

令和元年度 環境ロボティクス学科 卒業論文

ワイヤレス給電システムの最適化のための の能率的な動作周波数スイープ

令和2年(2020)2月14日

森田 光流

指導教員：穂高一条教授

目次

1	緒言	3
1.1	背景	3
1.1.1	ワイヤレス給電の概要	3
1.1.2	ワイヤレス給電の特徴	3
1.1.3	ワイヤレス給電の将来性	3
1.2	研究目的	4
2	本論文の構成	5
3	本研究での基礎理論と原理	6
3.1	ワイヤレス給電の原理	6
3.2	共振周波数	7
3.3	平均電力・効率	8
3.3.1	定常解の導出	8
3.3.2	短形波入力の場合の定常解	9
3.3.3	平均電力と電力効率の導出	10
3.4	実験における平均電力の求め方	12
4	実験	14
4.1	周波数スイープについて	14
4.2	プログラムによる周波数スイープ自動化	15
5	実験結果	17
6	結言	18
6.1	考察	18
6.2	結論	18
6.3	今後の展望	19
7	謝辞	20
8	付録	22
8.1	ワイヤレス給電の状態方程式導出	22
8.2	オペアンプについて	24
8.3	プログラムについて	25

1 緒言

1.1 背景

1.1.1 ワイヤレス給電の概要

ワイヤレス給電とは、コネクタや金属接点の接触を用いず無線で電力を供給，伝搬することが可能な給電方法である．非接触充電，非接触電力送電，無線給電などとも呼ばれている．従来の電気製品の給電方法の金属接点やコネクタを使用したものは，水がかかると水による感電やショートを起こす，配線による転倒などの安全面に関して問題点がある．しかしワイヤレス給電は金属接点が非接触なので前述に述べた安全面の問題点の解決するだけではなく，人が立ち入れない危険な場所にある機器や装置の遠隔操作ができるなどという利点が期待されており，現在盛んに研究されている．

1.1.2 ワイヤレス給電の特徴

ワイヤレス給電が無線で電力を供給するためには，送電側と受電側の2つのコイルを用いるため物理的な金属接点やコネクタが存在せず，また非金属のものであれば，コイル間に存在していても送電側と受電側の電力に影響を及ぼさない．また，送電コイルを表面に露出させる必要がないため，水による感電やショートする心配がないので，水場での使用が可能である．なお，ワイヤレス給電の給電方式には送電側と受電側との間発生する誘導磁束を利用した電磁誘導方式や，キャパシタを送電電極と受電電極として電力を送電する電解結合方式などがある [4][9] が，本研究ではその中の電磁共鳴方式で行う．

1.1.3 ワイヤレス給電の将来性

現在，ワイヤレス給電の特徴を生かして様々な場所に導入しようと研究が行われている．前述に述べた通り金属接点が非接触なので，ワイヤレス給電化することにより物理的な金属接点が少なくなるので電源コードは最小限で済み余分な空間を作ることにより直接配線などの問題解決並びに自動化・効率化に大きく貢献している [10][1]．

また，地球温暖化の影響により CO₂ の削減のため電気自動車開発並びに普及が望まれている．現在，電気自動車に使用されている接触式充電方式は雨天時にプラグを扱うと感電する可能性がある．それをワイヤレス給電に置き換えると自宅の駐車場に駐車するだけで充電が可能になり電の可能性が払拭される，充電の際に運転者が降車する必要がなくなり運転者の負担が大きく軽減されることが期待される．[7]

1.2 研究目的

ワイヤレス給電の電力供給の効率の改善には、

1. 最適な周波数調整
2. 結合操作
3. マルチループコイルと LC 回路を用いた適応マッチング

などがある [8]. 本研究においては 1. に注目し, そのために最適動作周波数の探索をすることとした. 1. を最適動作周波数を探索するには, 受電電力と送電電力を使った電力効率を求める必要がある. しかし, 電力効率が高いだけでは最適動作周波数が適切であるとはいえない. 受電側と送電側の元々の電力が小さい場合も電力効率が高いということが考えられる. それではワイヤレス給電が最適に動作しているとは考えられない. したがって, 最適動作周波数は高効率かつ大電力が出力される周波数と定義される.

また最適動作周波数を探索するためには多くの周波数を測定しなければいけないので測定時間をなるべく短くしたい. しかし, ワイヤレス給電に周波数を送った後, 安定な電力が出力されるまで時間がかかる. したがって本研究は能率的な周波数スイープ設計し, ワイヤレス給電システムの最適動作周波数を能率的な周波数スイープによる測定, 考察を目的にした.

