日曜プログラマでもできる！

ラジコン&パンツァー

RADICON und PANZER

Vol. 1

1/35 M1A1 エイブラムス ラジコン化

Arduino＋Android＋赤外線で超信地旋回してみた

まえがき

「ガールズ＆パンツァー」のＴＶシリーズの放送開始から 4 年、劇場版の公開から 1 年が経ちました。特に劇場版を観て以来、戦車にドハマリしてしまい、今年の 2 月にはロシアのクビンカ戦車博物館に、7 月にはポーランドの軍事技術博物館に、8 月には富士総合火力演習にと、戦車漬けな日々を送っています。そんな中で、戦車を眺めるだけではなく、「自分で作りたい！ 動かしたい！」と思い、プラモデルを改造してラジコンを作ってみることにしました。初めての挑戦だったのでまだまだ不十分で未完成ですが、今までの取り組みをこの本にまとめています。どうぞお付き合いください。

今回制作したプログラムのソースコードは Github 上に公開しています。
https://github.com/misakahi/IRTank-Arduino
https://github.com/misakahi/IRTank-Android

2016 年 2 月、クビンカ戦車博物館にて

目次

第 I 部

システム概要

1　要求事項

ガルパンを見て「俺も戦車動かしたい！ というか作りたい！ 」と思ったのは自分だけではないはずです。超信地旋回や行進間射撃、したくないですか？

射撃を行うのはさすがに難しいので、

- 信地旋回・超信地旋回したいから、キャタピラは左右独立で動かす。
- 砲塔を **360** 度自由に回す。
- 戦車本体を一から作ることは無理だからプラモデルを改造する。
- 有線だとかっこ悪い。ラジコンにする。
- コントローラには今時らしく、スマートフォンを使う。

ことを目標に、プラモデルをラジコン化することにしました。

2　システムアーキテクチャ

ということで、考えたアーキテクチャが図 1 です。

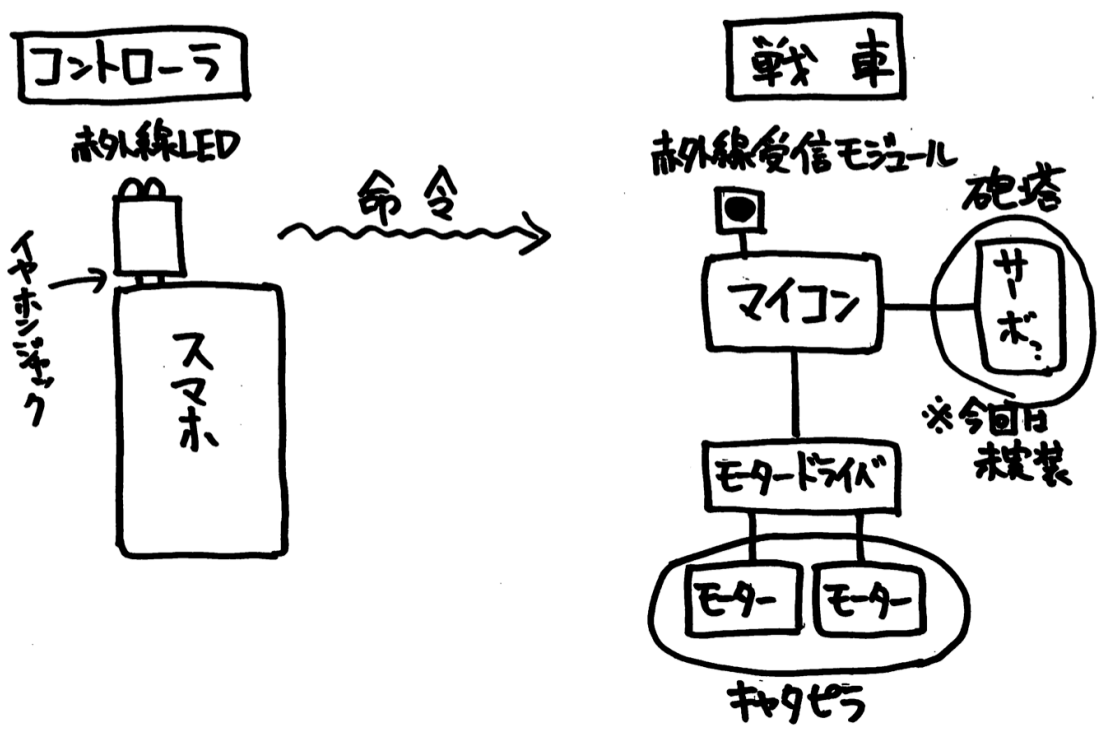


図 1　アーキテクチャ。

シンプルに、コントローラから命令を受け取って、左右のキャタピラを駆動するモーターと、砲塔を旋回させるサーボを動かす、という構成です。コントローラはスマートフォンのアプリで、画面にタッチすることで赤外線の信号をマイコンに向けて送信します。マイコンは信号を受け取ると、その内容に従ってモーターを動かします。マイコン及びモーターはプラモデルのシャシーに収めます。

3　制作方針

制作するにあたり、参考資料として「**1/35** 可動戦車模型の作り方」(新紀元社) というムック本を買いました。作例が３つ載っています。最初の作例は、ラジコンキットの中身を別のプラモデルに組み込むことで、簡単にラジコン化を実現する、というものです。動機として、戦車を動かしたいよりも、動く戦車を自分の手で作りたいほうが強かったので、これは採用せず、中身は自分で作ることにしました。ちなみに、肝心のタミヤの 1/35 ラジコン戦車は現在は手に入りにくいです。残りの２つの作例は小さい車体に機能を盛り込んだり、ギミックを満載にしたりと、初心者には真似出来ない感じでした。

そこで、構成要素にはなるべく既製品を使うことにしました。これにより、工作はプラモデル程度で済みますし、電子工作もマイコンとモジュールを繋げるだけで済みます。一部、部品を作るのに 3D プリンタを使用しました。3D プリンタを持っていなくても 3D プリントサービスを利用すれば大丈夫です。

また、使用した部品は安く手に入りやすいものを使うことにしました。今回使用した部品はほぼ全て Amazon、秋月電子、SWITCHSCIENCE のネット通販で購入できますし、値段も高々数百円程度です。

ぐだぐだ書きましたが、まとめると

- 中身は自分で作りたい！
- けど、サボれるところは極力サボる！
- 予算は一万円以内！
- とりあえず動くものを作る！

という方針で作っていきます。

図２　製作中の１コマ。秋山殿に搭乗していただいた。

第 II 部

構成要素

図 1 に示したアーキテクチャの構成要素の詳細について考えていきます。

4　戦車

図 3　今回使用したプラモデル。戦後第 3 世代主力戦車なのでガルパンには登場しない。

ベースとなる戦車のキットはなるべくシャシーの空間が広いものを選んだほうが楽です。なぜなら、シャシーの狭い空間の中に制御基板、電池、ギアボックスを詰め込む必要があるからです。そこで、今回はタミヤの **1/35** ミリタリーミニチュアシリーズ **No.156** アメリカ **M1A1** 戦車 ビッグガン・エイブラムス[*1] を使用することにしました (図 3)。Amazon では 2,000 円くらいで買えます。「ガルパンに出てこねぇじゃねぇか！ 」というツッコミは置いといて、このキットはシャシー内部が広く、また、過去にはラジコンキットも販売されていた[*2] ため、ラジコン化しやすくなっています。まさに初めての挑戦には最適といえるでしょう。

5　マイコン

マイコンも当然小さい方が望ましいです。そこで、今回は Arduino シリーズの中でも比較的小型な **Arduino Nano** を使用しました。Arduino Nano には USB 端子が付いているので、USB で PC と接続するだけで、買ってすぐに開発が始められます。値段も安く、中国製の互換品が Amazon で数百円で購入できます。

モータードライバ

モーターは左右２つのキャタピラを独立で動かす必要があります。そこで、１つの基板で２系統の出力があり、小型なデュアルモータードライバ **TB6612FNG** [*3] を使用しました。チップを実装済みの基板が SWITCHSCIENCE で 600 円程度で購入できます。

[*1] 定価は 3,240 円 http://www.tamiya.com/japan/products/35156m1a1_abrams/index.htm
[*2] 現在では絶版です。http://www.tamiya.com/japan/products/48201abrams/abrams.htm
[*3] https://www.switch-science.com/catalog/236/

図 4　マイコン Arduino Nano とモータードライバ TB6612FNG。

6　通信方法

スマホと Arduino 間で通信する方法としてまず思いつくのは、Wi-Fi か BlueTooth でしょう。Wi-Fi の場合、ESP-WROOM-02 等、安価で手に入り、ネットでも情報が多く手に入るモジュールがあります。BlueTooth の場合も、HC-05[*4]等の BlueTooth-シリアル変換モジュールを使えば、簡単に実現できます。

しかし、ただ左右のモーターを動かす命令を送るだけのためにこれらの方法を用いるのは仰々しいように思えます。そこで、今回はイヤホンジャックからの音声信号を赤外線として発信することで通信することにしました。イヤホンジャックは PC、スマホに関わらず装備されている[*5]ので、将来的に色んなデバイスから操作することも考えられます。

