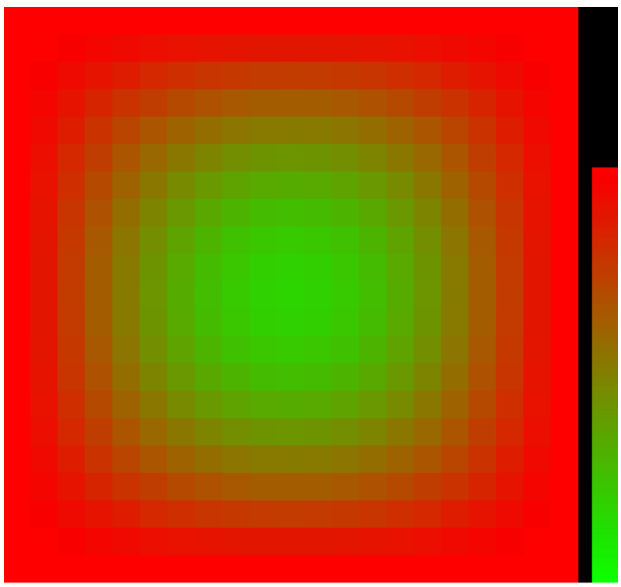


図 1　数値解

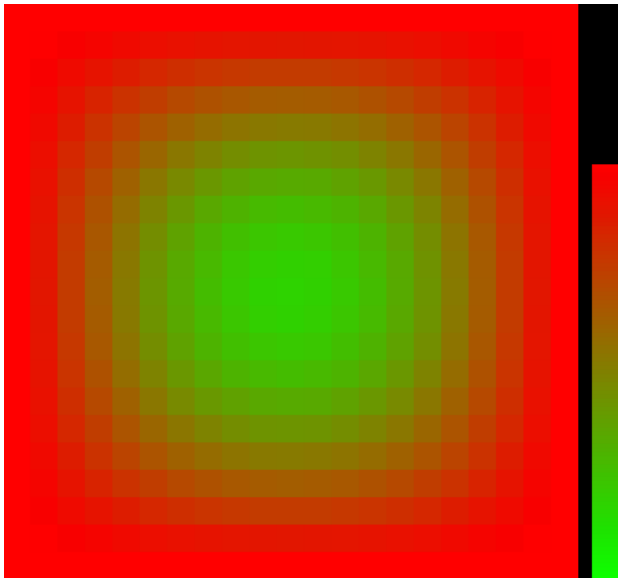


図 2　解析解

両者ともに [-0.06,0] で色を示し、誤差はおよそ 0.7% であった。

8　github

具体的なコードはこちらから

9　参考文献

有限要素法による流れのシミュレーション 日本計算工学会編 丸善出版 2015 年

レーザーカッターで革を切る

kat0h

2025 年 3 月 30 日

1　動機

革細工では革を綺麗に切断することが非常に重要。特に曲線を上手に切ることは難易度が高く、曲線のクオリティが完成品のクオリティに直結する。

レーザーカッターを使うことで、カッターでは難しい曲線や複雑な模様・ポンチ穴などを綺麗に切り抜くことを目指す。

2　実践した環境

電通大のピクトラボにある 30w の CO2 レーザーカッター (Etcher Laser Pro) を利用した。半導体レーザーを使ったレーザーカッターでも手順は変わらないと思う。

3　分かったこと

3.1　革はあらかじめ平らにしておく

最も重要な手順。特にタンニンなめしのような革は固いので歪んでいることがある。レーザーカッターは切断する高さに焦点を合わせるので、場所によって高さが違うと切れ方にムラができたり切り口が斜めになる可能性がある。

予め重しを引いて平らにしておいたり、ガムテープでステージに貼り付けると良い。

3.2　色の薄い革には向かない

レーザーカッターの性質上熱で対象を焼いて切断するので、断面が焦げる。よって、色の薄い革だと断面のみ黒く焦げてしまう。この焦げはヤスリで削れる程度だが断面の繊維が荒れるので、コバ磨きの工程に支障が出る。

3.3　表面が焦げる

煙によってレーザーが拡散されたり、ステージへ反射したりするなどした革の表面が焦げてしまう。切った後に染色するとしてもできるだけ表面へのダメージは避けたい。レーザーカッターで最後まで切るのではなく、銀面に薄皮一枚残すような形で切るとダメージが少ない。

3.4　型紙データの作成が面倒

今回は 3DCAD の fusion360 のスケッチ機能で型紙を起こした。スケッチを薄く押し出して、図面にした後 dxf 形式で出力したが、どう考えても遠まわりで面倒くさい。

おそらく 2DCAD で図面を作って dxf などの形式に出力するのが最適解だと思うので研究したい。(Linux で利用できる便利な 2DCAD を知りたい)

3.5　匂い問題

レザークラフトで使う革は大抵牛の革をなめしたものなので、燃やすとタンパク質特有の匂いがする。髪の毛をライターで燃やしたときと同じような匂いでお世辞にも良い匂いとはいえない。

大学ならまだしも、一般家庭では実践しにくい。

4　現時点で最も上手く切れた設定

4.1　条件

- 1.5mm タンニン鞣し革 (染色なし)
- レーザーカッターの設定
  - 速度: 2000mm/m
  - 出力: 60%
  - 繰り返し回数: 2 回

4.2　設定のねらい

レーザーのあたる面は煙の拡散などにより焦げやすいので、革の銀面 (ツルツルした面) を裏面にして切断した。(実際は裏面も焦げることがあった)

図 1は型紙を印刷して革に収まるか、どの部分を使うかを検討している写真。

図1　型紙を印刷して確認する

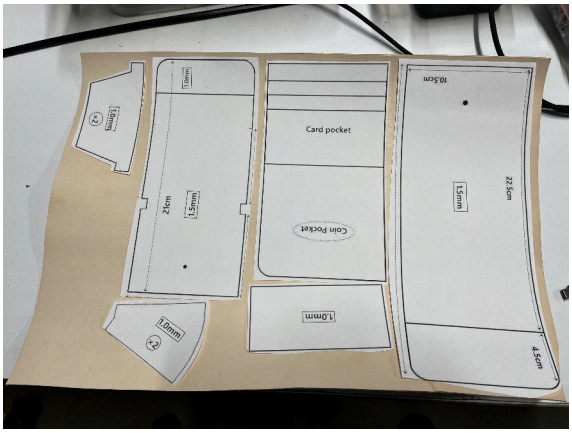


図 2は実際に革を切り出したもの。曲線も直線も十分に綺麗に切れている。真ん中の二つは完全に切り出せなかったため、カッターで分離した。左端は裏に焦げができてしまった。

図2　切り出した革

麻婆豆腐のおいしい作り方

kat0h

2025 年 3 月 30 日

おいしい麻婆豆腐を家庭で作りたかった記録。

図1　麻婆豆腐

1　麻婆豆腐とは

麻婆豆腐は、中国・四川省発祥の料理で、豆板醤を使った辛みのあるスープに豆腐を加えて煮立たせたものである。1862 年にに四川省の料理人に、陳劉氏によって考案されたもので、比較的歴史が浅い。[1]

日本で広まった四川式の麻婆豆腐のレシピは四川飯店の創業者である陳建民によって伝えられたものがベースになっている。さらに、日本では中国政府公認の「陳麻婆豆腐店」が 2000 年から営業しており、本場のレシピで提供している。

2　材料

麻婆豆腐の基本的な材料は以下の通りである。以降の章では特に重要な材料と、調理道具などについて説明する。

- 豆板醤
- 甜麺醤
- 豆鼓
- 花椒
- 粉唐辛子
- 生姜
- ニンニク
- 木綿豆腐
- 挽肉
- 葱
- 中華スープ
- 辣油
- 砂糖

3　豆板醤

麻婆豆腐の味の決め手は豆板醤である。私は郫県豆板醤 (図 2) を使っている。郫県豆板醤は中国の郫県 (ピーシェン) で作られている豆板醤で、長期間熟成されることが特徴である。長期間熟成されることで、通常の豆板醤よりも色が黒くなり旨味がでる。郫県豆板醤は通常の豆板醤よりも固く色が出にくいため、あらかじめ少量の水と油で炒めておくことにした。油で炒めることで赤い色がでて、香りも開く。通常の豆板醤よりも辛さのトゲが少ないが量を増やすと塩気も増えてしまうため、辛さの調節には粉唐辛子を増やすとよい。

4　豆腐

豆腐には木綿豆腐を使う。絹漉し豆腐の方が口溶けが良いが、崩れやすく調理の難易度が高い。1.5cm

図2　郫県豆板醤 [2]

程度の賽の目状に切り沸騰した湯で下茹でする。

5　豆鼓

豆鼓は黒大豆を発酵させた納豆に近い食材である。独特の香りを持つが、炒めることでとても良い旨味となる。スーパーなどでは豆鼓を刻み油などと炒めたものが豆鼓醤として販売されている。

図3　使用した豆鼓

6　辣油

辛さを足すだけでなく、仕上りの見た目を良くするために辣油を使用する。スーパーなどで販売されている小瓶の辣油を使っても良いが、すぐなくなる上、高いので自作することにした [3]。自作にはサラダ油・粉唐辛子・唐辛子・葱・花椒・生姜・八角を使用する (図 4)。

図4　辣油の材料

用意した材料を油とともに鍋で加熱した後、葱などから香りが十分に移ったら油を漉し、少量の水で練った粉唐辛子に油を熱いままかけて作る。

図5　香りを移している様子

7　調理道具

中華料理の厨房で使われている機材を揃えると、より楽に美味しく作れる。なにより楽しい。自炊として料理を作るのではなく、趣味として作ると長続きすると思う。

図6　完成した辣油

7.1　中華鍋

豆板醤や豆鼓など材料を炒めるために底の丸い鍋があるとやりやすい。一箇所に液体が集まるためである。

そもそも中華鍋がなぜあの形をしているのかを考察する。中華料理は大火力のバーナーで食材に一気に火を通し短時間で料理を仕上げるという特徴を持つ。油が多いのというのも油を熱を伝える媒体として利用して火を素早く通すためだと考えられる。よって、鍋は火の熱を即座に食材に伝えられるものでなくてはならない。また、高温に耐えられる必要もある。

図7　広東鍋を空焚きしている様子

中華鍋には大きく北京鍋と広東鍋がある。北京鍋は底が深い片手鍋で、広東鍋は比較的底が浅い両手鍋である。基本的にどちらの鍋を使うかは好みだが、北京鍋は重心が片方に寄っているため炒め物には持ちやすく便利な反面、煮物には不安定なため不向きである。

自宅では 30cm と 36cm の広東鍋を購入した。鉄板の厚さは 1.2mm で火にかけるとすぐ表面まで熱が回る。チタニウムなどの素材は熱伝導率が低すぎるため焦げ付きやすく、アルミニウムは柔らかく熱に弱いので薄い鍋には不向きである。

大抵の料理は 36cm の鍋でこなす。家庭で使うには少々大きいが、多少振っても材料が溢れないため麻婆豆腐の調理にはとても便利である。小回りが効くサイズとして 30cm、オールラウンダーなサイズとして 36cm と使い分けた。

山田工業所が作っている中華鍋が国内で購入できる中では一番上等なものである。浅草近くの河童橋道具街で複数の店が販売している。おすすめの店を下に示すので参考にしてほしい。

- 釜浅商店: 新しい建物で器具を選ぶ楽しさがある。個人客をターゲットにしているのか、店員さんがとても親切で分からないことなども教えてくれるはず。(`https://kama-asa.co.jp/`)
- 藤田道具: 500 円ほど相場より鍋が安いらしい。(`https://fujitadougu.com/`)
- ヤマヤ: 山田工業所の鍋が常時複数あるかも。(`http://yamaya-s.co.jp`)
- 銅源サイトウ: すこし分厚い 1.6mm の鍋の取り扱いあり。(`http://www.dougen-saito.co.jp/`)