2 本論文の構成

緒言

本研究の背景および、研究目的をこの章で述べる．

本研究での基礎理論と原理

ワイヤレス給電についての原理，または本研究において扱うワイヤレス給電測定による基礎理論についてを述べる．

3 本研究での基礎理論と原理

3.1 ワイヤレス給電の原理

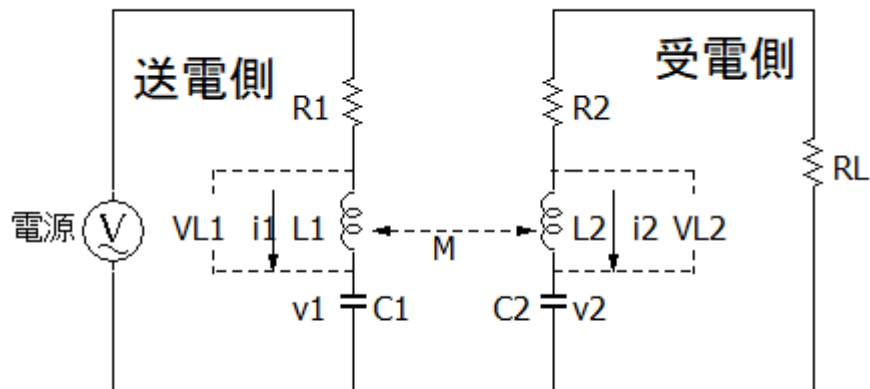


図 1: ワイヤレス給電回路図

本研究で扱うワイヤレス給電の回路は図1で示される。図1に示すように、ワイヤレス給電は送電側と受電側が電気的につながっていないことが分かる。送電側の交流電源の電流が時間的に変化することにより受電側の磁束が変化しファラデーの電磁誘導の法則より受電側に誘導起電力が生じ誘導電流を伝えるという原理である。また、図1の回路素子は以下の通りである。

- L1:送電側のコイルの自己インダクタンス。
- L2:受電側のコイルの自己インダクタンス。
- M:送電側と受電側のコイルの相互インダクタンス。
- R1:送電側コイルの内部抵抗
- R2:受電側コイルの内部抵抗
- C1:送電側のコンデンサの電気容量
- C2:受電側のコンデンサの電気容量
- RL:受電側の負荷抵抗
- 電源:送電側回路の交流電源

3.2 共振周波数

周波数と電力の関係として、共振周波数について説明する．共振周波数は回路が共振するときの周波数である．本研究において図2のRLC直列回路を使用する．この場合はRLC直列回路の共振周波数になる．

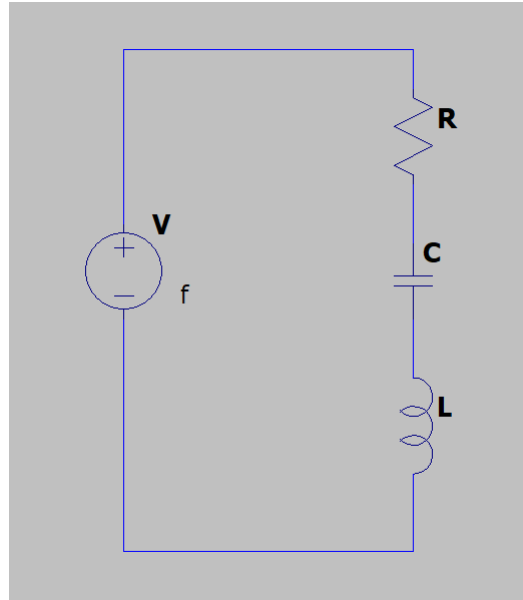


図 2: RLC 直列回路

共振周波数を求めるためにはまずインピーダンスを求める必要がある．図2の時インピーダンスは、

$$\begin{aligned}\dot{Z} &= R + j(X_L - X_C) \\ &= R + j\left(\omega L - \frac{1}{\omega C}\right)\end{aligned}\quad (1)$$

となる．このとき ω は角周波数であり、

$$\omega = 2\pi f \quad (2)$$

となる．RLC 直列回路が直列共振する条件は、1 の $X_L - X_C$ が 0 になるときなので

$$\begin{aligned}\omega L - \frac{1}{\omega C} &= 0 \\ \omega L &= \frac{1}{\omega C}\end{aligned}\quad (3)$$

$$\dot{Z} = R \quad (4)$$

となる．またその時の周波数を f_0 とし、式(2)式(3)から、

$$2\pi f_0 L = \frac{1}{2\pi f_0 C} \quad (5)$$

式 (5) から f_0 を解くと

$$f_0 = \frac{1}{2\pi\sqrt{LC}} \quad (6)$$

となる。

式 (4), 図 3 から RLC 直列回路が共振周波数近づくほどインピーダンスが小さくなり遠くなるほど大きくなる。インピーダンスが小さいと電流が大きくなる。したがって周波数が共振周波数のとき最も電力が大きいことが分かる。

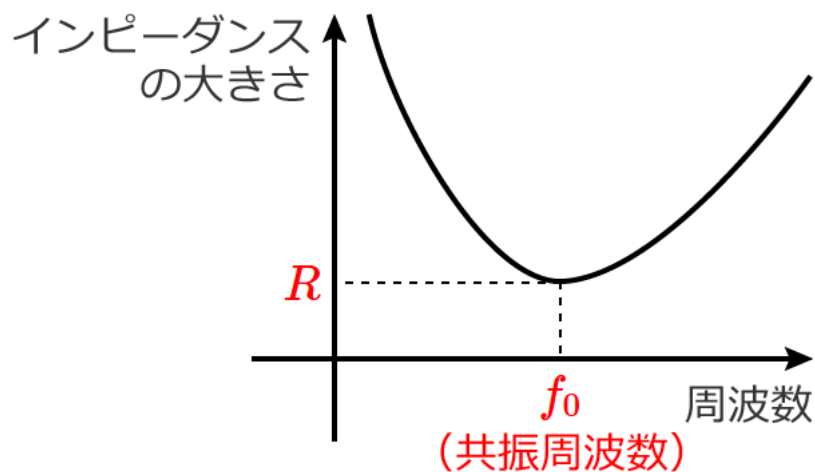


図 3: 周波数とインダクタンスの関係

3.3 平均電力・効率

ワイヤレス給電の最適動作周波数を探索するためには平均電力と効率を求める必要があるが、平均電力を求めるにはワイヤレス給電の定常解を求める必要がある。

3.3.1 定常解の導出

状態方程式は、一般的に

$$\dot{x} = Ax + Bu \quad (7)$$

と表せられる。(ワイヤレス給電の状態方程式の求め方は後述の付録参照)