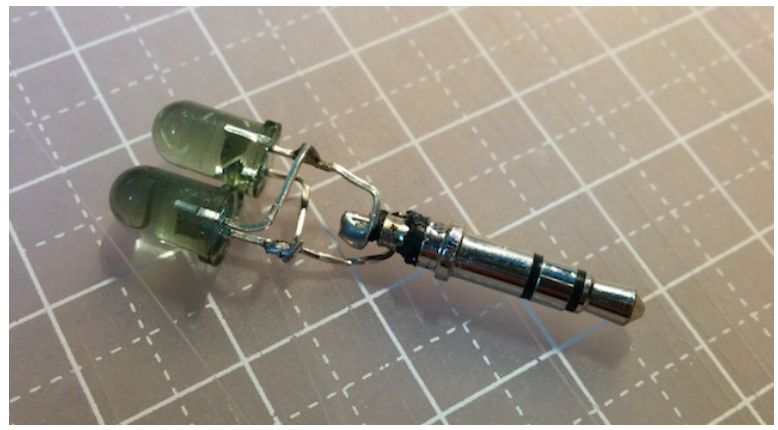


図 5　イヤホンジャックに赤外線 LED を挿すことで赤外線の信号を送ることができる。

7　ギアボックス

ギアボックスにはタミヤのツインモーターギアボックス[*6] を使用しました (図 6)。寸法がちょうどよく、シャシーに固定するだけでそのまま使えそうでした。難点は少し大きいことです。

他にも、省スペース化のため、タミヤの「ミニモーター低速ギヤボックス」[*7] を２つ使うことを考えましたが、シャシーとの高さの調整と、２つのギアボックス間の位置の調整が必要で、手間がかかりそうなため、今回は採用し

[*4] HC-05 は数百円で手に入りますが、技適マークが付いていません・・・。
[*5] iPhone7？ 知らない子ですね・・・。
[*6] http://www.tamiya.com/japan/kousaku/k_products/70097_2motor.htm
[*7] http://www.tamiya.com/japan/products/70189mini_lowspeed/

図 6　ツインモーターギアボックス。

ませんでした。

8　電源

電源はモーター用とマイコン用の２系統用意します。

モーター用電源には乾電池２本を直列に使いました。モーター用電源はモータードライバに接続します。モータードライバのデーターシート[*8]には、最小 2.5V、標準 5V、最大 13.5V が許容動作範囲とありますので、アルカリ乾電池 1.5V を２本直列にすれば動作すると思われます。実際には、ニッケル水素電池 1.2V[*9]を２本直列にしても動作しました。

マイコン用の電源は、Arduino Nano へ 5V を供給するために使います。今回は、秋月電子の昇圧型 DC-DC コンバータ[*10]を使用しました (図 7)。0.9V〜5V の範囲の入力で 5V 出力します。実際にニッケル水素電池 1 本の入力に対し、出力に Arduino Nano、モータードライバ、赤外線受信モジュールを接続してみたところ、安定して動作しました。

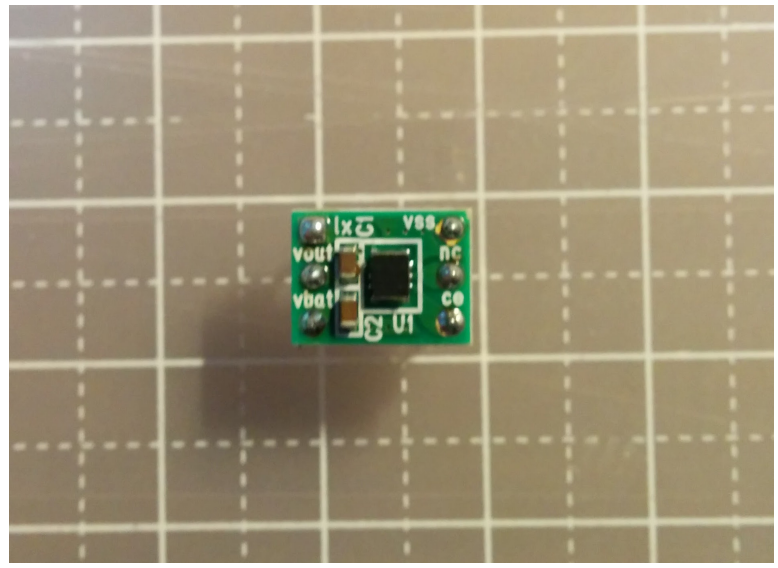


図 7　昇圧型 DC-DC コンバータ。

[*8] http://doc.switch-science.com/datasheets/TB6612FNG_datasheet_ja_20141001.pdf
[*9] いわゆるエネループ。
[*10] http://akizukidenshi.com/catalog/g/gM-08619/

第 III 部

赤外線による Android と Arduino の通信

ここでは、Android スマートフォンから Arduino へ赤外線を使って通信する方法について詳しく説明します。最近のスマートフォンには赤外線通信機能が付いてないのが普通です。しかし、イヤホンジャックの音声信号で赤外線 LED を光らせることで、情報を送ることができます。

9　赤外線リモコンの通信フォーマット

赤外線リモコンでは赤外線の点灯時間・消灯時間の組み合わせで 0 か 1 を判定する PPM(Pulse Position Modulation) 信号という方式が用いられます。通信フォーマットにはいくつかありますが、ここでは、NEC フォーマットを例にとって説明します。

NEC フォーマット

NEC フォーマットの場合、信号は、「リーダーコード」、「データ」、「ストップビット」、「フレームスペース」で構成されます（図 8)。1 フレームの合計時間は 108ms (固定) です。

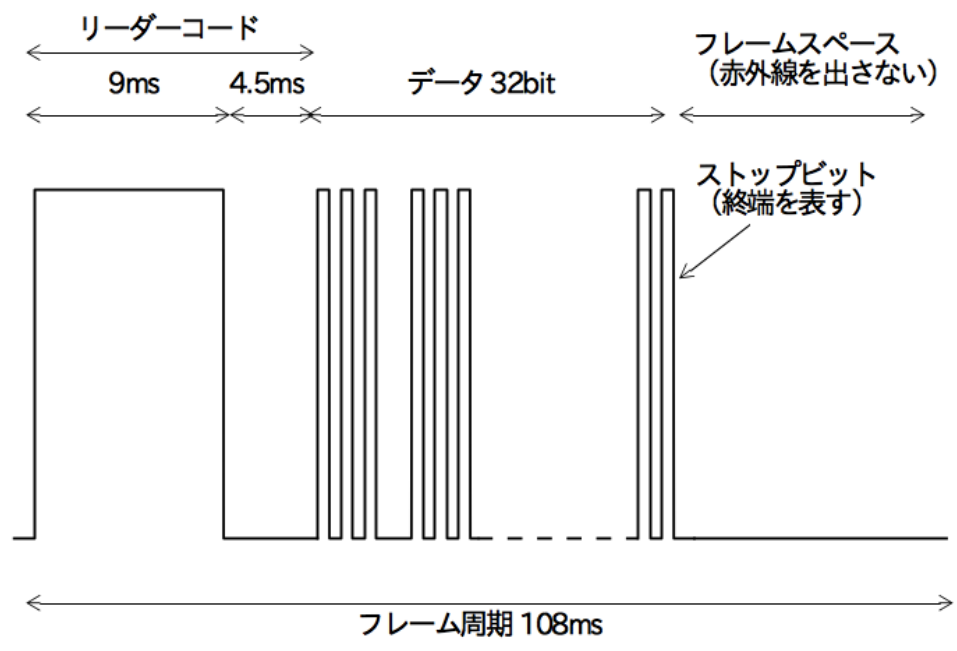


図 8　NEC フォーマットの信号 (1 フレーム分)。

リーダーコード

リーダーコードはフレームの開始を知らせるのが役割です。9ms の点灯の後、4.5ms 消灯します。図 8 を見てわかる通り、この部分は他の部分とは波形が大きく異なるため、容易に識別できます。

データ

データは 32bit 分の情報を表します。0/1 の区別は、赤外線の消灯時間により区別されます (表 1、図 9)。0 の場合は 0.56ms 点灯の後に 0.56ms 消灯し、1 の場合には 0.56ms 点灯の後に 1.69ms 消灯します。

実際の信号の場合、32bit のうち、16bit がカスタマーコード、8bit がデータコード、残り 8bit は反転データコード (データコードのビットを反転したもの) という内訳になっています。反転データコードはエラーの検出のために