自宅の 36cm 鍋は中尾アルミ製作所のものを使っている。既に中華鍋の製造を辞めてしまったようだが、河童橋に直営店が複数あるほかデッドストック品が安価に出回っている可能性がある。

7.2　おたま

中華鍋を購入するならセットで買いたいのが鉄もしくはステンレスで作られたお玉である。おたま一つで掬う・混ぜる・計量する・延すなど多彩な動作をすることができる。200ml サイズのおたまが

図8　河童橋道具街のシンボル

36cm なべに丁度良く、1cup を簡単に計量できるためお勧めする。

7.3　炸鍾

炸鍾は片手鍋に多数の穴を空けた形状をしている。スープや油から大きいものを掬い出すのに使われる。基本的に必需品ではないが鍋から豆腐を掬い上げたり、油淋鶏を作るときに肉に油を掛けたりするために非常に便利な道具である。

図9　炸鍾

7.4　ステンレスの平皿

切った材料を載せる、火を通した食材を一度上げる、落し蓋にするなど様々な用途で使用できる。ステンレスだと雑に扱えるほか、薄く収納できる。調理器具として買うと一枚 300 円 と少々値が張るが、100 円ショップでキャンプグッズとして販売されているものだと模様があるが安い。4 枚あれば大抵の料理には不自由しない。

図10　使用している平皿

8　調理工程

図11　材料の準備

8.1　挽肉を炒める

本来の麻婆豆腐は牛挽肉を利用するが高価で入手性が悪いため、豚挽肉を油を敷いた鍋で炒める。ここで重要なのが、挽肉をしっかり炒めることである。豚肉は特有の臭みを持つが、火を良く通すことで臭みを消すことができる。初め挽肉は白く濁った色の油を出すが、臭みが飛ぶころになると透明な油になる。

また、肉が焦げる寸前までしっかり火を通すことで食感を出す。酥と表現され、サクサクした食感にすることが重要である。

火を通したら、甜麺醤を絡ませ軽く炒める。甜麺醤は焦げやすいため、軽く炒めるだけで良い。できた味付きの肉は一度別皿に取っておく。

図12　肉を炒めている様子

8.2　スープを作る

鍋に油を多めに敷き、みじん切りにした生姜とにんにくを香りが出るまで炒める (焦がさないように注意)。十分に香りが出たら豆板醤と好みの量の粉唐辛子を加え、表面に無数の泡が湧き立つ程度炒めて油に豆板醤の赤みを移す。赤みを移せるタイミングは今しかないため、しっかりと炒めること。

豆鼓を加えたら鶏ガラスープを 350cc ほど入れる。スープは顆粒のものを使用しても良いし、味覇などのスープの素を使っても良い。表面に赤い油が出てきたところで、酒・醤油・砂糖を少量加える。

8.3　豆腐を茹でる

鍋に少量の塩を入れた湯を沸かせる。沸騰した湯に賽の目にした豆腐を入れ、再度沸騰したあたりで豆腐を引き上げる。

8.4　煮込む

豆腐を煮込む。このとき、おたまで豆腐を崩さないように注意する。基本的に混ぜる際はおたまの丸い面で豆腐を押すようにする。別の角度から混ぜるときは鍋を回して中の角度を変えるようにする。

図13　できたスープ

図14　下茹でした豆腐

1 分ほどたったらみじん切りにしておいた葱を混ぜる。このあたりからスープの流動性が低くなる。

8.5　固める

おたまで押したとき道ができる程度に固まったら、別にしておいた肉を全体に馴染ませる。水解き片栗粉を回し入れ、かき混ぜる。このとき強火にしていると片栗粉がすぐに固まってしまい、うまくと

図15　豆腐を煮込んで味を付ける

ろみがつかないため注意する。

全体に行き渡ったら辣油を表面に掛ける。この辣油は飾り油として機能する。最後に香りを出しつつ、片栗粉を固めるために強火で加熱する。片栗粉は加熱しないととろみを出さない。

図16　とろみを付けた様子

8.6　もりつけ

器に盛り付けたら、細かく挽いた花椒を掛ける。麻婆豆腐で重要な味わいの、麻辣の麻の部分となる。

図17　完成品

9　おわりに

陳健一のレシピを基に、調理するなかで気づいたことなどをまとめました。麻婆豆腐は簡単に調理できる上、味も安定しやすい便利な料理です。ぜひ、麻婆豆腐を作ってみて欲しいです。

10　参考文献

参考文献

[1] 麻婆豆腐発祥の店「陳麻婆豆腐」のストーリー, https://chenmapo.jp/history/

[2] 三明物産郫県豆瓣, https://sannmei.co.jp/product/%e9%83%ab%e7%9c%8c%e8%b1%86%e7%93%a3%e9%86%a4%e3%83%94%e3%83%bc%e3%82%b7%e3%82%a7%e3%83%b3%e3%83%88%e3%82%a6%e3%83%90%e3%83%b3%e3%82%b8%e3%83%a3%e3%83%b3/

[3] ホットペッパーグルメ自家製ラー油の作り方を「四川料理のスゴイ人」に教わってきた, https://www.hotpepper.jp/mesitsu/entry/noriki-washiya/19-00089

[4] とにかく売れたい中華屋, 【有料級】10 年かかって覚えた技を 10 分で教える人。, https://

www.youtube.com/watch?v=Zc5QlyB8s2c
[5] 料理王国, 陳建一シェフ「究極の麻婆豆腐」｜赤坂四川飯店, https://www.youtube.com/watch?v=VbZxN0sKuVg&t=602s

ロジック回路でタイマーを作りたかった & 麻雀の親システムおよび点棒システムを作った

23 わらび

2025/3/21

作りたかったシリーズ

1 概要

工研 LT(工学研究部による発表イベント)が 3 月 20 日に行われるが、前回のイベントにて、予定時刻の約 1.5 倍もの開催時間を経てしまい、発表時間を視覚的に示し、かつ時間を過ぎたときに終了を急かすため、水晶振動子を用いたタイマーを作る予定であった。しかし、MOSFET を用いていないか、ブレッドボードかジャンパ線のノイズが悪いため、仮組することができず、基板発注する時間もないため、急遽別の作品を製作し始めた。

ここで麻雀の「親」について説明する。親とは、点数移動を倍にする、牌を取り始める場所を決めるなどの役割をする人のことで、プレイヤー内から一人、順番に回すものである。その回し方についてはここでは省略する。

ロジック IC の中には、Dフリップフロップというものがある。これはクロック信号を受け取ると、D 端子の入力により出力の状態 Q を決めるものである。親をこの状態にあてはめて、入力したボタンに対応した LED が光り続けるものが今回の作品である。また、バイナリカウンターを用いて、同じ入力をした回数-1 の数字を 7 セグメント LED に表示する回路、すなわち点棒システムを作る予定だったが回路設計に不備があり、未だ完成していない。

2 使用した電子部品

2.1 ロジック IC

- 74hc74(D-ff)
- 74hc32(OR)
- 74hc08(AND)
- tc4030(XOR)
- tc4520(バイナリカウンター)
- tc4511(7 セグドライバー)
- 7 セグメント LED (カソードコモン)

2.2 他部品

- 7 セグメント LED (カソードコモン)
- コンデンサ、抵抗等

3 回路

図 1 に論理回路図を示す。尚、図の挿入の都合上 90 度傾けている。

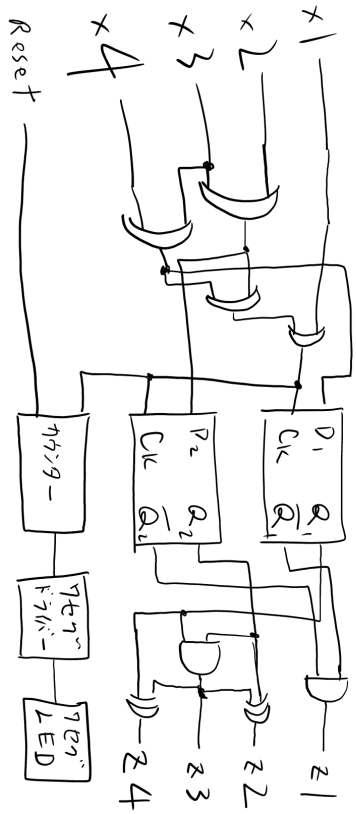


図 1　論理回路図

これをブレッドボードに配線した様子を図 2 に示す。

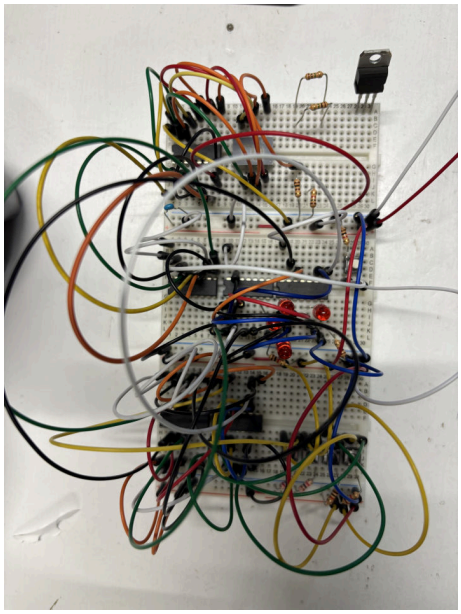


図 2　ブレッドボードに配線した時の様子

4 動作

5V の安定化電源を用いた結果、LED 込みで 0.05A での動作を確認した。尚、5V レギュレーターを用いた時は更に電力を要する。

5 考察・展望

カウンターの本来の目的は、同じ入力を繰り返すごとにカウントされ、他の入力でリセットされることを目標としていたが、うまく動かなかったためにリセットに対して入力を設けた。今後の展望としてはタイマーを製作することも含め、基板化を行う予定である。

シェルスクリプトで画像をつくってあそぶ

fluidflint

作成日　2025-03-19
最終更新日　2025-05-09

1　やったこと

　まず、以下のシェルスクリプトを作成した (紙面の都合上改行を適切にエスケープした上で行っている)。

```
#!/usr/bin/env ksh93
set -e
Date=$(date +'%0F-%H-%M-%S')

[ -e ./"$Date".ppm ] && { printf '%s\012' \
'ERROR: file already exists' >&2; exit 1 ;}

case $1$2 in
    *+++*) error=increment;;
    *--*) error=decrement;;
    *-*) error=minus;;
    */*) error=division;;
    *%*) error=remainder;;
    *\!*) error=negation;;
    *\**) error=multiplication;;
    *\&*) error=AND;;
    *^*) error=XOR;;
```

```
    *\|*) error=OR;;
    *'>>'*) error='bitwise shifts';;
    *'<<'*) error='bitwise shifts';;
    *) error= ;;
esac
[ -n "$error" ] && { printf '%s\012' \
"ERROR: arguments contain $error operator(s)" >&2; exit 2 ;}

    {
    printf '%s\012' "P3 $1 $2 255"
    od -tu1 -N$((3 * "$1" * "$2")) -w$((3 * "$1")) \
    -An -v /dev/urandom
}> ./"$Date".ppm
```

そして実行 (実行ファイルが./random_image.sh の場合。言うまでもないが実行権限をつける必要がある)

```
$ ./random_image.sh 1280 800
```

　こんなかんじになる (LaTeX でこの記事を作成した都合上 graphicsmagick で画像を PNG に変換している、と宣言しておく。これを述べずにおくと、「PPM 形式で入力したとは思えない。詐欺師気取りが」という指摘の生産ラインが急ピッチで建設されるであろう。その論争の生産を制限すること、いわば一人っ子政策が必要であったのда)。