この式 (15) を解くと,

$$e^{-At}\dot{x}(t) - e^{-At}Ax(t) = e^{-At}Bu(t) \quad (8)$$

記号の都合により, t を τ にして求めると,

$$\frac{d}{d\tau}(e^{-A\tau}x(\tau)) = e^{-A\tau}Bu(\tau) \quad (9)$$

$$\left[e^{-A\tau}x(\tau) \right]_0^t = \int_0^t e^{-A\tau}Bu(\tau)d\tau \quad (10)$$

$$\begin{aligned} e^{-At}x(t) - x(0) &= \int_0^t e^{-A\tau}Bu(\tau)d\tau \\ e^{-At}x(t) &= x(0) + \int_0^t e^{-A\tau}Bu(\tau)d\tau \end{aligned} \quad (11)$$

両辺に e^{At} をかけると,

$$x(t) = e^{-At}x(0) + e^{-At} \int_0^t e^{A\tau}Bu(\tau)d\tau \quad (12)$$

となり, 式(12)は状態方程式の一般解である. 平均電力と効率は状態方程式の周期的な解に基づくため, 式を周期的な関数にしなければならない. したがって $x(t)$ が周期 T をもつ関数とすると,

$$x(t+T) = x(t) \quad (13)$$

となり, 式(13)に式(12)を代入して整理すると,

$$x(0) = (I - e^{AT})^{-1} e^{A(T-t)} \int_0^T e^{-A\tau}Bu(t+\tau)d\tau - \int_0^t e^{-A\tau}Bu(\tau)d\tau \quad (14)$$

で求まる (但し, $\det(1 - e^{AT}) \neq 0$ の時だけ). 式(14)を式(12)に代入し, 周期解 $x_{ss}(t)$ を求めると,

$$\begin{aligned} x_{ss}(t) &= e^{At}(I - e^{AT})^{-1} e^{A(T-t)} \int_0^T e^{-A\tau}Bu(t+\tau)d\tau - \int_0^t e^{-A\tau}Bu(\tau)d\tau + \int_0^t e^{-A\tau}Bu(\tau)d\tau \\ x_{ss}(t) &= (I - e^{AT})^{-1} e^{At} \int_0^T e^{-A\tau}Bu(t+\tau)d\tau \end{aligned} \quad (15)$$

よって, 周期 T 定常解は式 15 で求められる.

3.3.2 短形波入力の場合の定常解

短形波入力の定義は,

$$\begin{aligned} u(t) &= \begin{cases} u_0 & (0 \leq t < Td) \\ 0 & (Td \leq t < T) \end{cases} \\ \text{かつ } u(t) &= u(t+T) \end{aligned} \quad (16)$$

であらわされる. T は 1 周期であり d はデューティー比である. そこから式(15)に式(19)を代入し範囲ごとに場合分けで積分し, $x_{ss}(t)$ を求める.

i. $0 \leq t < Td$ の場合

$$\begin{aligned}
 \int_0^T e^{-Ap} Bu(t+p) dp &= \int_0^{Td-t} e^{-Ap} Bu(t+p) dp + \int_{Td-t}^{T-t} e^{-Ap} Bu(t+p) dp + \int_{T-t}^T e^{-Ap} Bu(t+p) dp \\
 &= \int_0^{Td-t} e^{-Ap} Bu_0 dp + \int_{T-t}^T e^{-Ap} Bu_0 dp \\
 &= -A^{-1} e^{-AT} (e^{A(t+T(1-d))} - e^{At} - e^{AT} + I) Bu_0
 \end{aligned} \tag{17}$$

ii. $Td \leq t < T$ の場合

$$\begin{aligned}
 \int_0^T e^{-Ap} Bu(t+p) dp &= \int_0^{T-t} e^{-Ap} Bu(t+p) dp + \int_{T-t}^{t+dt-t} e^{-Ap} Bu(t+p) dp + \int_{t+dt-t}^T e^{-Ap} Bu(t+p) dp \\
 &= \int_{T-t}^{T+dt-t} e^{-Ap} Bu_0 dp \\
 &= -A^{-1} e^{-AT} (e^{A(t-Td)} - e^{At}) Bu_0
 \end{aligned} \tag{18}$$

となる。よって、式 (17) と式 (18) から $x_{sst}(t)$ は、

$$x_{sst}(t) = \begin{cases} -(I - e^{AT})^{-1} A^{-1} (e^{A(t+T(1-d))} - e^{At} - e^{AT} + I) Bu_0 & (0 \leq t < Td) \\ -(I - e^{AT})^{-1} (-I) (e^{A(t-Td)} - e^{At}) Bu_0 & (Td \leq t < T) \end{cases} \tag{19}$$