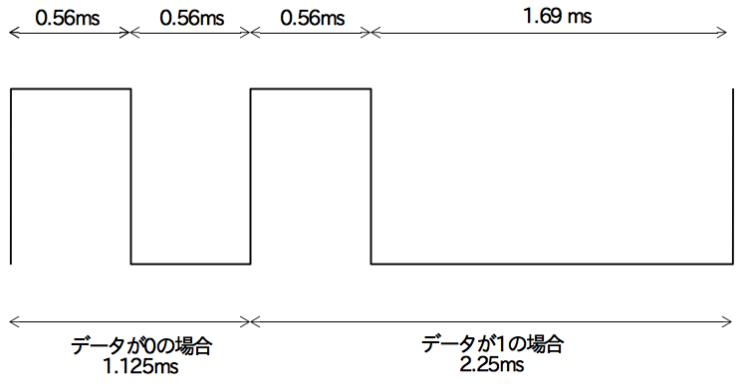


図 9　データが"01"の場合の信号。共に点灯時間は同じだが、消灯時間が異なる。

データ	0	1
点灯時間	0.56ms	0.56ms
消灯時間	0.56ms	1.69ms

表 1　データのビットの点灯時間/消灯時間。

用いられます。エラーの判定は

$$(\text{データコード})\,\&\,(\text{反転データコード}) = 0$$

をチェックすればよいです。

ストップビット・フレームスペース

データの後ろに 0.56ms の点灯 (ストップビット) をし、その後、リーダーコードの開始から 108ms に至るまで消灯します (フレームスペース)。受信側はストップビットを受け取った時点で受信処理を終了することになります。

キャリア周波数

自然界には多くの赤外線のノイズ源が存在しており、ノイズレベル以上の強度で赤外線を送信する必要があります。そのため、赤外線をただ点灯するだけではなく、実際には、図 10 のように、キャリア周波数で赤外線を ON/OFF します。こうすることで、同じ電力でも出力のレベルを上げることができます。

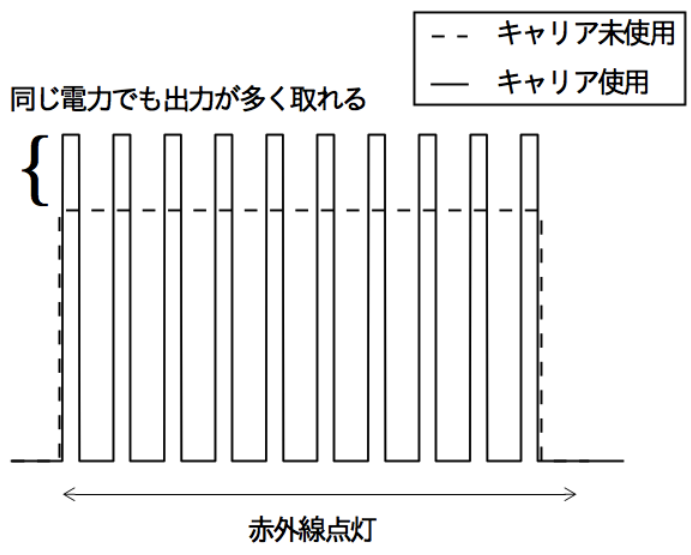


図 10　キャリア周波数のイメージ。

市販の赤外線受信モジュールを見てみると、「周波数 38kHz」のような記述を見かけます。これは、受信側でこの

周波数 (キャリア周波数) でバンドパスフィルタをかけおり、結果、よりノイズに強い通信を行うことができます。NEC フォーマットの場合、キャリア周波数は 38kHz です。

10 Arduino での赤外線の受信

Arduino で赤外線の信号を受信し、データの中身を見てみましょう。

Arduino と赤外線受信モジュールを図 11 のように配線します。

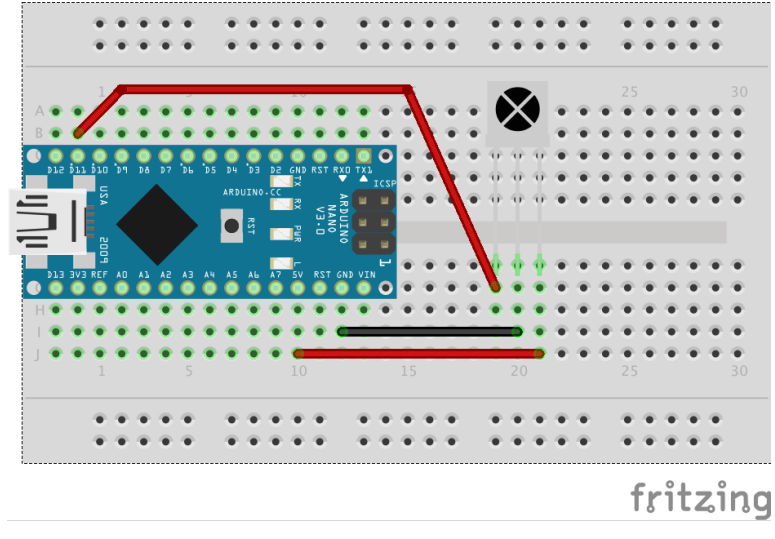


図 11　Arduino と赤外線受信モジュールの接続。信号、VCC、GND を接続する。

信号の解析には、Arduino-IRremote[*11] というライブラリを使用します。このライブラリを使うと、赤外線リモコン信号の送受信を簡単に行うことができます。信号のフォーマットも複数対応しています。

Arduino-IRremote のサンプルコードを使って赤外線の受信と信号の解析を行ってみましょう。Arduino-IRremote の中にある examples/IRrecvDump/IRrecvDump.ino というスケッチを Arduino に書き込みます。このサンプルコードを使うと、受信した信号のフォーマット (NEC, SONY, etc) と、値がシリアルモニタに表示されます。ソースコードの中に RECV_PIN という変数で赤外線受信モジュールの信号線を繋ぐピンを指定しています。今回はデフォルトの D11 にしますが、それ以外のピンを使用する場合は適宜変更します。

配線とスケッチの書き込みが済んだら、適当なリモコンの信号を読み取りましょう (図 12)。読み取った信号のフォーマットと値がシリアルモニタに表示されるはずです。

11 音声データから赤外線へ

ここでは、イヤホンジャックから出力される音声信号により、赤外線リモコンのキャリア周波数 38kHz で LED を点滅させる方法について述べます。

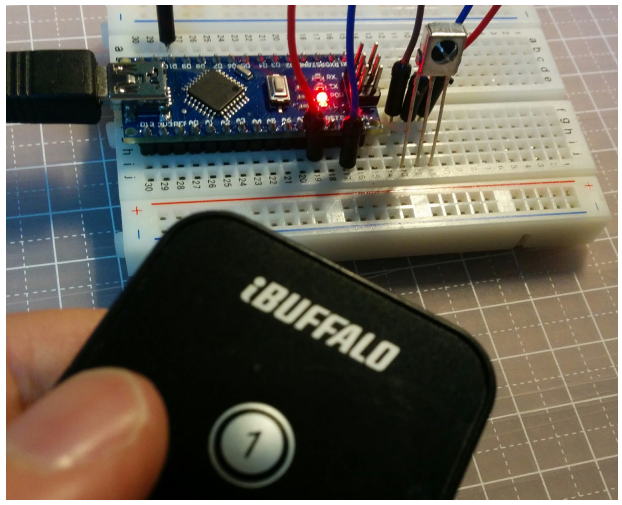
PCM データの作成

Java で PCM を WAVE ファイルに保存するサンプルコードを示します。PCM を作る部分は後で Android アプリを作る時に使いまわすことができます。WAVE ファイルの保存は javax.sound.sampled.AudioFormat にバイト列を渡すだけで簡単に作成できます。

```
import javax.sound.sampled.*;

...
```

[*11] https://github.com/z3t0/Arduino-IRremote



(a) HDMI 切替器のリモコン。

(b) 読み取った信号の内容。どうやら NEC フォーマットが使われているようだ。

図 12　リモコンの信号を受信して中身を見てみる。

```
// サンプリングレート 44.1kHz, 16bit (=2byte), ステレオ, 1秒
byte[] buffer = new buffer[44100 * 2 * 2];

// データの作成
for (int i = 0; i < buffer.length; i+=4) {
  buffer[i    ] = ...  // 左 下位8bit
  buffer[i + 1] = ...  // 左 上位8bit
  buffer[i + 2] = ...  // 右 下位8bit
  buffer[i + 3] = ...  // 右 上位8bit
}

// データの書き出し
AudioFormat format = new AudioFormat(44100, 16, 2, true, false);
AudioInputStream ais = new AudioInputStream(
                               new ByteArrayInputStream(buffer),
                               format,
                               buffer.length);
AudioSystem.write(ais, AudioFileFormat.Type.WAVE, new File("test.wav"));
```