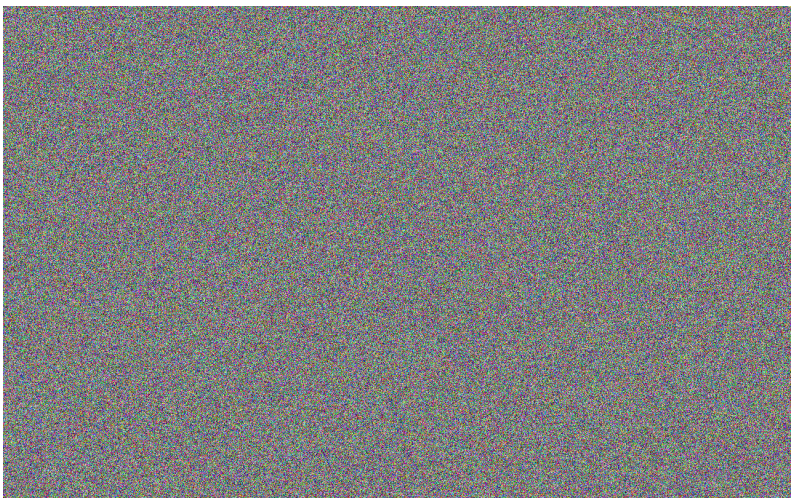


図 1　乱数は、画像になった

2　なにがおきたか

2.1　主要部

プレーンテキスト形式で画像を表現することのできる PPM 形式を用いた (プレーンテキスト形式のときは開始行の行頭に P3 と入れる。バイナリ形式では P6)。/dev/urandom(/dev/random でもよいが、その場合乱数が足りないときに読み込めなくなるんだったかなんだか) を用いて乱数を生成し、それを「od -tu1」で 1 バイトで表せる符号なし 10 進数に変換した。このとき、od によるアドレスの表示や、同じ値が連続するときの「*」表示を防ぐため、「-An」と「-v」をそれぞれ加えた。そして画像表示とテキストデータの関係をわかりやすくするため「-w$((3 * "$1"))」で行を区切った。最後{ ... }によりグループを作り、printf の出力と od の出力をまとめてファイルにリダイレクトしている。

2.2　非主要部

このシェルスクリプトで date の出力を一旦変数に入れているのは冒頭の set -e によるエラー処理をちゃんとやるためだ。

あとなんとなくだが、ファイルを上書きしないように [コマンドを用いてみた。

```
$ WhoHasPPMsThatMustNotBeOverwritten() { \
> exec kill -s TERM $$ ;\
> }; WhoHasPPMsThatMustNotBeOverwritten
```

3　何だお前のその shebang は!! 真の漢は Zsh を使うんだよ!!

あの..... 使うシェルに真の漢かどうかは関係ないだよ。朕は真の漢に非ず。そもそも、なんだろうな、zsh は元々 ksh と今では趣味シェルと言っても良い (かどうかは知らない)csh の中間を目指している感じだった。csh と ksh は結構違うだよ (csh は C 言語の文法とは大きく違うらしいだよ、私には使ったことがないが ShellSpec の開発者の ko1nksm が Qiita で言っていた [1])。だが POSIX sh のベースとなっているの

は ksh88(のサブセット?) だ。zsh はデフォルトで POSIX の規定や bash、ksh の挙動とは異なる。挙動を変更する方法はあるが、広くシステムシェルとして普及している bash、ksh と挙動が異なることを知らない初心者向け、とは必ずしも言えないと私は考える。まあ互換性よりも合理性を重視することでユーザフレンドリな部分もあるとは思うが (パス名展開 [2] とか)。つまり

- zsh の良さと、他のシェルや POSIX の規定との違いをわかっているユーザ。
- インタラクティブにしか使わず、シェルスクリプトを書くことが (これからも) ないライトユーザ。
- zsh がデフォルトのログインシェルである OS しか使わないライトユーザ。
- 他のシェルを全く使わないし、これからも使わないと魔王様に誓った存在。
- 重責を持ってシェルを利用する機会がなく、他人に迷惑を (掛けたくても) 掛け (られ) ない存在。
- どんな場合でも zsh が/bin/sh の環境しか使わな (くてよ) い時空の住人 (現世では macOS でも/bin/sh は bash だしデフォルトではまあ存在しないでしょう。仮に symlink を貼ったりとかして無理矢理/bin/sh を zsh に変更してもシステム内部のシェルスクリプトで/bin/sh が使われている限り同じ動作になることが (いわゆる POSIX モードでも) 保証されません。つまりこの場合は現状無視してください)。

以外の初心者にはあまりおすすめできない。

ここからは pdksh 系統のシェルと ksh93 系統のシェルが別物であることを念頭にお読みいただきたいが、ついに本題である。ksh93 にはバグの多い実装もあるので注意が必要かもしれぬ。しかし、事実上の正統 (?) 後継である ksh93u+m では多くのバグ... あれ、犬の話になっていた。バグではなくバクでもなくバグだ。失礼。ksh93u+m では多くのバグも修正されているそうだし、いいと思うが、インタラクティブにはあまり使われていないような気がする。え、なぜ私は ksh93 をスクリプトに使うのかって？ 何が優れているかって？ 仮想サブシェルである。サブシェル [3] を別プロセスで実行しないから速いらしい。しかも別プロセスである必要がある場合には透過的に「真の」サブシェルに変化する!!!らしい。一番大事なのは、これは特別な文法を必要としないしないしないしないしないしないしないしないしないしないしない。実装が大変だから前述のたくさんのバグもできたらしいが、その分他の POSIX シェルがこれから (元のライセンスのまま) 実装することはないと思っている。まあループとかしない

限り (そもそもループを避けたいが。特に外部コマンド) わからないレベルだと思う。また私が前使ったときに、インタラクティブに使うにはなんか UTF-8 対応が不十分な気がしたのでその用途に関しては現状おすすめしない。また、ksh93(bash や Zsh も) とかでは date を使わなくても日時を出せるらしいが移植性という観点 (はどうでもいいがファイル名に適した形式を使えるか調べるのも面倒くさいの) でやめた。

4　びこう

　ちゃんと shellcheck を行い不適切な部分を修正した。自分でも 1 回は実行した。え、それだけじゃテストにならないって？ ShellSpec も使っていないお前が部報にシェルスクリプトを載せるのは全ての工研者と ko1nksm(工研者ではないよたぶん) に対する反逆だって？
よいではないか～よいではないか～(工研者とは？ たぶんそんな言葉はない)

図 2　今回のスクリプトがテキスト形式で (リンクではなく) 入っている

参考文献

[1] ko1nksm、“シェルの歴史 総まとめ（種類と系統図）と POSIX の役割 ～ シェルスクリプトの現在・過去・未来”、2025-03-20 閲覧、https://qiita.com/ko1nksm/items/e7f43428352c0b4c78f9#c-シェル-csh-tcsh

[2] ko1nksm、“ls -l *.txt でファイルが見つからない時の bash (POSIX shell) と zsh の微妙で大きな違いと罠”、2025-03-20 閲覧、https://qiita.com/ko1nksm/items/97303aaf1c0bc14743db

[3] ko1nksm、“シェル必須の基礎知識「サブシェル」とは何か？ をちゃんと正しく解説する”、2025-03-20 閲覧、https://qiita.com/ko1nksm/items/19d300c4cb812b0fde1e

Discord の Bot で Wake ON Lan するお話

Dr.Latency

1 前回のあらすじ

立てたサーバから送った Magic Packet で起きたマシンに Team Viewer で接続できるようにした。

2 今回の概要

Discord から起動できるようにしてみる。

3 環境

前回と同じ。表記が不十分だった部分は修正した。

ホスト Windows10 Home 22H2
クライアント Windows11 Pro 24H2
サーバ UbuntuServer 22.04 (Proxmox LXC)

4 Discord Bot について

以下のサイトが分かりやすく、かつ今回の目的に合致しているので参考にした。
Discord 発 RaspberryPi 経由 WoL 線 (Discord を経由して外出先から PC の電源を ON にする)
`https://zenn.dev/amano_spica/articles/raspberrypi-wol-bot`

5 コード

最低限、Bot が動作しているか確認する簡単な応答と、事前に登録したマシンに Magic Packet を送信するもののみを用意した。Magic Packet の送信方法はコード内 13 行目にあるサイトを参考にした。場合によっては 17 行目の Addr に代入するアドレスを変更する必要があるかもしれない。本文中に書くには長すぎるので最後に示した。

6 MAC アドレス

7 環境整備

動作に必要な Python と discord.py の導入手順を以下に示す。先述したコードを/opt/M_Manager.py に保存したとして記す。2〜3 行目の pip の導入は環境によっては必要ないかもしれない。

```
$ sudo apt-get install libffi-dev libnacl-dev python3-dev
$ wget https://bootstrap.pypa.io/get-pip.py
$ python3 get-pip.py
$ python3 -m pip install -U discord.py
```

8 デーモン化

ここまででも十分動作するがどうせなのでデーモン化する。以下の.service を/etc/systemd/system/M_Manager.service に保存する。

/etc/systemd/system/M_Manager.service

```
[Unit]
Description = M_Manager daemon

[Service]
ExecStart = /opt/M_Manager.py
Restart = always
Type = simple

[Install]
WantedBy = multi-user.target
```

保存したら以下のコマンドで有効化する。

```
$ sudo systemctl enable M_Manager.service
$ sudo systemctl start M_Manager.service
```

9 結果

期待通りの動作をしたので、私のメイン機は Tailscale を使わずに起動・Team Viewer から操作をできるようになった。

10 余談

最近部屋のエアコンのコンセントにアースがあるのをようやく発見して PC 周りを頑張って全てアースを取りました。ついでに大学で拾った UPS にサーバ、ルーター、ハブを載せたのですがバックアップ運転の開始を認識させる仕組みをまだ作っていません。気が向いたら今回のものを RaspberryPi などに載せ替えて GPIO ピンにバックアップ運転開始の信号を受け取って何かできるようにしてもいいかもな、と思うだけ思っています。

参考文献

[1] Discord Developer Portal (https://discord.com/developers/applications)
[2] discord.py ドキュメント (https://discordpy.readthedocs.io/ja/latest/intro.html)
[3] Linux に pip をインストールする方法 (https://wellknowledge.org/linux-pip/)
[4] 超簡単 遠隔で PC の電源を入れる Wake on LAN 信号を Python で送信してみた (https://note.com/rcat999/n/n7dd94e68f231#ac707bbd-5679-4f4c-b543-f2b7640cf9f8)
[5] .service 自作して Python プログラムを systemd でデーモン化 (https://qiita.com/aKuad/items/c702b508375a5d9c476b)

/opt/M_Manager.py

```
import discord
import binascii
import socket

#your bot token
YourBotToken = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'

intents = discord.Intents.default()
intents.message_content = True

client = discord.Client(intents=intents)


#credit:(https://note.com/rcat999/n/n7dd94e68f231#ac707bbd-5679-4f4c-b543-f2b7640cf9f8)
def WOL(MAC):
        BinMac = binascii.unhexlify(MAC.replace(":",""))
        Header = b'\xFF\xFF\xFF\xFF\xFF\xFF'
        Addr = ("255.255.255.255",9)

        buf = Header
        for i in range(0,16):
                buf += BinMac
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        s.sendto(buf, Addr)


@client.event
async def on_ready():
    print(f'We have logged in as {client.user}')

@client.event
async def on_message(message):
    if message.author == client.user:
        return

    if message.content.startswith('$hello'):
        await message.channel.send('Hello!')

    if message.content.startswith('$Wake'):
        #your device's MAC address
        WOL('XX:XX:XX:XX:XX:XX')
        await message.channel.send('Now Booting...')

client.run(YourBotToken)
```