となる。

3.3.3 平均電力と電力効率の導出

ある時刻における電力を瞬時電力というが、平均電力は時間範囲内でその瞬時電力を時刻ごと加算して平均したものである。ここでは計算での平均電力と電力効率を求める方法を述べる

ある時刻における瞬間電力 p_t 、電圧 v 、電流 i とおくと

$$p_t = vi \tag{20}$$

と表すことができる。瞬間電力が周期 T をもつ場合、平均電力は 0 から T まで積分し、そして T で割ることにより求められる。よって、

$$P = \frac{1}{T} \int_0^T p_t dt \tag{21}$$

となる。また、送電側と受電側の瞬時電力は次のように表される。

$$p_{t1} = ui_1 \tag{22}$$

$$p_{t2} = R_L i_2^2 \tag{23}$$

一次側と二次側の平均電力は式 (22) と式 (23) より

$$P_1 = \frac{1}{T} \int_0^T u(t)i_1(t)dt \quad (24)$$

$$P_2 = \frac{1}{T} \int_0^T R_L i_2^2 dt \quad (25)$$

と導くことができる。

このように正弦波のような連続の関数を持つ周期関数はそのまま式 (24) と式 (25) に代入することが可能だが、短形波の平均電力は連続ではないので、注意して求める必要がある。したがって短形波の平均電力を求めるには、式 (15) の状態方程式と式 (19) の短形波の定常解を使って解くと、

$$P_1 = \frac{1}{T} \int_0^{Td} u i_{1ss1}(t) dt \quad (26)$$

$$P_2 = \frac{1}{T} R_L \left(\int_0^{Td} i_{2ss1}^2 dt + \int_{Td}^T i_{2ss2}^2 dt \right) \quad (27)$$

となる。また i_{1ss1}, i_{2ss1} は $0 \leq t < Td$ のときの i_1, i_2 で i_{2ss2} は $Td \leq t < T$ のときの i_2 である。また、電力効率 は 求めた P_1, P_2 より

$$\eta = \frac{P_2}{P_1} \quad (28)$$

と求まる。これにより、送電側と受電側の平均電力と効率を求められる。

3.4 実験における平均電力の求め方

前の小節にも述べた通り，最適動作周波数を求めるには送電側と受電側の平均電力と効率を求めなければならない．平均電力を求めるためには積分を使わなければならない．そこで，以下の方法が挙げられる．

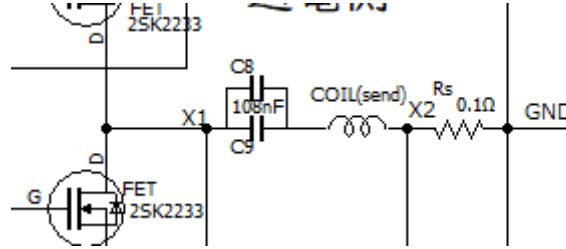


図 4: 送電側回路測定部分

図 (4) に示されているものは，送電側の回路図上にシャント抵抗 R_s を加えたものである．コイルに通る電流を i ，シャント抵抗の抵抗は $R_s = 0.1\Omega$ なので，シャント抵抗間の電圧は，

$$v_s = 0.1i \quad (29)$$

となる．

電力を求めるにはシャント抵抗の電圧とコイル間電圧 V_L を乗算器でかけることにより求めることができる．実験で使用した乗算器は乗算した値が 1/10 倍出力される特性を持つから，オペアンプでシャント抵抗の電圧を 100 倍にすることが求められる． v_s を 100 倍された値は，

$$0.1i \times 100 = 10i \quad (30)$$

となる．

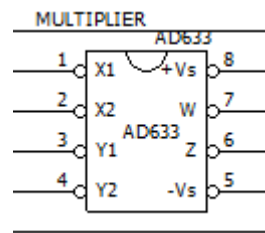


図 5: 乗算器回路図

図 (fig:zouhuku) の乗算器回路において，計算方法としては以下のとおりである．

$$W = \frac{(X1 - X2)(Y1 - Y2)}{10} + Z \quad (31)$$

コイル間電圧 V_L の電位の $X1$ と $X2$ を乗算器の $X1$ と $X2$, オペアンプを通した後の電圧を乗算器の $Y1$, 回路の GND を $Y2$ とすると, 図 (fig:shunt) 上での電力の以下になる. なお, 図 (10) の Z は GND につないでいる.

$$\begin{aligned}
 W &= \frac{(X1 - X2)(Y1 - Y2)}{10} + Z \\
 &= \frac{(X1 - X2)(10V_s - 0)}{10} + 0 \\
 &= \frac{V_L 10i}{10i} = V_L i
 \end{aligned} \tag{32}$$

となり実験中で電力を求めることができる.