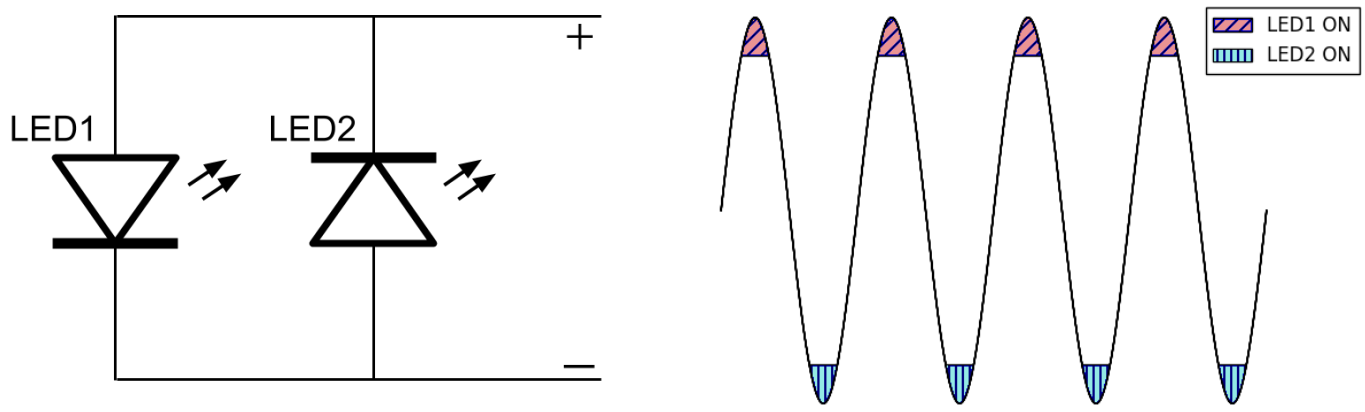
ここでは、サンプリングレートを 44.1kHz にしました。あまりテキトーにサンプリングレートを選ぶと再生できない場合がありますので、ここでは、CD と同じサンプリングレート **44.1kHz** を採用します。

２倍の周波数で点滅させる

38kHz の信号を離散データで表現するには、標本化定理から、少なくとも 2 倍の周波数である 76kHz で標本化する必要があります。したがって、サンプリングレート 44.1kHz では不十分です。しかし、**19kHz** の信号で **LED** を **38kHz** で点滅させる方法があります。

まず、LED を極性を逆にして接続します (図 13(a))。こうすると、(＋の電圧) ＞ (ーの電圧) の時は LED1 が点灯し、逆に (ーの電圧) ＞ (＋の電圧) の時は LED2 が点灯します。ただし、＋とー間の電圧は LED の降伏電圧を超えないものとします。

このとき、図 13(b) のような正弦波を入力するとどうなるでしょうか。信号が正にある程度大きくなると、LED1 が点灯します。一方、負にある程度大きくなると、LED2 が点灯します。すると、もとの信号の半周期ごとに LED が点灯し、すなわち、LED の点滅の周波数は、もとの信号の周波数の２倍になります。



(a) 回路図。LED の極性を逆にして接続する。

(b) 信号。＋ が大きくなると LED1 が点灯し、-が大きくなると LED2 が点灯する。

図 13　信号の２倍の周波数で LED を点滅させる回路。

パソコンから赤外線を送信してみよう

ここでは、実際にサンプリングレート 44.1kHz で、19kHz の信号を作成し、WAVE ファイルとして保存、再生を行い、赤外線を送信しています。

まず、WAVE ファイルを作成します。NEC フォーマットに従い、赤外線が ON の時には下記のコードのように正弦波を生成し、OFF の時には 0 を入れます。

```
    // サンプリングレート 44.1kHz
    static final int SAMPLING_RATE = 44100;
    // キャリア周波数 19kHz
    static final int IR_FREQ = 19000;

    ...

    // 赤外線が ON の間、サンプリングする点についてのループ
    for (int i = 0; i < sample_length; i++) {
      double t = (double)i / SAMPLING_RATE;

      // 正弦波の生成
      short v = (short)(Short.MAX_VALUE * Math.sin(2 * Math.PI * IR_FREA * t));
      // バッファの位置
      int index = i * 4;

      // リトルエンディアン
      buffer[index]     = (byte)(v & 0xff);           // 左 下位 8bit
      buffer[index + 1] = (byte)((v >>> 8 ) & 0xff);  // 左 上位 8bit
      // 右は左の音の符号を反転する
      buffer[index + 2] = (byte)(-buf[index]);        // 右 下位 8bit
      buffer[index + 3] = (byte)(-buf[index + 1]);    // 右 上位 8bit
    }

    // サンプリングレート 44.1kHz, 16bit, ステレオとして保存（省略）
```

ここでは、右側の音は左側の音の符号を逆にした値を入れています。こうすることで、左-右の電位差は、左-GND 及び右-GND の電位差の２倍になるので、出力を２倍にすることが出来ます。

ここで作成した WAVE ファイルを適当な音楽編集ソフトで開いて波形を確認しましょう。図 14 はフリー音楽編集ソフトである Audacity[*12]で見た、NEC フォーマットで 0xff0000ff に対応する波形です。思った通りの波形になっているでしょうか？

[*12] http://www.audacityteam.org/

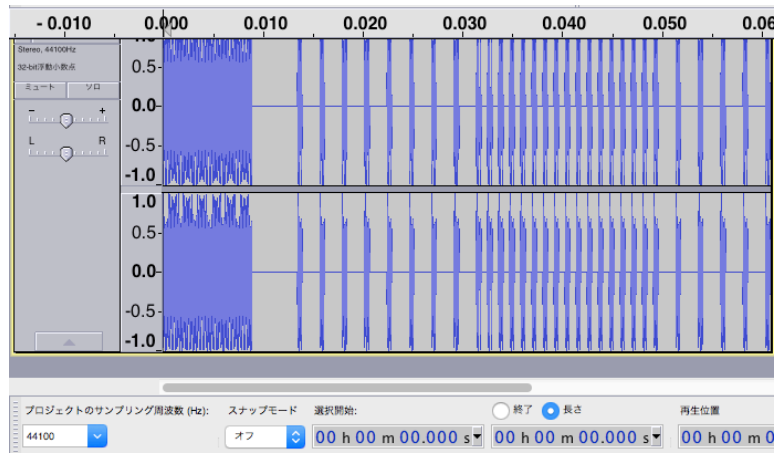
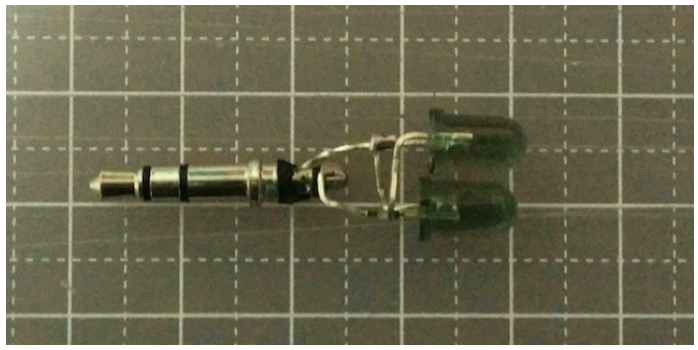
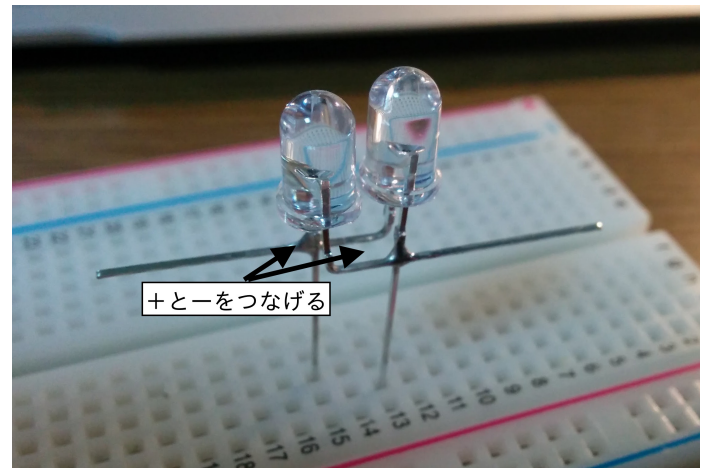


図 14　Audacity で見た NEC フォーマットで 0xff0000ff に対応する波形。

次に、イヤホンジャック直挿し赤外線 LED プラグ (図 15(a)) を作ります。図 13(a) の回路を思い出すと、２つの赤外線 LED を極性を逆にしてつないでいました。これを実際にやるには、単に赤外線 LED の長い足と短い足をつなげるだけです (図 15(b))。あとは適当なプラグの左と右にそれぞれはんだ付けするだけです。ちなみに私は 100 円ショップで買ったイヤホンを分解してプラグを取り出しました。



(a) イヤホンジャック直挿し赤外線 LED プラグ。



(b) 赤外線 LED のはんだ付け。長い足と短い足をつなげればいい。

図 15　イヤホンジャック直挿し赤外線 LED プラグの制作。

送信と受信のテスト

ここで、これまで取り組んだ音声信号→赤外線の送受信のテストをします。受信側は図 11 の配線と同じです。送信側は作成したイヤホンジャック直挿し LED プラグをパソコンのイヤホンジャックに差し込み、WAVE ファイルを再生してみましょう。Arduino のシリアルモニタに想定した整数値が表示されていますか？