化石3dプリンターを再生する

packetloss40

2025年3月30日

1　初めに

大学で化石 3d プリンター「DaVinci Pro 1.0」を拾った。ブラウザから直接インターフェースを操作できるようにするために klipper 化に挑戦してみた。

2　プリンター破壊への道

まずはオリジナルのハードのスペックを調べた。一見、corexy 型に見えるが門型と同じ動作方式にねじ式の Z 軸、パイプをガイドとするベルト式の x,y 軸。フィラメントは小型のスプールをプリンターの中のケースに入れる方式でエンドストップスイッチは 3 軸とも光学式である。ノズルとベッドのヒーターは 12V 駆動で独自規格であった。WIFI に接続して専用のソフトから遠隔で操作することができる。マザーボードに新しいファームウェアを上書きしようとしたがうまく動作せず元のファームウェアもネットに転がってなかったのでボードが文鎮化した。

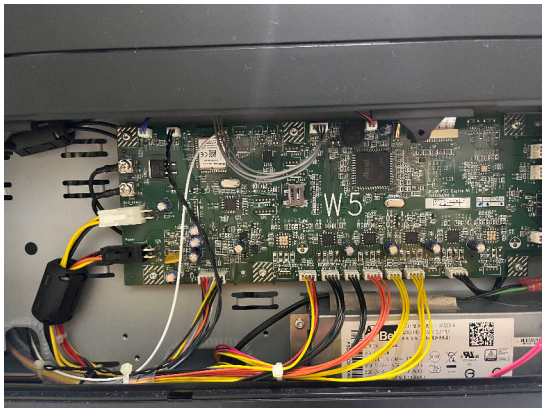


図 1: オリジナルのマザーボード

3　部品の調達

エンドストップを機械式スイッチにするためにオムロンスイッチを 3 つ、性能のいいノズルにするために E3D V6 REV というホットエンド、このホットエンドが 40W 駆動のため部室の Ender3pro からベッドヒーター、LED 電源を拝借した。またのちに x 軸のステップモーターが壊れていることが判明したためステップモーターも拝借した。壊れたマザーボードの代わりに Bigtree tech の skr pico を調達した。

図 2: アリエクから届いた部品

4　3D プリント部品の印刷

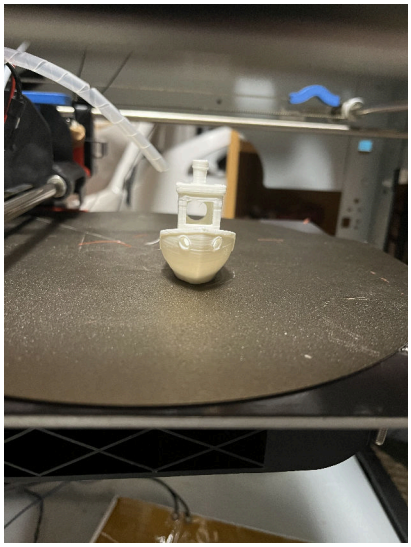
エンドストップを交換するに当たって、手間を減らすためにインターネットに転がっているアタッチメントを一部改変して印刷した。ベッドヒーターを交換するにあたって既存のフレームとベッドヒーターをつなぐアダプターを設計し、印刷した。ここで家にある安いほうのフィラメントが尽きたが何とかまともに使えるものが作れた。家に転がっているベッドレベリング用のアダプターもとりあえずつけれるように設計した。

5　klipper の設定

Raspberry Pi には MainsailOS をインストールした。MainsailOS は起動と同時に klipper をインストール済みで、起動しすぐにブラウザから操作することができる。skr pico も GitHub にファームウェアが与えられているのでそのままファームウェアを上書きすることで簡単に導入が完了した。コンフィグファイルは pico のリポジトリにあるファイルをもとに各ステッパーの回転距離やセンサー、ベッドメッシュなどを書き加えた。

6　試運転

無事に動作確認ができたので 3dbenchy を印刷した。ビルドプレートを家に持って帰り忘れたので家のデルタ用で代用。



7　終わりに

今までだれもやったことのない 3D プリンターの klipper 化やプリンターの 24V 化は調べることが多く、コンフィグファイルの各値の意味をよく知ることができるのでとても楽しかった。また改造の参考に klipper のコミュニティのスレッド (https://klipper.discourse.group/t/xyzprinting-davinci-1-0a-klipper-configs/556) も参考にした。これで自分の所有するプリンターは門型とデルタ型になったので次は corexy プリンターを 0 から組み立てて導入したい。ほかにはペットボトルフィラメントにも挑戦してみたい。

C 言語で弾幕シューティングゲームを作る

terraria

2025/3/18

回路設計の進捗は 0…。

1　はじめに

電気通信大学の後学期 (2 学期) には基礎プログラミングおよび演習 (以降、基礎プロとする) という講義がある。この基礎プロでは総合演習として C 言語を用いた動画の生成という課題があり、私はこの課題で弾幕を生成するプログラムを作成した。stbtruetype.h というヘッダファイルも用いて、フォントを導入するなど弾幕シューティングの再現を試みた。しかし、実際の弾幕シューティングは操作可能で当たり判定があるなど、動画だけでは再現できない部分が多い。そこで、ウィンドウ生成からキー入力、当たり判定の実装などを C 言語で扱い、弾幕シューティングゲームを作成した。

2　概要

今回、C 言語でウィンドウ生成やキー入力を行うため、SDL.h というヘッダファイルを用いて作成した。SDL は Simple Directmedia Layer の略で、ウィンドウ管理、2D レンダリング、入力管理、サウンドなどゲーム制作に必要な機能がまとめられているヘッダファイルであり、これを用いてゲーム制作を行った。また、弾幕や背景の画像を基礎プロで作成したプログラムを用いて生成した。

3　SDL.h について

今回、以下の SDL の機能を使用した。

1. ディスプレイ、ウィンドウ管理
2. 2D レンダリング
3. サーフェスの生成、描画

4. フォントの導入
5. イベント処理
6. キー操作
7. マウス操作

3.1 ディスプレイ、ウィンドウ、2D レンダリング

ウィンドウは SDL_Window という構造体を用いて、SDL_CreateWindow 関数で生成する。ここらへんは関数の定義どおり指定すればよいのであまり言うことはない。ウィンドウの大きさは width=1440,height=1080 の大きさで生成した。

3.2 サーフェスの生成、描画

サーフェスの生成、描画では、作成したウィンドウに画像を描画させることができ、SDL_Rect で指定した位置、領域内にその画像を表示する。SDL_Rect は x,y,width,height で構成される構造体で、図 1 のように基準の位置である x,y は左上が基準となる。座標の基準が左上であるのはこれだけでなく図 2 ウィンドウの座標系も左上が 0,0 となっている。

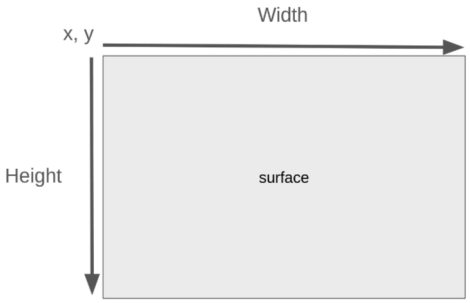


図 1　描画領域

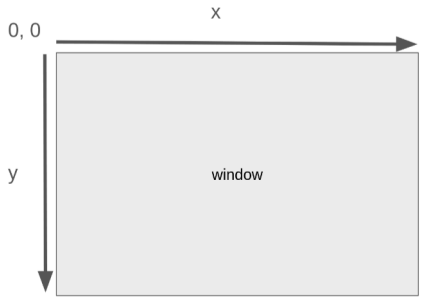


図 2　ウィンドウの座標系

これらの座標や、大きさは全て int 型であるので、桁落ちなどには十分注意する。今回の場合は、rect の他に、弾幕の座標、大きさを管理する配列 (float 型) を用意し、それを int 型でキャストして座標を指定している。

描画の関数はいくつかあり、以下のようになる。

1. SDL_RenderCopy
2. SDL_RenderCopyEx
3. SDL_RenderCopyExF
4. SDL_RenderCopyF

弾幕生成プログラムでは SDL_RenderCopy のみで構成されており、この関数の場合描画領域を回転させるといったことはできない。ただ、回転や反転等の関数は必要な変数が増え、回転の基準がわかりにくかったり、画像が汚くなるなどの弊害 (私の設定や環境が良くないのかもしれないが) があり使うことはあまりおすすめしない。特に丸い弾幕であれば回転の必要がないので、むやみに使用しないほうが良いと思われる。なるべく回転のいらない設計をすることを勧めます。

3.3　フォントの導入

フォントを導入するには、〜.ttf(truetypefont) ファイルを c のファイルが入っているディレクトリに入れ、TTF_OpenFont 関数で導入する。文字の大きさはこの段階で決めるが、文字をビットマップ形式にしたものを rect の領域内に描画するという方法であるから、rect の大きさを変更することでも文字の大きさを変更できる。ただし、この方法では小さい文字を大きくなるように拡大すると文字が汚くなるので、この方法を取るのであれば、TTF_OpenFont では文字の大きさを使用する最大の大きさより少し大きめにして生成しておくとよい。また、フォントサイズそのままの大きさで描画したいときは、TTF_SizeUTF8 関数で大きさを取得できる。
他にも SDL には多くの機能があり、詳しくは github などを参考にしてほしい。[1],[2]

4 画像生成

基礎プロで私が作ったものをほぼそのまま流用しているので画像生成の方法はあまり参考にしないほうが良い。この講義では、画像の出力形式が ppm で色の入力が rgb のみであり、rgba まで扱える png などと比べると少し扱いづらかったので、2 つの方法で余白の部分を削除した。1 つ目に、そもそも余白を作らない。これは画像の生成サイズをいじれば簡単に実現できるが、長方形画像に限られるやり方である。2 つ目に、余白部分を黒 (0,0,0) にして、プログラムでその部分を削除 (不透明度を 0) するという方法で以下のプログラムを用いた。

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <png.h>

void ppm_to_png_with_transparency(const char *input, const char *output) {
    FILE *fp = fopen(input, "rb");

    char format[3];
    int width, height, maxval;

    char c;
    while ((c = fgetc(fp)) == '#') {
        while (fgetc(fp) != '\n');
    }
    ungetc(c, fp);
    while (fgetc(fp) != '\n');
    size_t pixel_count = 3 * width * height;
    uint8_t *rgb = (uint8_t *)malloc(pixel_count);
    fclose(fp);
    FILE *png_fp = fopen(output, "wb");
    png_structp png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
    png_infop info_ptr = png_create_info_struct(png_ptr);

    setjmp(png_jmpbuf(png_ptr));

    png_init_io(png_ptr, png_fp);
    png_set_IHDR(png_ptr, info_ptr, width, height, 8, PNG_COLOR_TYPE_RGBA,
                 PNG_INTERLACE_NONE, PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

    png_bytep *rows = (png_bytep *)malloc(sizeof(png_bytep) * height);
    for (int y=0; y<height; y++) {
        rows[y]=(png_bytep)malloc(4 * width);
        for (int x=0; x<width; x++) {
            int idx=(y*width+x)*3;
            rows[y][x*4+0]=rgb[idx+0];  // R
            rows[y][x*4+1]=rgb[idx+1];  // G
            rows[y][x*4+2]=rgb[idx+2];  // B
            rows[y][x*4+3]=(rgb[idx] == 0 && rgb[idx+1] == 0 && rgb[idx+2] == 0) ? 0 : 255;
        }
    }

    png_set_rows(png_ptr, info_ptr, rows);
    png_write_png(png_ptr, info_ptr, PNG_TRANSFORM_IDENTITY, NULL);

    fclose(png_fp);
    png_destroy_write_struct(&png_ptr, &info_ptr);
    free(rgb);
    for (int i = 0; i < height; i++) free(rows[i]);
    free(rows);
}
int main() {
    ppm_to_png_with_transparency("Red.ppm", "Red.png");
    ppm_to_png_with_transparency("Orange.ppm", "Orange.png");
    ppm_to_png_with_transparency("Yellow.ppm", "Yellow.png");
    ppm_to_png_with_transparency("Green.ppm", "Green.png");
    ppm_to_png_with_transparency("Aqua.ppm", "Aqua.png");
    ppm_to_png_with_transparency("Blue.ppm", "Blue.png");
    ppm_to_png_with_transparency("Purple.ppm", "Purple.png");
    return 0;
}
```

生成した画像は図 3、4 のようになった。

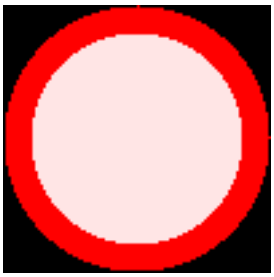


図 3　生成した弾幕 (ppm)

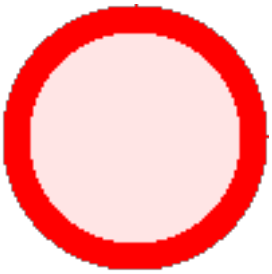


図 4　生成した弾幕 (png)

5　完成したもの

5.1　初期化