4 実験

4.1 周波数スイープについて

実験による最適動作周波数を探索する方法としては周波数スイープが挙げられる。周波数スイープとは一定の時間や速度において周波数を変化させ出力させる方法である。なお本研究における周波数スイープは以下のようなやり方である。(図6)

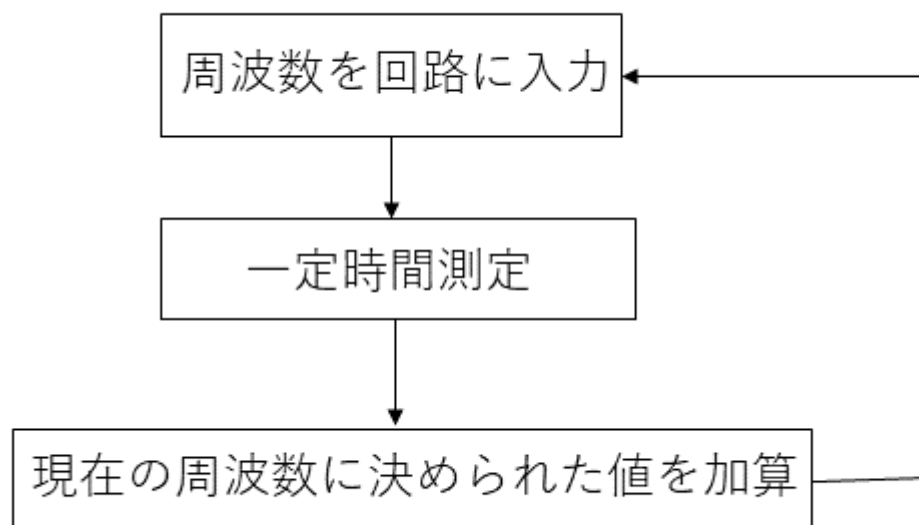


図 6: 周波数スイープ方法図

1. 一番最初の周波数を回路に送る。なお一番最初の周波数，一番最後の周波数，1つの周波数における測定時間，次の周波数で加算する周波数の量をあらかじめ決めておく必要がある。
2. 決めた時間で測定する。0.1秒ずつデータが出力される。
3. 一定時間に達したら，周波数に決めた量の周波数を加算し一番最後の周波数に達するまで繰り返す。

4.2 プログラムによる周波数スイープ自動化

先行研究では、ワイヤレス給電の周波数スイープを Arduino だけで実行した。しかし Arduino だけでは周波数や時間を変更する時、わざわざコンパイルしなければならず面倒であった。そこで本研究では周波数スイープの周波数を送る、ワイヤレス給電の電力を受け取る作業を python で自動化させるようにした。

一連の操作は以下の箇条書きと図7のとおりである。

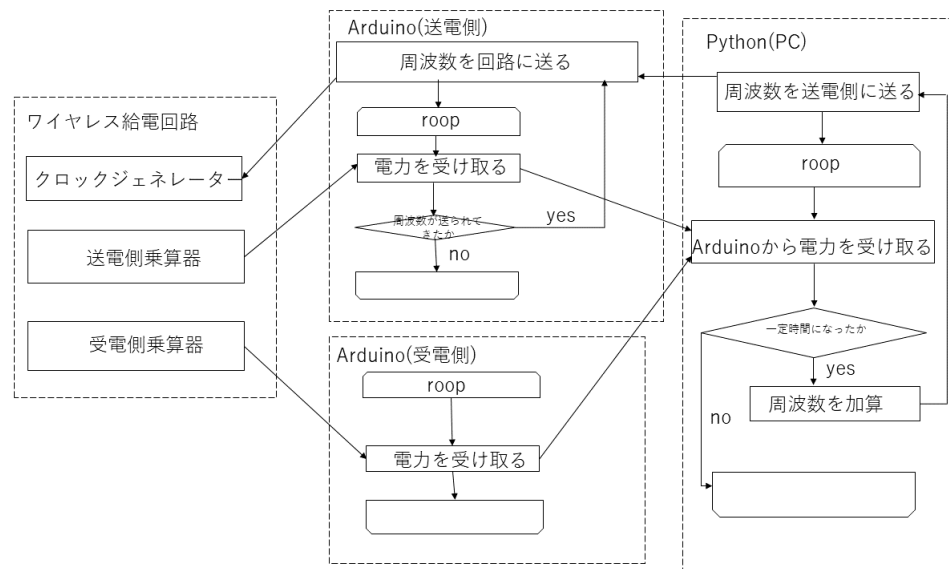


図 7: 周波数スイープ自動化のフローチャート

1. python によって送られてきた周波数の値を送電側の Arduino で、ワイヤレス給電回路の中にある矩形波を出力することができるクロックジェネレータへ指定した周波数を送る。
2. 乗算器によって出力された電力を送電側・受電側の Arduino にデューティー比で送る。また送られたデューティー比を電力に直す。
3. 電力を python へ送りデータを保存する。また自分が指定した時間が経ったら周波数を変えて、送りなおす。
4. 1～3 までの一連の操作を最後の周波数になるまで送り続ける。

以上のような流れですべての周波数の時の電力の測定をする。また、最初の周波数、最後の周波数、周波数をなん Hz ずつ見るための加算する周波数、一つの周波数の電力を測定する時間を GUI で入力できるようにした。次の図 (8) は本実験において使用した GUI である。また作成した python と arduino のプログラムは本論文の付録に書かれている。