私の場合、次のような問題があり、赤外線が送信されていない、または、受信されたのが全く違った信号になっている場合がありました。

音量の問題

再生側の音量が小さい場合、言い換えると、イヤホンの左右の間の電位差が小さすぎる場合、当然 赤外線 LED は点灯しないか、もしくは暗すぎて受信できません。音量を最大にしてみましょう。私の環境では、パソコン (MacBookAir) と iPad では音量を大きくすればうまいこと送信できていました。

再生するソフトの問題

これについては原因はわかりませんが、再生するソフトによっても違いがありました。波形を見る時に使った Audacity ではきちんと送信できていましたが、他のソフト (iTunes, VLC 等) では別の信号になっていました。編集ソフトでは波形を忠実に再生するのでしょうか？ iPad でも同様に、アプリによって違いがありました。

Android からの赤外線の送信

Android のスマートフォン (Nexus5) で直挿し LED プラグを使った送信をテストしてみたところ、何も受信されませんでした。原因は音量が小さいからだと考えられます。そこで、アンプを使って音声信号を増幅することにしました。今回使用したのは、秋月電子の超小型・低電圧駆動アンプモジュール*13です。乾電池１本〜２本で動作します。

使い方は簡単で、電源と入力＋/ー、出力＋/ーを接続するだけです。今回は電源にエネループ２本 (2.4V)、入力に Nexus5 のイヤホン左右、出力に赤外線 LED を接続します (図 16)。赤外線 LED の代わりに普通の LED を接続すれば、信号を送信する時に光るはずなので、通電チェックが出来ます。

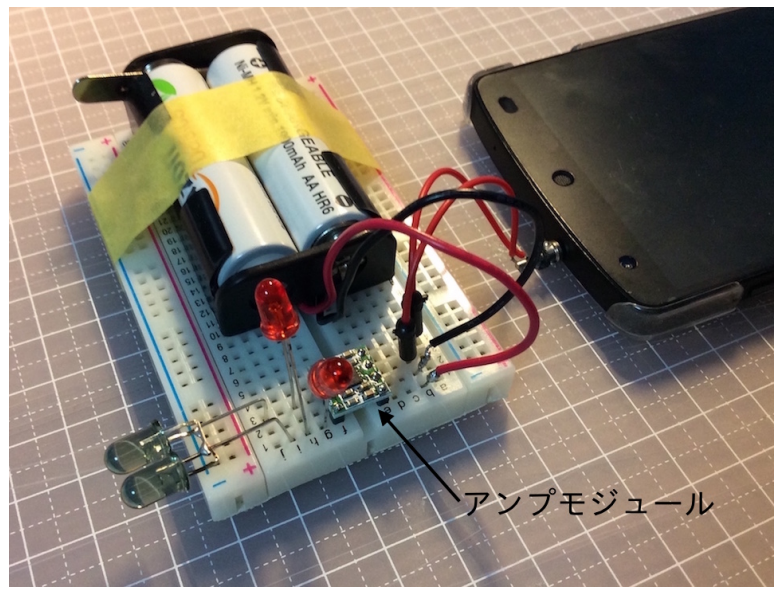


図 16　アンプモジュールのテスト。出力に赤外線 LED と、通電チェック用の赤色 LED 繋いでいる。

アンプを接続したら、送受信のテストをしてみましょう。アンプを使うことで、安定して想定した信号が受信されているのがシリアルモニタからわかると思います。

12　通信プロトコルの定義

スマホから Arduino へ赤外線で信号を送れるようになったところで、赤外線に乗せる 32bit の情報の中身を決めます (図 17)。

データ部

まず、全体の 32bit のうち、前半 16bit をデータ部、後半 16bit をデータ部のビットを反転したものとします。この反転したビットは、NEC フォーマット同様、エラーの検出に使用します。

$$(\text{データ部 16bit}) \,\&\, (\text{データ部の反転 16bit}) = 0$$

この判定をクリアできない命令は捨てることにします。

実際に送受信を行ってみるとエラーは 1bit では収まらないことも多く、パリティビットを付けるだけでは不十分だと思ったので、データ半分をエラーの検出に用いるようにしました。

*13 http://akizukidenshi.com/catalog/g/gM-06560/

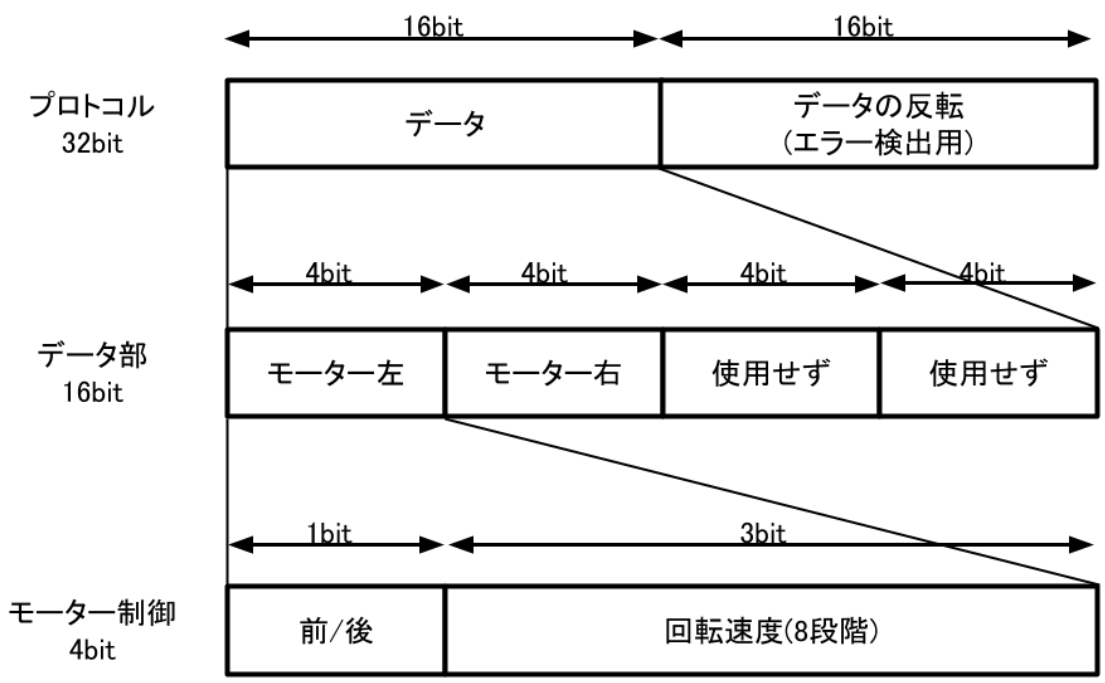


図 17　Arduino との通信のプロトコルの定義。全体が 32bit 、データ部 16bit、4bit の命令４つという構成。

データ部の内訳

16bit のデータ部には 4bit の命令が 4 つ格納されます。
今回は左右のモーターを動かすのに命令が２つ必要なので、

| モーター左 | モーター右 | 0000 | 0000 |

と命令が並びます。残りの 8bit は使用しません。将来的には砲塔を回転させるといった命令が入る予定です。

モーター制御

　モータの制御は 4bit の命令で行います。先頭 1bit が回転方向を表すフラグで、1 が前進、0 が後退を表します。残り 3bit で回転速度を 0〜7 の 8 段階で表します。

第 IV 部

戦車の制作

前の章では主に通信について説明しました。この章ではハードウェア (戦車本体と回路) と、マイコン及び Android のプログラミングについて説明します。

13　足回りの制作

ギアボックス

ツインモーターギアボックスはギアー比やシャフトの位置の組み合わせを３パターンの中から選べるようになっています。今回は標準仕様 **(**ギアー比 **58:1) A** タイプを選びます。

組み立ててそのままシャシーへ入れてみたのが図 18 です。シャフトの高さがちょうどシャシーの切れ込みの真ん中くらいに来ます。これならギアボックスの位置の調整は前後の位置のみで済みそうです。

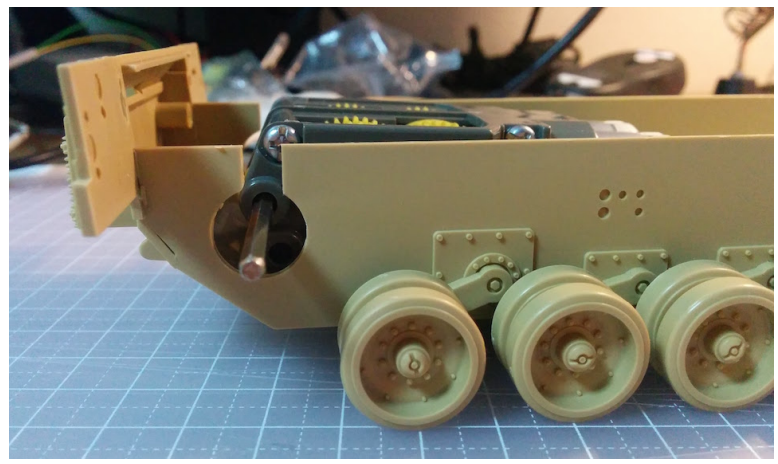


図 18　ギアボックスをシャシーへ入れてみると、なんかピッタリ！

シャフトと起動輪の固定

キャタピラで走行するためには、ギアボックスのシャフトの回転を起動輪に伝える必要があります。今回使用したプラモデルの起動輪にはラジコンキットの名残で、直径 6mm[*14]の六角形の穴が空いています (図 19)。一方ギアボックスのシャフトの断面は直径 3mm の六角形です。したがって、直径 **6mm** の中に直径 **3mm** の穴が空いた六角柱型の部品があれば、内側の穴にシャフトを挿入し、起動輪の六角形の穴にはめ込むことで、シャフトの回転を起動輪へ伝えることができます。