```
//title.c
#include <SDL.h>
#include <SDL_image.h>
#include <SDL_ttf.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

int main() {
  //Init
  SDL_Init(SDL_INIT_VIDEO); TTF_Init(); IMG_Init(IMG_INIT_PNG) & IMG_INIT_PNG;
  SDL_Window *window = SDL_CreateWindow("SDL ウィンドウ",
    SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, 1440, 1080, 0);

  SDL_Renderer* renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
  SDL_SetRenderDrawColor(renderer, 100, 100, 100, 128); SDL_RenderClear(renderer);
```

5.2　フォント、色の設定

```
  //fontsettings
  TTF_Font* titlefont = TTF_OpenFont("msmincho001.ttf", 200);
  TTF_Font* font = TTF_OpenFont("msmincho001.ttf", 60);
  TTF_Font* gamefont = TTF_OpenFont("msmincho001.ttf", 50);

  //colorsettings
  SDL_Color Red = {255, 0, 0, 255}; SDL_Color Black = {0, 0, 0, 255};
  SDL_Color Magenta = {255, 0, 255, 255}; SDL_Color Blue = {0, 0, 255, 255}; SDL_Color Gray = {200, 200, 200, 255};
```

5.3 文字設定

```
// Title
int TWidth, THeight; TTF_SizeUTF8(titlefont, "電", &TWidth, &THeight);
int iWidth, iHeight; TTF_SizeUTF8(titlefont, "通", &iWidth, &iHeight);
int tWidth, tHeight; TTF_SizeUTF8(titlefont, "工", &tWidth, &tHeight);
int lWidth, lHeight; TTF_SizeUTF8(titlefont, "研", &lWidth, &lHeight);
int eWidth, eHeight; TTF_SizeUTF8(titlefont, "抄", &eWidth, &eHeight);

//titlelength
int TitleWidth=TWidth+iWidth+tWidth+lWidth+eWidth;
int TitleHeight=(THeight+iHeight+tHeight+lHeight+eHeight)/2;

SDL_Surface* Ttitle = TTF_RenderUTF8_Blended(titlefont, "電", Red);
SDL_Texture* TTexture = SDL_CreateTextureFromSurface(renderer, Ttitle); SDL_FreeSurface(Ttitle);
SDL_Surface* ititle = TTF_RenderUTF8_Blended(titlefont, "通", Red);
SDL_Texture* iTexture = SDL_CreateTextureFromSurface(renderer, ititle); SDL_FreeSurface(ititle);
SDL_Surface* ttitle = TTF_RenderUTF8_Blended(titlefont, "工", Red);
SDL_Texture* tTexture = SDL_CreateTextureFromSurface(renderer, ttitle); SDL_FreeSurface(ttitle);
SDL_Surface* ltitle = TTF_RenderUTF8_Blended(titlefont, "研", Red);
SDL_Texture* lTexture = SDL_CreateTextureFromSurface(renderer, ltitle); SDL_FreeSurface(ltitle);
SDL_Surface* etitle = TTF_RenderUTF8_Blended(titlefont, "抄", Red);
SDL_Texture* eTexture = SDL_CreateTextureFromSurface(renderer, etitle); SDL_FreeSurface(etitle);

// Titlerect
SDL_Rect Tdest ={720-TitleWidth/2, 0, TWidth, THeight};
SDL_Rect idest ={720-TitleWidth/2+TWidth, 0, iWidth, iHeight};
SDL_Rect tdest ={720-TitleWidth/2+TWidth+iWidth, 0, tWidth, tHeight};
SDL_Rect ldest ={720+TitleWidth/2-lWidth-eWidth, 0, lWidth, lHeight};
SDL_Rect edest ={720+TitleWidth/2-eWidth, 0, eWidth, eHeight};

// Gametitlerect
int yTitle = 500;
SDL_Rect Tdest1 ={1150, yTitle, TWidth/2, THeight/2};
SDL_Rect idest1 ={1100, yTitle+THeight/2, iWidth/2, iHeight/2};
SDL_Rect tdest1 ={1150, yTitle+THeight/2+iHeight/2, tWidth/2, tHeight/2};
SDL_Rect ldest1 ={1100, yTitle+TitleHeight-eHeight/2-lHeight/2, lWidth/2, lHeight/2};
SDL_Rect edest1 ={1150, yTitle+TitleHeight-eHeight/2, eWidth/2, eHeight/2};

// NewGame
int newgameWidth, newgameHeight;
TTF_SizeUTF8(font, "NewGame", &newgameWidth, &newgameHeight);
SDL_Surface* newgame = TTF_RenderUTF8_Blended(font, "NewGame", Black);
SDL_Texture* newgameTexture = SDL_CreateTextureFromSurface(renderer, newgame); SDL_FreeSurface(newgame);
SDL_Rect newgamedest = { 720-newgameWidth/2, 500, newgameWidth, newgameHeight};
// NewGameCursor
SDL_Surface* newgameC = TTF_RenderUTF8_Blended(font, "NewGame", Magenta);
SDL_Texture* newgameTextureC = SDL_CreateTextureFromSurface(renderer, newgameC); SDL_FreeSurface(newgameC);
SDL_Rect newgamedestC = { 720-newgameWidth/2, 500, newgameWidth, newgameHeight};

// Exit
int exitWidth, exitHeight;
TTF_SizeUTF8(font, "Exit", &exitWidth, &exitHeight);
SDL_Surface* exit = TTF_RenderUTF8_Blended(font, "Exit", Black);
SDL_Texture* exitTexture = SDL_CreateTextureFromSurface(renderer, exit); SDL_FreeSurface(exit);
SDL_Rect exitdest = { 720-exitWidth/2, 600, exitWidth, exitHeight};
// ExitCursor
SDL_Surface* exitC = TTF_RenderUTF8_Blended(font, "Exit", Magenta);
SDL_Texture* exitTextureC = SDL_CreateTextureFromSurface(renderer, exitC); SDL_FreeSurface(exitC);
SDL_Rect exitdestC = { 720-exitWidth/2, 600, exitWidth, exitHeight};

// Life
int lifeWidth, lifeHeight;
TTF_SizeUTF8(gamefont, "Life", &lifeWidth, &lifeHeight);
SDL_Surface* life = TTF_RenderUTF8_Blended(font, "Life", Black);
SDL_Texture* lifeTexture = SDL_CreateTextureFromSurface(renderer, life); SDL_FreeSurface(life);

SDL_Rect lifedest = {1100, 100, lifeWidth, lifeHeight};

char str[2];
int numsize[10][2];
SDL_Rect Numdest[10][6];
SDL_Texture* NumTexture[10];
for (int i=0; i<10; i++) {
  snprintf(str, sizeof(str), "%d", i);
  TTF_SizeUTF8(gamefont, str, &numsize[i][0], &numsize[i][1]);
  SDL_Surface* num = TTF_RenderUTF8_Blended(gamefont, str, Black);
  NumTexture[i] = SDL_CreateTextureFromSurface(renderer, num);
  SDL_FreeSurface(num);
  Numdest[i][0].x=1200; Numdest[i][0].y=100; Numdest[i][0].w=numsize[i][0]; Numdest[i][0].h=numsize[i][1];
  for (int n=1; n<6; n++) {
    Numdest[i][n].x=1200+numsize[i][0]*(n-1); Numdest[i][n].y=200; Numdest[i][n].w=numsize[i][0]; Numdest[i][n].h=numsize[i][1];
  }
}

// Score
int scoreWidth, scoreHeight;
```

```
TTF_SizeUTF8(gamefont, "Score", &scoreWidth, &scoreHeight);
SDL_Surface* score = TTF_RenderUTF8_Blended(font, "Score", Black);
SDL_Texture* scoreTexture = SDL_CreateTextureFromSurface(renderer, score); SDL_FreeSurface(score);
SDL_Rect scoredest = {1100, 150, scoreWidth, scoreHeight};

// Time
int timeWidth, timeHeight;
TTF_SizeUTF8(gamefont, "Time", &timeWidth, &timeHeight);
SDL_Surface* timesurface = TTF_RenderUTF8_Blended(font, "Time", Black);
SDL_Texture* timeTexture = SDL_CreateTextureFromSurface(renderer, timesurface); SDL_FreeSurface(timesurface);
SDL_Rect timedest = {1100, 200, timeWidth, timeHeight};

// Game
int gameWidth, gameHeight;
TTF_SizeUTF8(titlefont, "Game", &gameWidth, &gameHeight);
SDL_Surface* game = TTF_RenderUTF8_Blended(titlefont, "Game", Gray);
SDL_Texture* gameTexture = SDL_CreateTextureFromSurface(renderer, game); SDL_FreeSurface(game);

// Over
int overWidth, overHeight;
TTF_SizeUTF8(titlefont, "Over", &overWidth, &overHeight);
SDL_Surface* over = TTF_RenderUTF8_Blended(titlefont, "Over", Gray);
SDL_Texture* overTexture = SDL_CreateTextureFromSurface(renderer, over); SDL_FreeSurface(over);

SDL_Rect gamedest = {500-gameWidth/2, 540-(gameHeight+overHeight)/2, gameWidth, gameHeight};
SDL_Rect overdest = {500-overWidth/2, 540-(overHeight-gameHeight)/2, overWidth, overHeight};
```

5.4 背景

```
// TitleBackground
SDL_Surface* image = IMG_Load("main.png");
SDL_Texture* imageTexture = SDL_CreateTextureFromSurface(renderer, image); SDL_FreeSurface(image);

// GameBackground
SDL_Surface* gameback = IMG_Load("gameback.png");
SDL_Texture* gamebackTexture = SDL_CreateTextureFromSurface(renderer, gameback); SDL_FreeSurface(gameback);

// GameFrame
int frpos[4]={60, 60, 940, 1020};
SDL_Surface* frame = IMG_Load("gamestruct.png");
SDL_Texture* frameTexture = SDL_CreateTextureFromSurface(renderer, frame); SDL_FreeSurface(frame);
SDL_Rect framedest={frpos[0]-10, frpos[1]-10, frpos[2]-frpos[0]+20, frpos[3]-frpos[1]+20};

// GameBlack
SDL_Surface* black = IMG_Load("gameblack.png");
SDL_Texture* blackTexture = SDL_CreateTextureFromSurface(renderer, black);
SDL_FreeSurface(black);
SDL_Rect blackdest={frpos[0], frpos[1], frpos[2]-frpos[0], frpos[3]-frpos[1]};