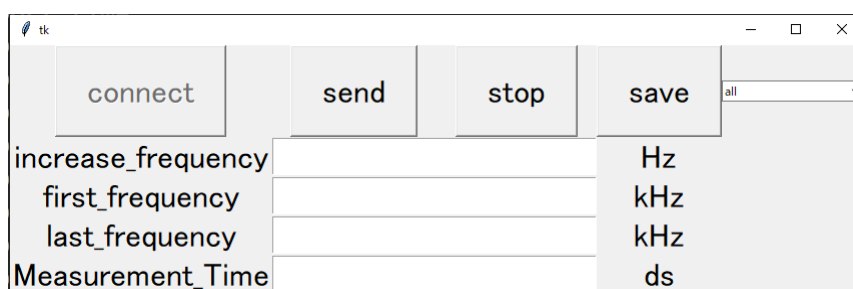


図 8: 電力測定するための GUI

5 実験結果

6 結言

6.1 考察

6.2 結論

6.3 今後の展望

7 謝辞

本研究の進行や本論文等の執筆にあたり，ご指導いただいた穂高一条教授に感謝の意を示すとともに深く御礼申し上げます。また本研究を進めるにあたり多大な助言・アドバイスをしてくださった自動制御研究室の先輩である白銀聡一郎さん，大畑貴弘さん，橋本菜生さん，松浦健斗さん，並びに共に研究した同期のメンバーも感謝の意を示すとともに深く御礼申し上げます。最後になりましたが，お世話になりました宮崎大学工学部環境ロボティクス学科の先生方，並びに大学関係各位の皆様にご心より感謝し，ここに御礼申し上げます。

参考文献

- [1] 松田一志：“ワイヤレス給電システムのための電力測定回路の開発”，宮崎大学学士論文，平成 30 年度
- [2] 中村裕馬：“ワイヤレス給電のための送電側 100kHz プッシュプル回路”，宮崎大学学士論文，平成 30 年度
- [3] ローム株式会社：“ワイヤレス給電とは”ーエレクトロニクス豆知識，
<https://www.rohm.co.jp/electronics-basics/wireless-charging/wireless-charging-what1>，最終アクセス：2020/1/20
- [4] ローム株式会社：“ワイヤレス給電（無線給電）方式”ーエレクトロニクス豆知識，
<https://www.rohm.co.jp/electronics-basics/wireless-charging/wireless-charging-what2>
- [5] keicode.com-技術入門シリーズ：“Tkinter による GUI プログラミング”ーPython 入門，
<https://python.keicode.com/advanced/tkinter.php>，最終アクセス：2020/1/21
- [6] 五位塚潤：“低周波数域駆動によるワイヤレス給電回路”，宮崎大学学士論文，平成 29 年度
- [7] 白銀聡一郎：“ワイヤレス給電のための矩形波電源装置の設計と開発”，宮崎大学学士論文，平成 29 年度
- [8] 盛田穰文：“ワイヤレス給電システムの効率と電力の最適化について”，宮崎大学修士論文，平成 29 年度
- [9] 伊東雅浩：“短径波入力による高効率ワイヤレス給電の制御について”，宮崎大学修士論文，平成 30 年度
- [10] B&PLUS：“ワイヤレス給電の現状と未来”，
<https://www.b-plus-kk.jp/about/mechanism.html> 最終アクセス：2020/1/23

- [11] 電気の資格とお勉強:”RLC 直列共振回路”, <https://eleking.net/study/s-accircuit/sac-resonant-rlcs.html> 最終アクセス : 2020/2/01

8 付録

8.1 ワイヤレス給電の状態方程式導出

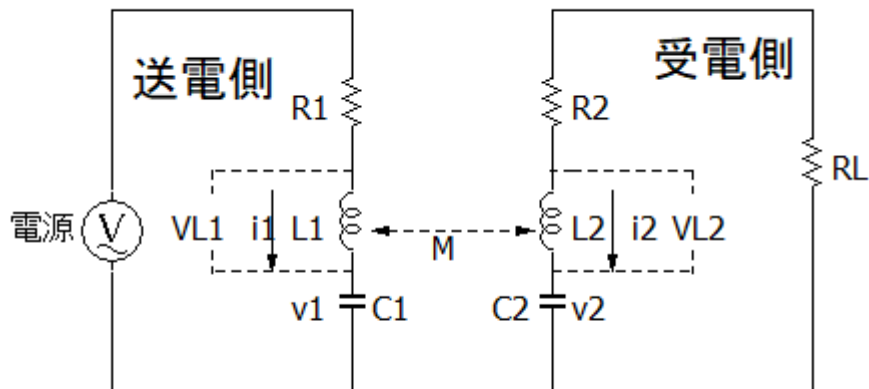


図 9: ワイヤレス給電回路図

図9の定数と電源の値を u として与えると、オームの法則とキルヒホッフの法則より、

$$u = R_1 i_1 + v_1 + v_2 \quad (1)$$

$$(R_{L1} + R_2) i_2 + v_1 + v_2 = 0 \quad (2)$$

コンデンサーの電圧と電流の関係より、

$$C_1 \dot{v}_1 = i_1 \quad (3)$$

$$C_2 \dot{v}_2 = i_2 \quad (4)$$

V_{L1} と V_{L2} を相互インダクタンス M を含めた形にすると

$$V_{L1} = L_1 \dot{i}_1 + M \dot{i}_2 \quad (5)$$

$$V_{L2} = M \dot{i}_1 + L_2 \dot{i}_2 \quad (6)$$

式 (3) と式 (4) を変形すると、

$$\dot{v}_1 = \frac{i_1}{C_1} \quad (7)$$

$$\dot{v}_2 = \frac{i_2}{C_2} \quad (8)$$

それぞれ式 (1) に式 (5) を、式 (2) に式 (6) を代入して i で揃えると

$$\dot{i}_1 = \frac{Mv_2 - L_2v_1 - R_1L_2i_1 + (R_L + R_2)Mi_2 + L_2u}{\Delta} \quad (9)$$

$$\dot{i}_2 = \frac{Mv_1 - L_1v_2 + MR_1i_1 - (R_L + R_2)L_1i_2 + Mu}{\Delta} \quad (10)$$