3D プリンターによる部品の制作

ちょうどいい部品を探したのですが見つからなかったので、今回は **3D** プリンターを使って作ってみることにしました。3D CAD は無料で使える **Autodesk 123D Design**[*15]で作成しました (図 20(a))。3D プリントは **DMM.make 3D PRINT**[*16]を利用しました。印刷するには、3D CAD で作成した STL ファイルををアップロードし、素材を選択して、支払いを済ませるだけです。今回は素材に **ABS** ライクを選択しました。値段は約 1,600 円、注文完了から 1 週間程度で届きました。実際に送られてきた部品が図 20(b) です。

[*14] 六角形の外接円の直径が 6mm の意。
[*15] http://www.123dapp.com/design
[*16] http://make.dmm.com/print/

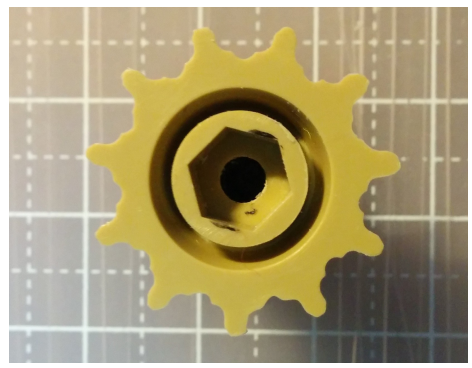
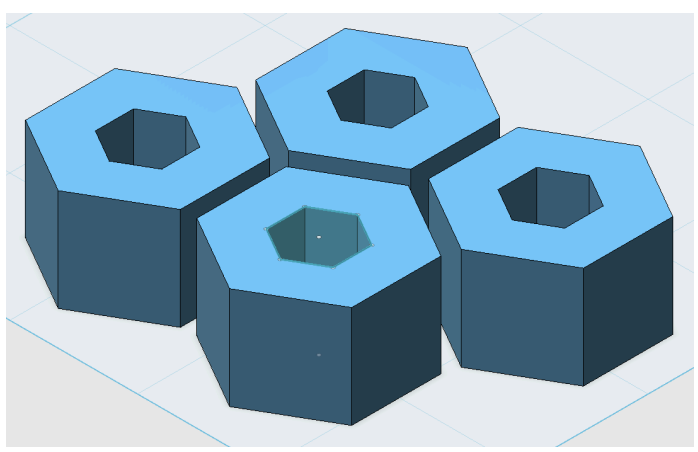


図 19　1/35 M1A1 エイブラムスの起動輪。六角形の穴が空いている。



(a) 3D CAD で作成したモデル。

(b) 3D プリンターで作成した部品。

図 20

3D プリンターで印刷した部品を使ってシャフトと起動輪を固定します。シャフトは長すぎるので、適当な長さで切断します。100 円ショップで買った金工用ノコギリを使って切断しましたが、切断するまでにかなり時間がかかりました。ノコギリ代をケチるなら少し覚悟が必要です。

シャフトを切断したら、3D プリンターで印刷した部品をシャフトに通します (図 21)。今回作成した CAD のデータでは、内側の穴の内径を、シャフトの直径と同じ 3mm に指定しました。実際に差し込んでみると少しキツ目になったようで、うまく固定できています。後で起動輪の位置を調整するため、あまり奥まで差し込まないようにします。

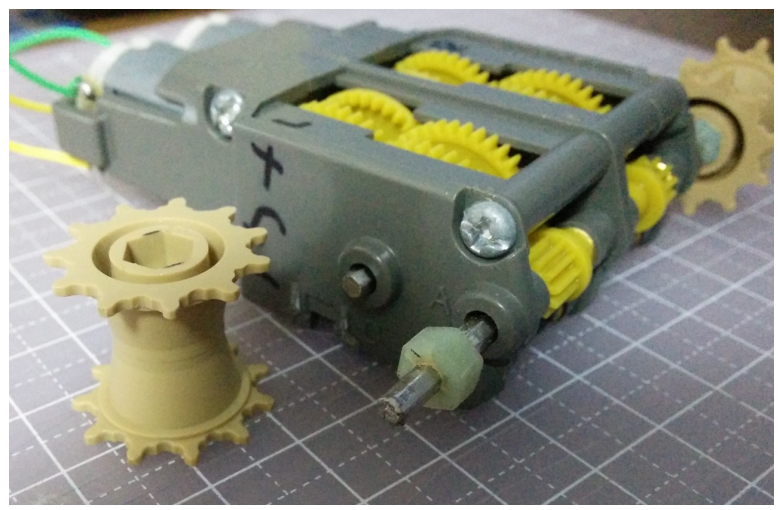


図 21　切断したシャフトと起動輪。

ギアボックスの固定と位置の調整

ギアボックスをシャシーに固定するための加工を行います (図 22)。まず、ニッパーで切断したり、ヤスリで削ったりしてギアボックスを置く時に邪魔になる突起を除去します。次に、ギアボックスをネジで固定するための穴を開けます。穴は縦長にしておくと、位置を前後できるため、キャタピラのテンションの調整ができるようになります。

穴を開けたら、ギアボックス付属のネジとナットで固定しましょう (図 23)。

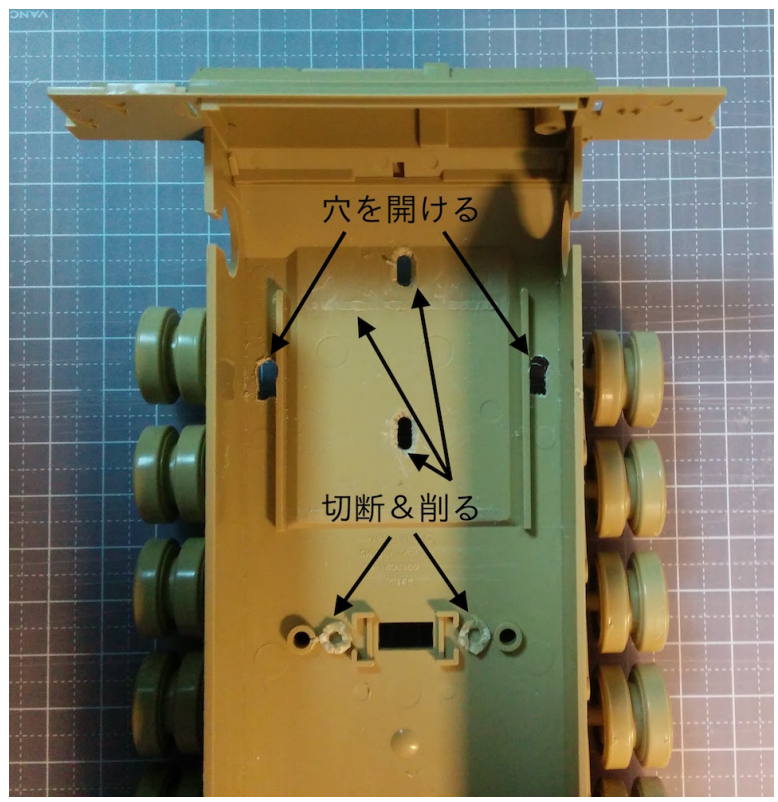


図 22　ギアボックス固定用の穴開けと邪魔な突起の除去。

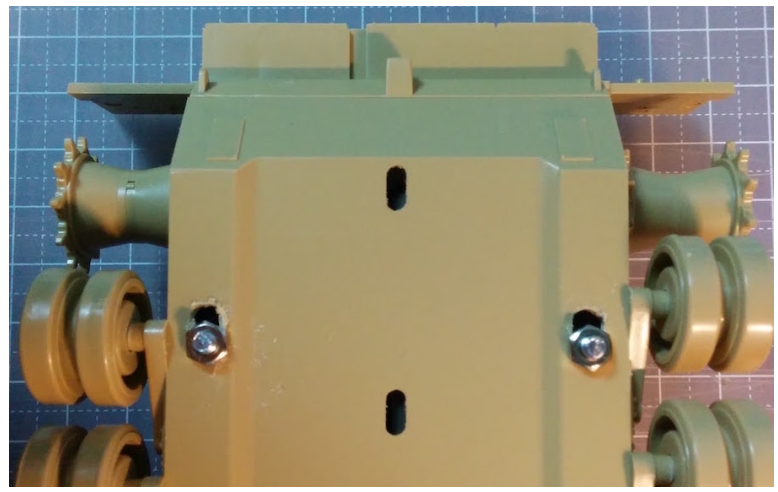


図 23　裏面から見たシャシー。ギアボックス付属のネジとナットで固定している。

14　回路の制作

戦車の頭脳となる回路は、１枚の基板に、Arduino Nano、モータードライバ、DC-DC コンバータ、赤外線受信モジュール、電池ボックスを乗せて作成しました (図 24,25)。トグルスイッチで電源が ON になります。