//Backgroundsize
SDL_Rect imagedest = { 0, 0, 1440, 1080 };
```

5.5 プレイヤー設定

```
// Gameplayer
SDL_Surface* player = IMG_Load("player.png");
SDL_Texture* playerTexture = SDL_CreateTextureFromSurface(renderer, player);
SDL_FreeSurface(player);

// Playersize
SDL_Rect playerdest;
```

5.6 弾幕用の配列設定

```
// Laser
SDL_Surface* laser = IMG_Load("laser.png");
SDL_Texture* laserTexture = SDL_CreateTextureFromSurface(renderer, laser);
SDL_FreeSurface(laser);

SDL_Surface* laser1 = IMG_Load("laser1.png");
SDL_Texture* laser1Texture = SDL_CreateTextureFromSurface(renderer, laser1);
SDL_FreeSurface(laser1);

// Danmaku
SDL_Texture* DTexture[7];
```

```
const char* filenames[7] = {
  "Red.png", "Orange.png", "Yellow.png", "Green.png", "Aqua.png", "Blue.png", "Purple.png"
};

for (int i=0; i<7; i++) {
  SDL_Surface* surface = IMG_Load(filenames[i]);
  DTexture[i] = SDL_CreateTextureFromSurface(renderer, surface);
  SDL_FreeSurface(surface);
}

// Danmakusize
int Dsize[2] = {25, 50}; float Dposition1[20][49][4]; float Dposition2[20][49][5];
float Dposition3[20][49][4]; float Dposition4[20][20][3]; float Dposition5[20][49];
// Danmakudest
SDL_Rect Ddest[20][49];
SDL_Rect Ldest[80];
```

5.7 ゲーム内の変数設定

```
SDL_Event e;
int runninggame = 0; int difficulty = 0; int gamenum; int starttime; int endtime; int endt=-200;
int zanki=3; int escapetime=-200; int dcheck; int keymode=1; int check[2]={0,0}; int timenum[5];
uint32_t t= 0; float scale; int x, y; Uint32 buttons; int keys[SDL_NUM_SCANCODES] = {0};  srand(time(NULL));
```

5.8 イベント処理

```
while (e.type != SDL_QUIT) {
  SDL_PollEvent(&e);
  SDL_PumpEvents(); // 最新のマウスの状態を確実に得る
  buttons = SDL_GetMouseState(&x, &y);
```

5.9 タイトル画面

```
  if (runninggame == 0) {
    difficulty = 0; zanki=3; escapetime=-200; check[0]=0; check[1]=0;
    playerdest.x=490; playerdest.y=920; playerdest.w=20; playerdest.h=40;

    Tdest.y=(int)(200+20*sin(t*.05));
    idest.y=(int)(200+20*sin(t*.05+2*M_PI/5));
    tdest.y=(int)(200+20*sin(t*.05+4*M_PI/5));
    ldest.y=(int)(200+20*sin(t*.05+6*M_PI/5));
    edest.y=(int)(200+20*sin(t*.05+8*M_PI/5));

    SDL_RenderClear(renderer);
    SDL_RenderCopy(renderer, imageTexture, NULL, &imagedest);
    SDL_RenderCopy(renderer, TTexture, NULL, &Tdest);
    SDL_RenderCopy(renderer, iTexture, NULL, &idest);
    SDL_RenderCopy(renderer, tTexture, NULL, &tdest);
    SDL_RenderCopy(renderer, lTexture, NULL, &ldest);
    SDL_RenderCopy(renderer, eTexture, NULL, &edest);
    SDL_RenderCopy(renderer, exitTexture, NULL, &exitdest);

    if ((x>=720-newgameWidth/2) && (x<=720+newgameWidth/2)) {
     if ((y>=500) && (y<=500+newgameHeight)) {
        SDL_RenderCopy(renderer, newgameTextureC, NULL, &newgamedest);
      if ((buttons & SDL_BUTTON_LMASK) != 0) {
        runninggame = 1; t=0;
      }
     }
    } else {
      SDL_RenderCopy(renderer, newgameTexture, NULL, &newgamedest);
    }

    if ((x>=720-exitWidth/2) && (x<=720+exitWidth/2) && (y>=600) && (y<=600+exitHeight)) {
      SDL_RenderCopy(renderer, exitTextureC, NULL, &exitdest);
      if ((buttons & SDL_BUTTON_LMASK) != 0) {
        SDL_DestroyTexture(imageTexture); TTF_CloseFont(titlefont); TTF_CloseFont(font); IMG_Quit();
        SDL_DestroyRenderer(renderer); SDL_DestroyWindow(window); SDL_Quit(); return 0;
      }
    } else {
      SDL_RenderCopy(renderer, exitTexture, NULL, &exitdest);
    }
  }
```

5.10 キー設定

```
//game
if (runninggame == 1) {
  if (e.type == SDL_KEYDOWN) {
    if (e.key.keysym.sym == SDLK_ESCAPE) {
      runninggame = 0;
    }
  }
  if (keymode==1) {
    if (e.type == SDL_KEYDOWN) {
      keys[e.key.keysym.scancode] = 1;
    }
    if (e.type == SDL_KEYUP) {
      keys[e.key.keysym.scancode] = 0;
    }
    if (keys[SDL_SCANCODE_UP] && playerdest.y > frpos[1]) {
      playerdest.y -= 5;
    }
    if (keys[SDL_SCANCODE_DOWN] && playerdest.y < frpos[3]-playerdest.h) {
      playerdest.y += 5;
    }
    if (keys[SDL_SCANCODE_LEFT] && playerdest.x > frpos[0]) {
      playerdest.x -= 5;
    }
    if (keys[SDL_SCANCODE_RIGHT] && playerdest.x < frpos[2]-playerdest.w) {
      playerdest.x += 5;
    }
  }
```

5.11 ゲーム画面

```
// GameBackground
SDL_RenderClear(renderer);
SDL_RenderCopy(renderer, gamebackTexture, NULL, &imagedest);
SDL_RenderCopy(renderer, frameTexture, NULL, &framedest);
SDL_RenderCopy(renderer, blackTexture, NULL, &blackdest);
SDL_RenderCopy(renderer, TTexture, NULL, &Tdest1);
SDL_RenderCopy(renderer, iTexture, NULL, &idest1);
SDL_RenderCopy(renderer, tTexture, NULL, &tdest1);
SDL_RenderCopy(renderer, lTexture, NULL, &ldest1);
SDL_RenderCopy(renderer, eTexture, NULL, &edest1);
SDL_RenderCopy(renderer, NumTexture[zanki], NULL, &Numdest[zanki][0]);
SDL_RenderCopy(renderer, timeTexture, NULL, &timedest);
timenum[0]=(t%100000-t%10000)/10000;
timenum[1]=(t%10000-t%1000)/1000;
timenum[2]=(t%1000-t%100)/100;
timenum[3]=(t%100-t%10)/10;
timenum[4]=t%10;
SDL_RenderCopy(renderer, NumTexture[timenum[4]], NULL, &Numdest[timenum[4]][5]);
SDL_RenderCopy(renderer, NumTexture[timenum[3]], NULL, &Numdest[timenum[3]][4]);
SDL_RenderCopy(renderer, NumTexture[timenum[2]], NULL, &Numdest[timenum[2]][3]);
SDL_RenderCopy(renderer, NumTexture[timenum[1]], NULL, &Numdest[timenum[1]][2]);
SDL_RenderCopy(renderer, NumTexture[timenum[0]], NULL, &Numdest[timenum[0]][1]);
SDL_RenderCopy(renderer, lifeTexture, NULL, &lifedest);
SDL_RenderCopy(renderer, scoreTexture, NULL, &scoredest);
if (zanki>=0) {
  SDL_RenderCopy(renderer, playerTexture, NULL, &playerdest);
} else {
  SDL_RenderCopy(renderer, gameTexture, NULL, &gamedest);
  SDL_RenderCopy(renderer, overTexture, NULL, &overdest);
}
if (t==1) {
  gamenum = rand()%4;
}
check[0]=check[1]; check[1]=0;
```

5.12 弾幕 1

```
//danamaku1
if (gamenum==0) {
  if (t==100) {
    framedest.y=frpos[1]-10; framedest.h=frpos[3]-frpos[1]+20;
    blackdest.y=frpos[1]; blackdest.h=frpos[3]-frpos[1]; keymode=1;
    for (int j=0; j<20; j++) {
      for (int i=0; i<49; i++) {
```

```
        Dposition1[j][i][2]=M_PI/48*i; Dposition1[j][i][0]=500; Dposition1[j][i][1]=100; Dposition1[j][i][3]=1;
        if ((i+j)%7==0) {
          Ddest[j][i].w = Dsize[1]; Ddest[j][i].h = Dsize[1];
        } else {
          Ddest[j][i].w = Dsize[0]; Ddest[j][i].h = Dsize[0];
        }
        Ddest[j][i].x = (int)Dposition1[j][i][0]-Ddest[j][i].w/2;
        Ddest[j][i].y = (int)Dposition1[j][i][1]-Ddest[j][i].h/2;
      }
    }
  }
  if ((t>=180) && zanki >= 0) {
    for (int j=0; j<20; j++) {
      if (t>=180+30*j) {
        for (int i=0; i<49; i++) {
          if (Dposition1[j][i][0]!=0) {
            if (Dposition1[j][i][1]>frpos[1] && (Dposition1[j][i][1]<frpos[3]-Ddest[j][i].h)) {
             if (Dposition1[j][i][0]>frpos[0] && (Dposition1[j][i][0]<frpos[2]-Ddest[j][i].w)) {
               Dposition1[j][i][0]+=5*cos(Dposition1[j][i][2]);
               Dposition1[j][i][1]+=5*sin(Dposition1[j][i][2]);
               Ddest[j][i].x=(int)Dposition1[j][i][0];
               Ddest[j][i].y=(int)Dposition1[j][i][1];
               SDL_RenderCopy(renderer, DTexture[(i+j)%7], NULL, &Ddest[j][i]);
             } else {
               Dposition1[j][i][0]=0; Dposition1[j][i][1]=0; Dposition1[j][i][3]=0;
               Ddest[j][i].x=(int)Dposition1[j][i][0]; Ddest[j][i].y=(int)Dposition1[j][i][1]; dcheck += 1;
             }
            }
          }
        }
      }
    }
  }
  if (dcheck == 980) {
    printf("%d\n", t); t=0; dcheck=0;
  }
}
```

5.13 弾幕 2