但し Δ は

$$\Delta = L_1L_2 - M^2$$

である．式 (9) と式 (10) を行列式でまとめると、

$$\begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \\ \dot{i}_1 \\ \dot{i}_2 \end{bmatrix} = \frac{1}{\Delta} \begin{bmatrix} 0 & 0 & \frac{\Delta}{C_1} & 0 \\ 0 & 0 & 0 & \frac{\Delta}{C_2} \\ -L_2 & M & -R_1L_2 & (R_L + R_2)M \\ M & -L_1 & R_1 & -(R_L + R_2)L_1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ i_1 \\ i_2 \end{bmatrix} + \frac{1}{\Delta} \begin{bmatrix} 0 \\ 0 \\ L_2 \\ -M \end{bmatrix} u \quad (11)$$

となる．この行列は以下のように A, B と状態変数 x として表すと、

$$A = \frac{1}{\Delta} \begin{bmatrix} 0 & 0 & \frac{\Delta}{C_1} & 0 \\ 0 & 0 & 0 & \frac{\Delta}{C_2} \\ -L_2 & M & -R_1L_2 & (R_L + R_2)M \\ M & -L_1 & R_1 & -(R_L + R_2)L_1 \end{bmatrix} \quad (12)$$

$$B = \frac{1}{\Delta} \begin{bmatrix} 0 \\ 0 \\ L_2 \\ -M \end{bmatrix} \quad (13)$$

$$x = \begin{bmatrix} v_1 \\ v_2 \\ i_1 \\ i_2 \end{bmatrix} \quad (14)$$

$$\dot{x} = Ax + Bu \quad (15)$$

となり、式 15 はワイヤレス給電の状態方程式となる．

8.2 オペアンプについて

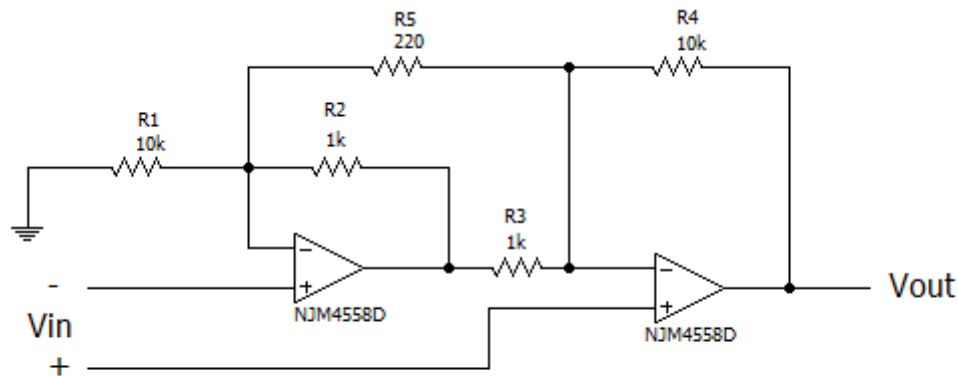


図 10: 計装用増幅回路

図 10 のように先行研究では計装用増幅回路を組み込んだオペアンプを使用した。しかしこの回路での出力は出力前と出力後で大きなずれが生じ、乗算器でうまく乗算することができない欠点があった。また周波数を上げるほど本来上げるべき倍率である 100 倍にうまく出力されない欠点 (周波数が高ければ高いほど出力される倍率が小さくなった) もあった。

そこでそこで今研究ではオペアンプを先行研究のように接続するのではなく、図 (11) 反転増幅回路を二つつなげて作った回路を使用した。また入力値と出力値とのずれを抑えるため $1000pF$ 程のコンデンサを入れることによりずれを防ぐようにした。