モーターは左上のターミナルブロックに接続します。モーターをはんだ付けしなかったのは、配線ミスで前後左

右逆になっても繋ぎ変えられるようにするためです[*17]。赤外線受信モジュールは設置する位置を調整する必要があるので、ケーブルで基板と繋いでいます。

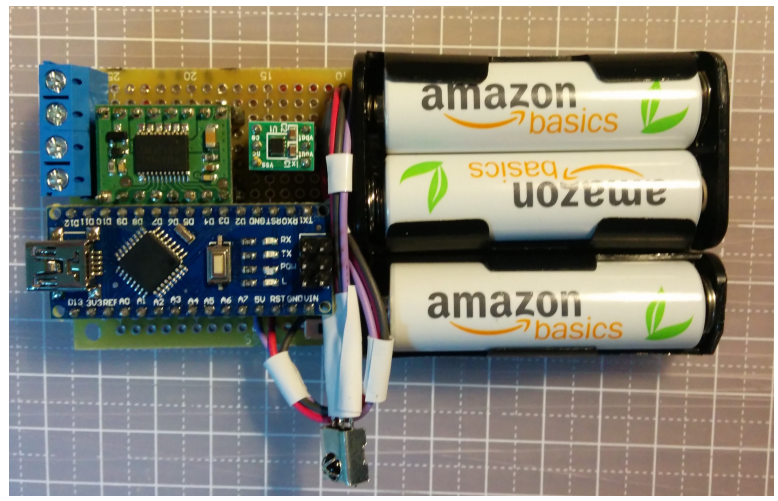


図 24　今回作成した回路。電池ボックスと一体にした。

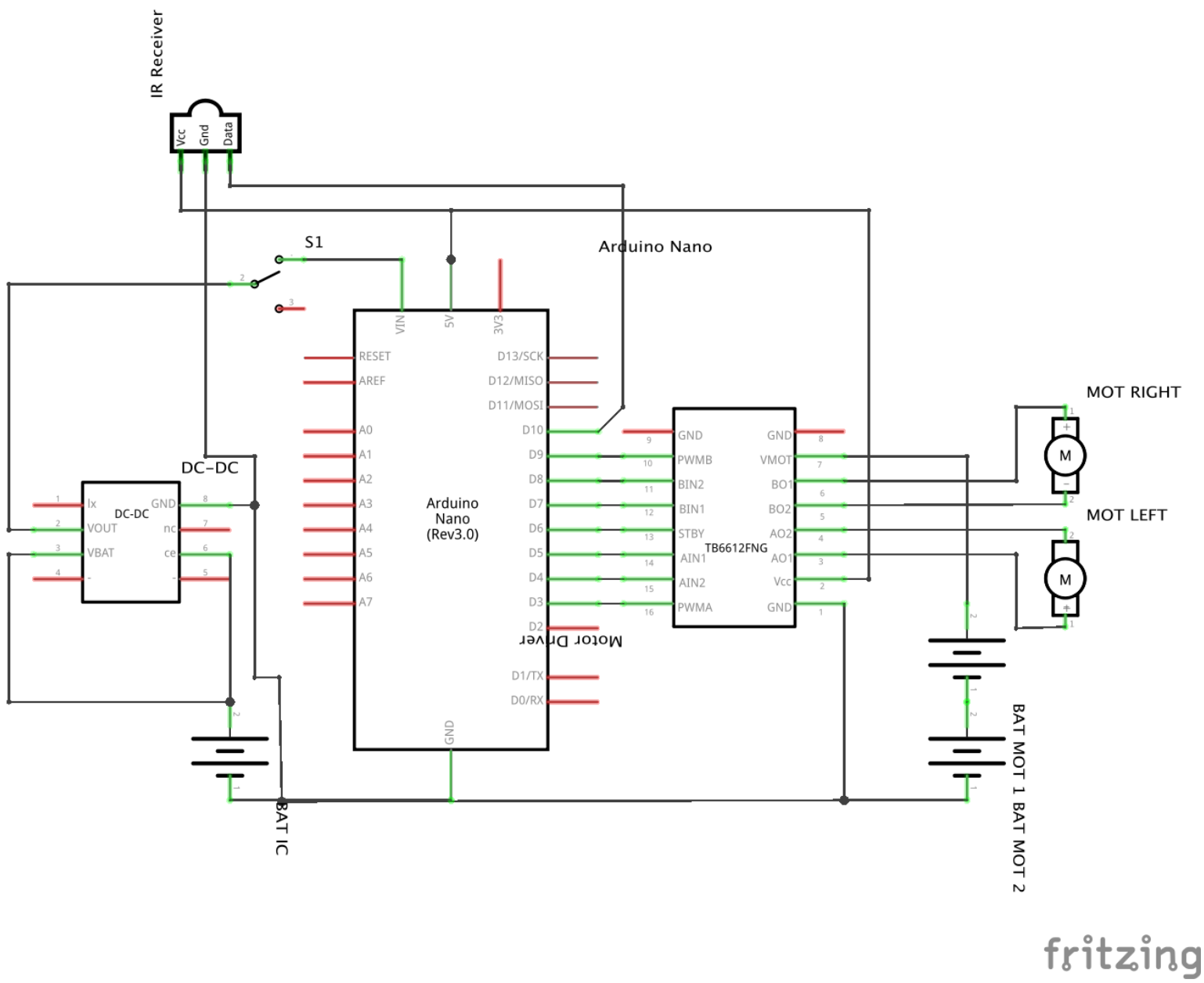


図 25　回路図。

[*17] 実際、配線ミスをしていたので、功を奏した

15　Arduino のスケッチの作成

Arduino のスケッチを作ります。

モーターの制御

入力				出力		
IN1	IN2	PWM	STBY	OUT1	OUT2	モード
H	H	H/L	H	L	L	ショートブレーキ
L	H	H	H	L	H	CCW
		L	H	L	L	ショートブレーキ
H	L	H	H	H	L	CW
		L	H	L	L	ショートブレーキ
L	L	H	H	OFF (ハイインピーダンス)		ストップ
H/L	H/L	H/L	L	OFF (ハイインピーダンス)		スタンバイ

図 26　モータードライバのデータシート。

マイコンは図 26 のモータードライバのデーターシート[*18]に従い、1 つのモーターにつき 2 つの GPIO の電圧を変えることによって回転方向を変えます。STBY はいつも HIGH にしておきます。回転速度はモータードライバに PWM で入力します。Arduino では PWM を 0〜255 の値で指定するので、受け取った 0〜7 の値を 0〜255 へマップし、モータードライバへ入力します。

loop 関数

赤外線信号の解析はライブラリがやってくれるので、loop 関数内で受信した値ごとに処理を振り分けるだけです。

- loop 関数内で受信した赤外線信号の解析をする。
- 受信した値の上位 16bit と下位 16bit を比べてエラーをチェックする。
- エラーがなければプロトコルに従いモーターを駆動する。

コードの一部を掲載します。

```
// 定義したプロトコルに従いモーターを駆動する
void work(int msg) {
  ...
}

// メインループ
void loop() {
  if (irrecv.decode(&results)) {  // 赤外線信号の解析
    long msg = (results.value >> 16) & 0xffff;  // 上位16bit
    long not_msg = results.value &     0xffff;  // 下位16bit

    if ((msg & not_msg) == 0) {
      // 信号にエラーがなければモーターを駆動する
      work(msg);
    } else {
      // エラー時の処理
    }
    irrecv.resume();
  } else {
    // 解析不能だった場合の処理
```

[*18] http://doc.switch-science.com/datasheets/TB6612FNG_datasheet_ja_20141001.pdf より引用。

```
    // 何度か続いたらモーターを止める
  }
}
```

16 Android アプリの作成

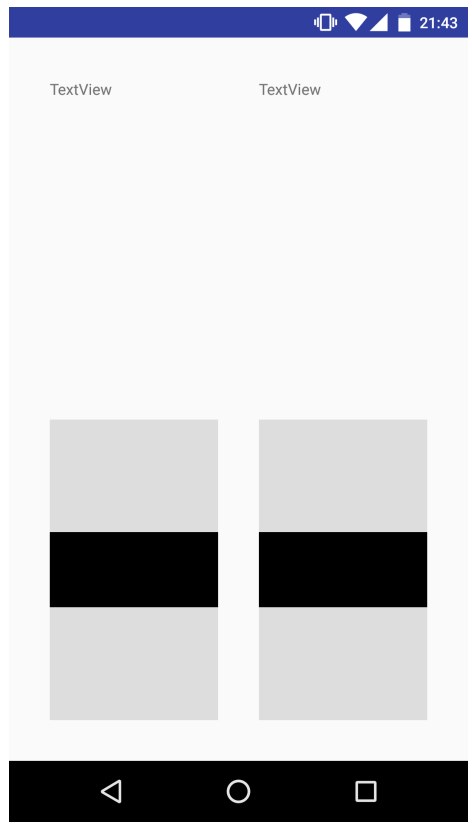


図 27 コントローラである Android アプリのスクリーンショット。実に殺風景である。

コントローラとなる Android のアプリを作ります。
機能として

- タッチの入力で左右のモーターの出力を決める。
- 定義したプロトコルに従い、赤外線で送信するデータ (32bit 整数値) を生成する。
- 赤外線の信号に対応する PCM を作成する。
- PCM を再生する。