```
// danmaku2
if (gamenum==1) {
  if (t == 100) {
    framedest.y=frpos[1]-10; framedest.h=frpos[3]-frpos[1]+20;
    blackdest.y=frpos[1]; blackdest.h=frpos[3]-frpos[1];
    keymode=1;
    for (int j=0; j<20; j++) {
      for (int i=0; i<49; i++) {
        Dposition2[j][i][4]=3; Dposition2[j][i][2]=cos(M_PI/48*i); Dposition2[j][i][3]=sin(M_PI/48*i);
        Dposition2[j][i][0]=500; Dposition2[j][i][1]=100;
        Ddest[j][i].w = Ddest[j][i].h = Dsize[0];
        Ddest[j][i].x = (int)(Dposition1[j][i][0]-Ddest[j][i].w/2);
        Ddest[j][i].y = (int)(Dposition1[j][i][1]-Ddest[j][i].h/2);
      }
    }
  }
  for (int j=0; j<20; j++) {
    if ((t>=180+30*j) && zanki >= 0) {
      for (int i=0; i<49; i++) {
        if (Dposition2[j][i][0]!=0) {
          if ((Dposition2[j][i][0]<=frpos[0]) && (Dposition2[j][i][2]!=fabs(Dposition2[j][i][2]))) {
            Dposition2[j][i][4]-=1; Dposition2[j][i][2]=fabs(Dposition2[j][i][2]);
          }
          if ((Dposition2[j][i][0]>=frpos[2]-Ddest[j][i].w) && (Dposition2[j][i][2]!=-fabs(Dposition2[j][i][2]))) {
            Dposition2[j][i][4]-=1; Dposition2[j][i][2]=-fabs(Dposition2[j][i][2]);
          }
          if ((Dposition2[j][i][1]<=frpos[1]) && (Dposition2[j][i][3]!=fabs(Dposition2[j][i][3]))) {
            Dposition2[j][i][4]-=1; Dposition2[j][i][3]=fabs(Dposition2[j][i][3]);
          }
          if ((Dposition2[j][i][1]>=frpos[3]-Ddest[j][i].h) && (Dposition2[j][i][3]!=-fabs(Dposition2[j][i][3]))) {
            Dposition2[j][i][4]-=1; Dposition2[j][i][3]=-fabs(Dposition2[j][i][3]);
          }
          if (Dposition2[j][i][4]>0) {
            Dposition2[j][i][0]+=6*Dposition2[j][i][2]; Dposition2[j][i][1]+=6*Dposition2[j][i][3];
            Ddest[j][i].x=(int)Dposition2[j][i][0]; Ddest[j][i].y=(int)Dposition2[j][i][1];
            SDL_RenderCopy(renderer, DTexture[(i+j)%7], NULL, &Ddest[j][i]);
          } else {
            Dposition2[j][i][0]=0; Dposition2[j][i][1]=0; Dposition2[j][i][2]=0; Dposition2[j][i][3]=0;
            SDL_RenderCopy(renderer, DTexture[(i+j)%7], NULL, &Ddest[j][i]); dcheck+=1;
          }
        }
      }
    }
    if (dcheck == 980) {
      dcheck=0; printf("%d\n", t); t=0;
    }
```

```
  }
}
```

5.14 弾幕 3

```
// danmaku3
if (gamenum==2) {
  int field[4];
  if (t==10) {
    playerdest.y=frpos[3]-playerdest.h; keymode=0;
  }
  if (t<100 && t>10) {
    framedest.y+=(frpos[3]-frpos[1]-playerdest.h)/90; blackdest.y+=(frpos[3]-frpos[1]-playerdest.h)/90;
    framedest.h-=(frpos[3]-frpos[1]-playerdest.h)/90; blackdest.h-=(frpos[3]-frpos[1]-playerdest.h)/90;
  }
  if (t == 100) {
    playerdest.y=blackdest.y; blackdest.h=playerdest.h; framedest.h=playerdest.h+20;
    for (int i=0; i<20; i++) {
      for (int j=0; j<49; j++) {
        Ddest[i][j].w=Ddest[i][j].h=Dsize[0];
        Dposition3[i][j][0]=frpos[0]+(frpos[2]-frpos[0])/19*i; Dposition3[i][j][1]=frpos[1]+Ddest[i][j].h;
        Dposition3[i][j][2]=(int)(20*sin(M_PI/10*j)); Dposition3[i][j][3]=6; Ddest[i][j].y=Dposition3[i][j][1];
      }
    }
  }
  if (t>100 && t<=120) {
    framedest.y-=4; blackdest.y-=4; playerdest.y-=4;
  }
  if (t>120) {
    if (e.type == SDL_KEYDOWN) {
      keys[e.key.keysym.scancode] = 1; // キーが押されたら1
    }
    if (e.type == SDL_KEYUP) {
      keys[e.key.keysym.scancode] = 0; // キーが離されたら0
    }
    if (keys[SDL_SCANCODE_LEFT] && playerdest.x > frpos[0]) {
      playerdest.x -= 5;
    }
    if (keys[SDL_SCANCODE_RIGHT] && playerdest.x < frpos[2]-playerdest.w) {
      playerdest.x += 5;
    }
  }
  if ((t>=180) && zanki>=0) {
    for (int j=0; j<49; j++) {
      if (t>180+10*j) {
        for (int i=0; i<20; i++) {
          if ((Ddest[i][j].y<=frpos[3]-10) && (Dposition3[i][j][3]!=0)) {
            Ddest[i][j].x=Dposition3[i][j][0]+Dposition3[i][j][2]-Ddest[i][j].w/2;
            Ddest[i][j].y+=Dposition3[i][j][3];
            SDL_RenderCopy(renderer, DTexture[i%7], NULL, &Ddest[i][j]);
          } else if (Dposition3[i][j][3]!=0) {
            for (int k=0; k<4; k++) {
              Dposition3[i][j][k]=0;
            }
            Ddest[i][j].x=0; Ddest[i][j].y=0; dcheck+=1; printf("%d\n", dcheck);
          }
        }
      }
    }
  }
  if (dcheck == 980) {
    endt=t; dcheck=0; field[0]=blackdest.y; field[1]=framedest.y; field[2]=blackdest.h; field[3]=framedest.h;
  }
  if ((t-endt<60) && (t-endt>=0)) {
    framedest.y+=(frpos[1]-field[0])/60; framedest.h+=(frpos[3]-frpos[1]-field[2]+20)/60;
    blackdest.y+=(frpos[1]-field[0])/60; blackdest.h+=(frpos[3]-frpos[1]-field[2])/60;
  }
  if (t-endt==60) {
    framedest.y=frpos[1]-10; framedest.h=frpos[3]-frpos[1]+20; blackdest.y=frpos[1]; blackdest.h=frpos[3]-frpos[1];
  }
  if (t-endt==120) {
    keymode=1; printf("%d\n", t); endt=-200; t=0;
  }
}
```

5.15 弾幕 4

```
// danmaku4
if (gamenum==3) {
  if (t == 100) {
```

```
    framedest.y=frpos[1]-10; framedest.h=frpos[3]-frpos[1]+20; blackdest.y=frpos[1]; blackdest.h=frpos[3]-frpos[1];
    for (int i=0; i<20; i++) {
      Ldest[i].y=frpos[1]; Ldest[i].h=frpos[3]-frpos[1]; Ldest[i].w=0; Ldest[20+i].x=frpos[0];
      Ldest[20+i].w=frpos[2]-frpos[0]; Ldest[20+i].h=0; Ldest[40+i].y=frpos[1]; Ldest[40+i].h=frpos[3]-frpos[1];
      Ldest[40+i].w=0; Ldest[60+i].x=frpos[0]; Ldest[60+i].w=frpos[2]-frpos[0]; Ldest[60+i].h=0;
      for (int j=0; j<10; j++) {
        Dposition4[i][j][0]=frpos[0]+30+(int)(20.0*sin(M_PI*i/5.0));
        Dposition4[i][10+j][0]=frpos[2]-30-(int)(20.0*sin(M_PI*i/5.0));
        Dposition4[i][j][1]=frpos[1]+30;
        Dposition4[i][10+j][1]=frpos[1]+30;
        Dposition4[i][j][2]=(float)j/20.0*M_PI;
        Dposition4[i][10+j][2]=(float)(10.0+j)/20.0*M_PI;
        Ddest[i][j].w=Dsize[0];
        Ddest[i][j].h=Dsize[0];
        Ddest[i][j+10].w=Dsize[0];
        Ddest[i][j+10].h=Dsize[0];
      }
      // for (int j=0; j<20; j++) {
      //   printf("rad=%f", Dposition4[i][j][2]);
      // }
      // printf("\n");
    }
  }
  if ((t>=180) && zanki>=0) {
    for (int i=0; i<20; i++) {
      if ((180+i*40)==t) {
        Ldest[i].x=playerdest.x+60;
        Ldest[20+i].y=playerdest.y+60;
        Ldest[40+i].x=playerdest.x-60;
        Ldest[60+i].y=playerdest.y-60;
      }
      if ((180+i*40)<t && (200+i*40)>=t) {
        Ldest[i].w+=1;
        Ldest[20+i].h+=1;
        Ldest[40+i].w+=1;
        Ldest[60+i].h+=1;
        SDL_RenderCopy(renderer, laserTexture, NULL, &Ldest[i]);
        SDL_RenderCopy(renderer, laser1Texture, NULL, &Ldest[20+i]);
        SDL_RenderCopy(renderer, laserTexture, NULL, &Ldest[40+i]);
        SDL_RenderCopy(renderer, laser1Texture, NULL, &Ldest[60+i]);
      }
      if ((200+i*40)<t) {
        for (int j=0; j<20; j++) {
          if (Dposition4[i][j][0]!=0) {
            Ddest[i][j].x=(int)Dposition4[i][j][0];
            Ddest[i][j].y=(int)Dposition4[i][j][1];
            Dposition4[i][j][0]+=8*cos(Dposition4[i][j][2]);
            Dposition4[i][j][1]+=8*sin(Dposition4[i][j][2]);
            if (Ddest[i][j].x>frpos[0] && (Ddest[i][j].x<frpos[2]-Ddest[i][j].w)) {
              if (Ddest[i][j].y>frpos[1] && (Ddest[i][j].y<frpos[3]-Ddest[i][j].h)) {
                SDL_RenderCopy(renderer, DTexture[(10*i+j)%7], NULL, &Ddest[i][j]);
              } else {
                for (int k=0; k<3; k++) {
                  Dposition4[i][j][k]=0;
                }
                Ddest[i][j].x=0;
                Ddest[i][j].y=0;
                dcheck+=1;
              }
            }
          }
        }
      }
      if ((180+i*40+5)<t && (180+i*40+20)>=t) {
        for (int j=0; j<20; ++j) {
          if (abs((Ldest[i].x+Ldest[i].w/2)-(playerdest.x+playerdest.w/2))<(Ldest[i].w/2+playerdest.w/2)) {
            check[1]=1;
          }
          if (abs((Ldest[20+i].y+Ldest[20+i].h/2)-(playerdest.y+playerdest.h/2))<(Ldest[20+i].h/2+playerdest.h/2)) {
            check[1]=1;
          }
          if (abs((Ldest[40+i].x+Ldest[40+i].w/2)-(playerdest.x+playerdest.w/2))<(Ldest[40+i].w/2+playerdest.w/2)) {
            check[1]=1;
          }
          if (abs((Ldest[60+i].y+Ldest[60+i].h/2)-(playerdest.y+playerdest.h/2))<(Ldest[60+i].h/2+playerdest.h/2)) {
            check[1]=1;
          }

        }
      }
      if ((180+i*40+21)==t) {
        Ldest[i].x=0;
        Ldest[i].y=0;
        Ldest[i].w=0;
        Ldest[i].h=0;
        Ldest[20+i].x=0;
        Ldest[20+i].y=0;
        Ldest[20+i].w=0;
        Ldest[20+i].h=0;
        Ldest[40+i].y=frpos[1];
        Ldest[40+i].h=frpos[3]-frpos[1];
```

```
            Ldest[40+i].w=0;
            Ldest[60+i].x=frpos[0];
            Ldest[60+i].w=frpos[2]-frpos[0];
            Ldest[60+i].h=0;
            dcheck+=1;
          }
        }
        if (dcheck == 420) {
          printf("%d\n", t);
          t=0;
          dcheck=0;
        }
      }
    }
```

5.16 当たり判定

```
    // atari
    if (gamenum!=NAN) {
      for (int j=0; j<20; ++j) {
        for (int i=0; i<49; i++) {
          float l;
          l=pow((playerdest.x-10-Ddest[j][i].x), 2)+pow((playerdest.y+20-Ddest[j][i].y), 2);
          if (l<pow((3+Ddest[j][i].w/2),2)) {
            check[1]=1;
          }
        }
      }
      if ((check[0]==0) && (check[1]==1)) {
        zanki-=1;
        if (zanki==-1) {
          escapetime=t;
          printf("time=%d\n", escapetime);
        }
      }
    }
```

5.17 死亡設定

```
    // gameend
    if (t-escapetime == 180) {
      runninggame=0;
    }
  }
  SDL_RenderPresent(renderer);
  SDL_Delay(16); // 60FPS (1000ms / 60 ≈ 16ms)
  t++;
}
```

5.18 終了

```
  SDL_DestroyTexture(imageTexture);
  TTF_CloseFont(titlefont);
  TTF_CloseFont(font);
  IMG_Quit();
  SDL_DestroyRenderer(renderer);
  SDL_DestroyWindow(window);
  SDL_Quit();
  return 0;
}
```

6 各弾幕

6.1 弾幕 1

扇形に広がる弾幕。一部の弾は大玉担っており、当たり判定も大きい。

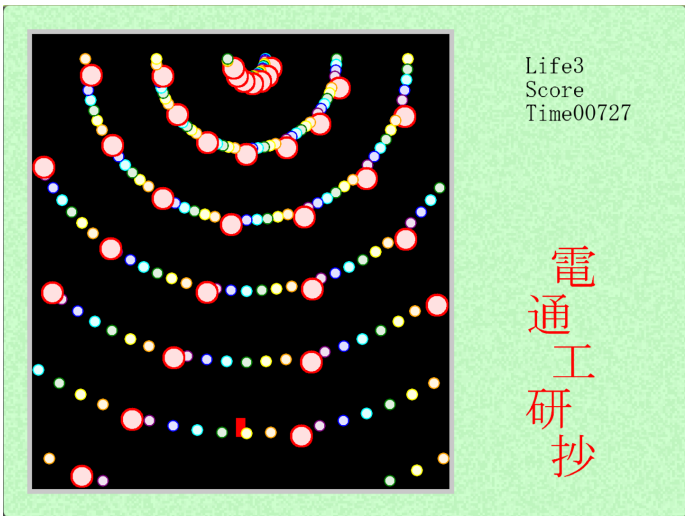


図 5　弾幕 1

6.2　弾幕 2

扇形に広がる弾幕で壁に当たると反射する (3 回まで)

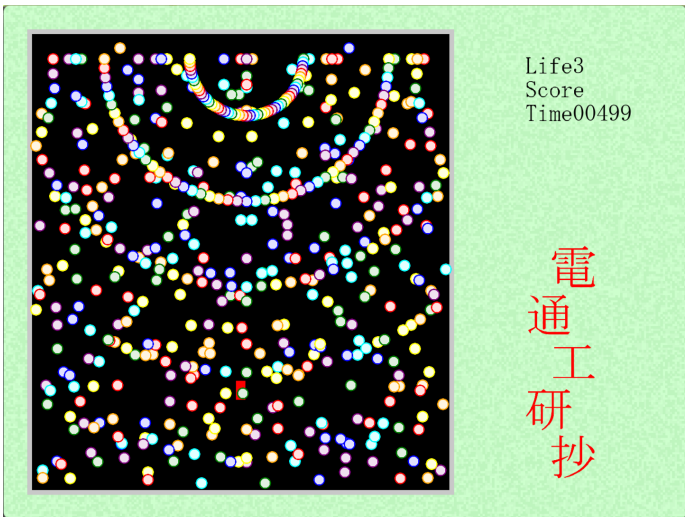


図 6　弾幕 2

6.3　弾幕 1

行動範囲が左右のみになり、弾幕は波上に上から下へ移動する。

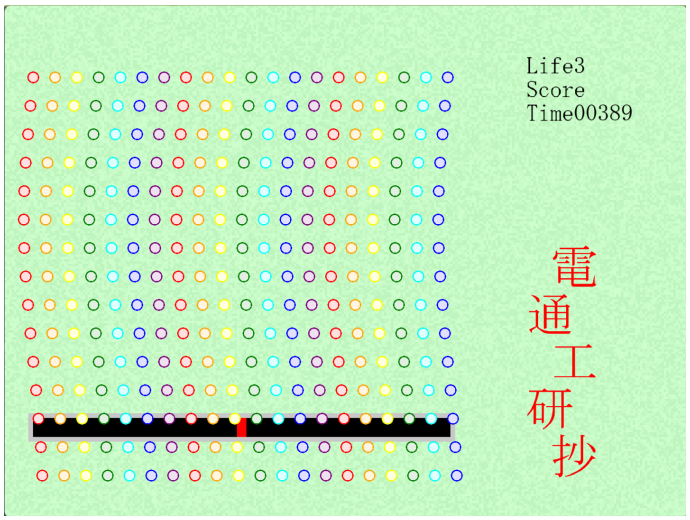


図 7　弾幕 3

6.4　弾幕 4

左上と右上から扇状に弾幕が広がり、自機の上下左右が一定時間ごとにレーザーで囲まれる。

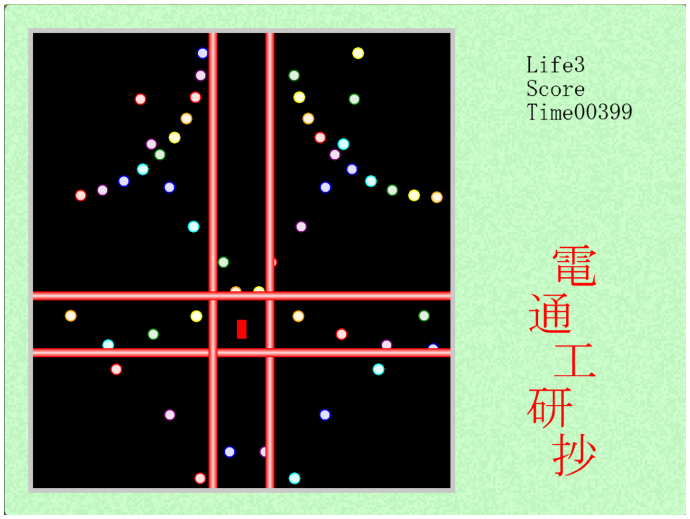


図 8　弾幕 4

7　感想

自機の絵が雑なのが残念。絵を書く暇がない上に画力が雑魚だから仕方無し。弾幕の生成自体はある程度うまく行ったが東方に比べると、弾幕の密度は薄い。あと単調な感じなのが気になる。東方は弾幕に変則さと美しさがあるからすごい。今回は SDL を活用することにも重点を置いたの

で、ゲームの機能としては面白いものもあったが、弾幕シューティングとしてはまだまだであるといった感じだ。スコアなんかも実装できればよかった。

8　おわりに

今回の弾幕シューティングゲーム作成は色々な部分が未完成ではあるが、最低限の機能が完成したので目的は達成できた。今後はまた回路設計をしていきたいと思っており、完成すればになるが、YMZ771 という音源 IC で楽器のようなものを作りたいと考えている。

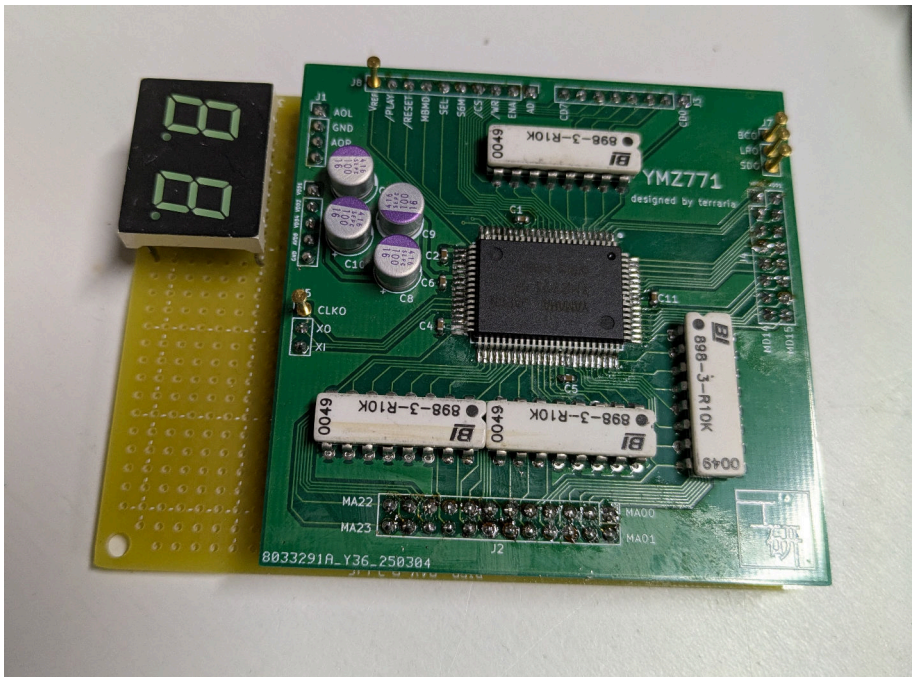


図 9　制作中のもの

参考文献

[1] SDL 2.0 API (機能別) https://maruhiro-ver0.github.io/sdl2manual-ja/ApiByCategory.html

[2] libsdl-org/SDL https://github.com/libsdl-org/SDL

NAS を自作してみた

とうふ

データ管理を楽にするために NAS(Network Attached Storage) を作成した。

1　経緯

私はスマートフォンでカスタム ROM を動かしている都合上頻繁にリセットをするためそのたびにデータの移動が生じていた。また、パソコンの環境も結構変えたりするため何かした常にデータ保管をする場所が必要だと考えた。ちょうどいいタイミングで、知り合いからラックマウント可能な PC ケースをいただいたのでそのケースを使って NAS を作ろうと考えた。

2　ケースへの組み込み

いただいたケースはこのような感じの明らかにラックマウントできそうな 2U ケースである。

このケースは Tyan の C612 マザーボードがもともと載っていた身体であるが、ほかのマザーボードも搭載可能である。

なので今回は昔使っていたパソコン中身をそのまま載せようと思う。CPU は Core i5-10400、メモリは 48GB、SSD は 500GB、HDD は 4TB である。SSD は OS 用に、HDD はデータ用にする。NAS にはオーバースペックだと思うが。HDD は前面にあるのマウンタを用いて接続している。中

では SATA ケーブルで接続している。また 10G SFP+NIC を搭載しているため光ケーブルで 10G 接続をしている。

3 構築

今回は TrueNAS Scale を用いて NAS を作成した。インストールについては省略する。インストールが完了したらデータ保管用の HDD の設定を行った。今回は、SMB と NFS の両方に対応させた。また、複数台 HDD があれば RAID の構築ができたが、今回は一台しかないため Stripe として設定した。いずれ RAID 構築も行ってみたい。

ここで設定をして、WindowsPC でマウントを行ったがファイルアクセスはできるものの書き込みなどができなかった。原因は権限不足であった。私のネットワーク環境は外部に公開していないため誰でもアクセス可能なように変更を行った。その後、WindowsPC でアクセスすることができた。

4 展望

今回 TrueNAS Scale で NAS の構築を行ったが、TrueNAS Apps といわれる機能があり多くのアプリケーションを手軽に導入することができる。今後はこの機能を使っていろいろなアプリケーションを導入していきたい。特に Emby Server 等で画像や映像の管理、Jellyfin で音楽の管理などを行いたい。

この NAS のマザーボードには IPMI 機能がなく外部からの再起動などができません。そのためどうしようかと考えていたところ、NanoKVM などの後付けで IPMI と同等の機能を提供できるソリューションが登場したのでそれらを使ってみたい。

またケースと一緒に RAID カードをいただいたのでそれを用いで RAID 構築を行ったりしていきたい。

国立大学法人 電気通信大学
工学研究部 部報 第 77 号

発行所　国立大学法人 電気通信大学工学研究部
〒182-8585 東京都調布市調布ヶ丘 1-5-1 サークル棟 2 階
Email ueckoken@gmail.com
URL http://www.koken.club.uec.ac.jp

発行責任者　加藤 滉太
編集者　堀合 風翔
新崎 佳苗
執筆　工学研究部 部員