を実装します。

試しに作ったアプリが図 27 です。２つあるバーの位置でモーターの出力を決めます。PCM データの作成は Java のサンプルコード (p.8) を使い回しています。サンプルコードでは AudioFormat クラスにバイト列を渡して WAVE ファイルとして保存しましたが、Android アプリでは代わりに android.media.AudioTrack にバイト列を渡して再生します。

```
    // AudioTrack にどういったフォーマットのデータなのか教えてあげる
    audiotrack = new AudioTrack(AudioManager.STREAM_MUSIC,
                                44100,
                                AudioFormat.CHANNEL_OUT_STEREO,
                                AudioFormat.ENCODING_PCM_16BIT,
                                minBufSize,
                                AudioTrack.MODE_STREAM);
    ...

    // 再生するとブロックされるので、別スレッドで実行する
```

```
    audiotrack.play();
    audiotrack.write(buffer, 0, buffer.length);
    audiotrack.stop();
    audiotrack.flush();
```

17　試運転

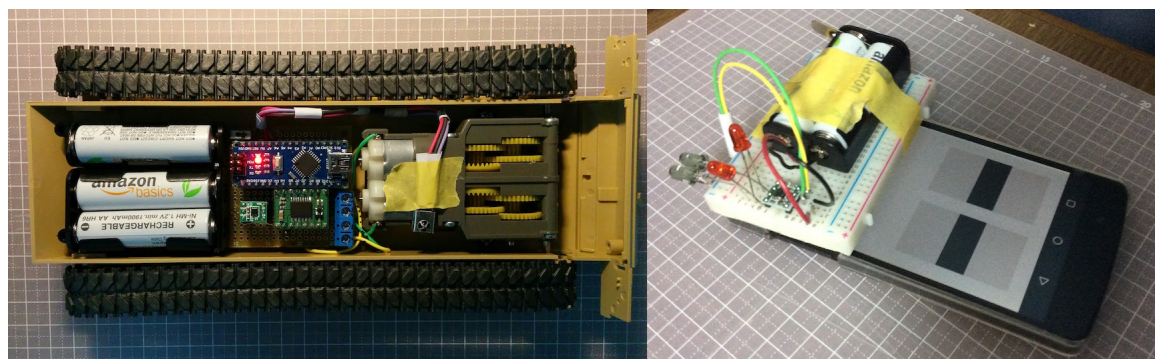


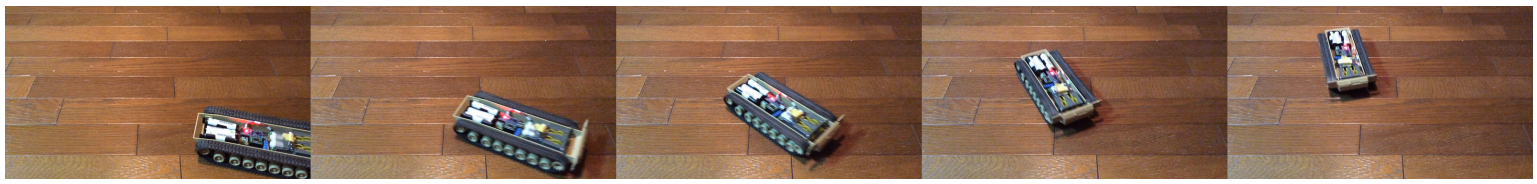
図 28　完成した足回りとコントローラ。ブレッドボードを画面の上半分にテープで貼り付けている。

回路をシャシーに入れて、Android アプリから操縦して遊んでみます！

概ねスムーズに操作を受け付けます。やはり赤外線は指向性があるので、向きによっては一瞬静止してしまう場合がありました。また、左のモーターは PWM による出力調整が出来ず、フルパワーで前後するだけになってしまいました。製作途中は正常に動いていたのですが・・・。現在原因を調査中です。

お待ちかねの信地旋回・超信地旋回をやってみます。

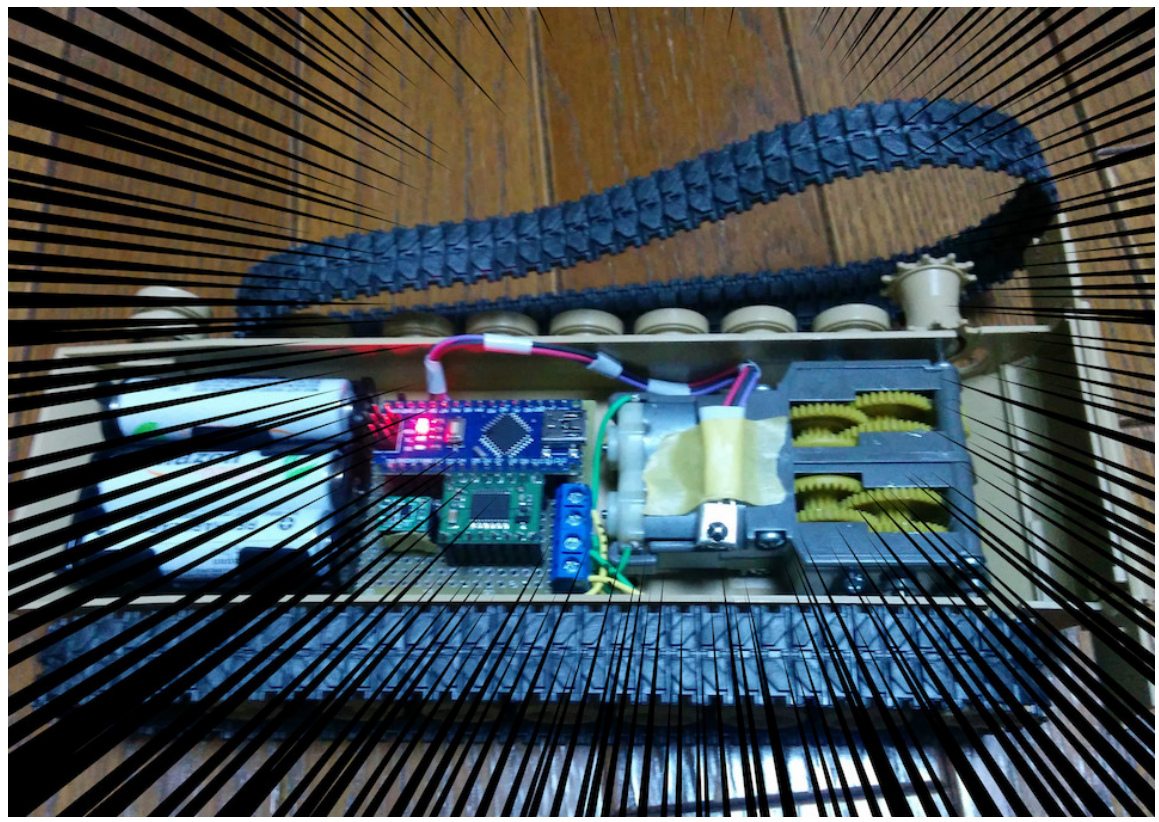
信地旋回！！



超信地旋回！！

ちゃんと動くじゃないか！！

と、調子乗って遊んでいる矢先・・・。



あああぁぁぁーーーーーーーっっ！！！

履帯が外れてしまいました・・・orz

　というわけで、今回はスマートフォンから赤外線を送って左右のキャタピラを独立に動かせる戦車ができました。結果はご覧の通り、安定した走行を実現するにはもうひと工夫必要なようです。今後はまず履帯のテンションを適切に保ち、はずれないようにする改良と、砲塔の旋回機構を実装する予定です。その後は、砲身の上下や、姿勢制御、サスペンションの可動化、エンジン音の再生など、色々取り組んでみたいです。

おくづけ

今年の５月頃からちまちま作り始め、なんとか動くものが作れました。ソフトもハードも設計がいい加減なので、これからも改良を重ね、完成度を高めていきたいです。いずれはプリント基板や部品の販売をして、より多くのガルパンおじさんにラジコン戦車の制作を楽しんでもらいたいと考えています。

それでは、パンツァー・フォー！！

2016 年 8 月、富士総合火力演習にて

日曜プログラマーでもできる！ ラジコン＆パンツァー Vol.1
2016 年 12 月 29 日 初版発行
サークル：御坂重工業
著者：ぜの
ホームページ：https://misakahi.github.io

御坂重工業