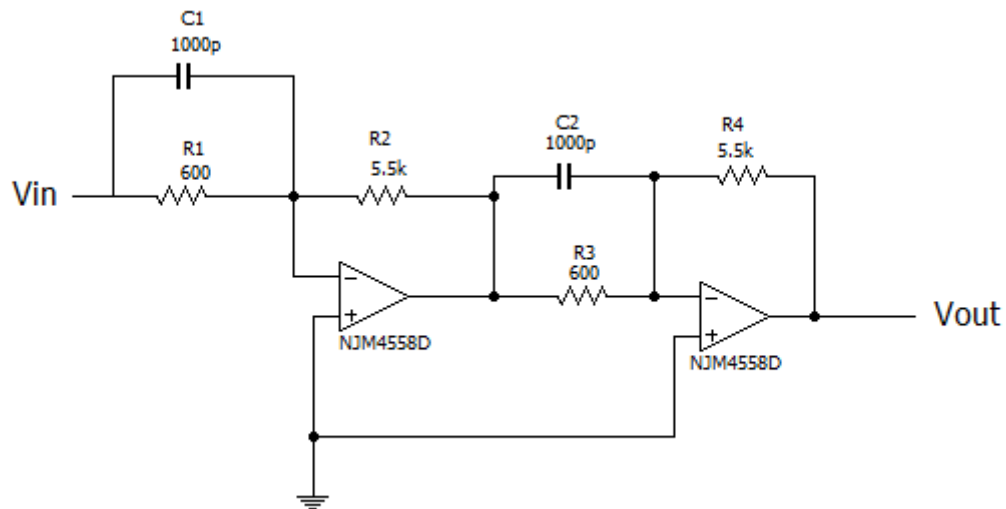


図 11: 反転増幅回路回路図

8.3 プログラムについて

前章に示された通り先行研究では周波数を変更する方法で、わざわざ Arduino のプログラムの一部を変更して再コンパイルさせ出力結果をシリアルモニターに表示させる方法であった．その面倒を省くため python を利用して周波数を arduino に送り，arduino の出力結果をデータに保存させる GUI を作成した．以下のプログラムは GUI で周波数を arduino へシリアル通信で送信して arduino に出力されたデータをシリアル通信で python 側に送る流れをするための，それぞれ GUI の python プログラムと送電側のマイコンの arduino プログラム，受電側の arduino プログラムである．なお，python のプログラムにおいて GUI を作成に tkinter を使用した．tkinter の使用方法並びに python の使い方については参考文献：[5] を参考にした．

ソースコード 1: GUI プログラム

```
1 import time
2 import statistics
3 import math
4 import serial
5 import collections
6 import csv
7 import tkinter as tk
8 import tkinter.filedialog as tkFileDialog
9 import tkinter.font as tkFont
10 import tkinter.ttk as ttk
11
12 x=0
13 L=[] #dataを保存
14 L1=[]
15 L2=[]
```

```
16 Lmsave=[]
17 Lmsend=[]
18 Lmreceive=[]
19 fre=0 #測定範囲の最小値
20 laf=0 #1目盛りの周波数
21 data=0 #測定範囲の最大値
22 ser1=0 #送電側のシリアル通信
23 ser2=0 #受電側のシリアル通信
24 t=1
25 tm=0
26
27
28
29 def maindef():
30     global x
31     global L
32     global ser
33     global fre
34     global data
35     global laf
36     global ser1
37     global ser2
38     global t
39     global tm
40     global L1
41     global L2
42     global Lmsave
43     global Lmsend
44     global Lmreceive
45
46     if x==0:
47         t=1#一応
48     elif x==1:
49         #次の周波数を入力
50         if float(fre) > float(laf):
51             x=3
52         else:
53
54             ser1.write('a'.encode('ascii')) # arduinoへ開始の合図を送る。
55             ser2.write('a'.encode('ascii'))
56             ser1.write(fre.encode('ascii'))
57             ser2.write(fre.encode('ascii'))
58             ser1.flush() # バッファ内の待ちデータを送りきる。
59             ser2.flush()
60             print("send:"+fre+"kHz")
61             L.append(fre+"kHz")
62             x=2
63             t=1
64         elif x==2:
65             #データ受け取りスイープ
66             line1 = ser1.readline().decode('ascii').rstrip()
67             line2 = ser2.readline().decode('ascii').rstrip()
68             L1.append(line1)
69             L2.append(line2)
70             print(fre+" "+line1+" "+line2+" ")
71             L.append(fre+" "+line1+" "+line2+" ")
72             if t>=int(tm):
73                 math1=collections.Counter(L1).most_common()[0][0]
74                 math2=collections.Counter(L2).most_common()[0][0]
75                 Lmsave.append(fre+" "+math1+" "+math2)
76                 Lmsend.append(fre+" "+math1)
77                 Lmreceive.append(fre+" "+math2)
78                 if float(fre)+float(data)*0.001 > float(laf) and float(laf) > float(fre):
79                     fre=laf
80                 else:
81                     fre=str(round(float(fre) + float(data)*0.001,3))
82                     x=1
83             L1=[]
```

```

84 L2=[]
85 else:
86 t=t+1
87
88 elif x==3:
89 #データを送らない、後始末
90 stop_data()
91 x=0
92 elif x==4:
93 #データ受け取り通常
94 line1 = ser1.readline().decode('ascii').rstrip()
95 line2 = ser2.readline().decode('ascii').rstrip()
96 print(fre+" "+line1+" "+line2+" ")
97 L.append(fre+" "+line1+" "+line2+" ")
98
99 root.after(10,maindef)
100
101 class Ser:
102 def __init__(self):
103 self.ser=None
104
105 def start_connect(self):
106 global ser1
107 global ser2
108 comport1='COM4' # arduino ideで調べてから。送電側
109 comport2='COM3' #受電側必ず comportは送電側受電側異なるものを使用
110 tushinsokudo=57600 # arduinoのプログラムと一致させる。
111 timeout=5# エラーになったときのために。とりあえず5秒で戻ってくる。
112 ser1=self.ser
113 ser2=self.ser
114 ser1 = serial.Serial(comport1,tushinsokudo,timeout=timeout)
115 ser2 = serial.Serial(comport2,tushinsokudo,timeout=timeout)
116 time.sleep(2) # 1にするとダメ！短いほうがよい。各自試す。
117
118 def send_com(self):
119 global x
120 global data
121 global fre
122 global laf
123 global ser1
124 global ser2
125 global L
126 global tm
127 global L1
128 global L2
129 global Lmsave
130 global Lmsend
131 global Lmreceive
132 # v,u,sの文字列は、
133 #それぞれv.get(),u.get(),s.get()で取り出す。
134 #下部send_entry内のTextvariableでデータ入力
135 data=v.get()
136 fre=u.get()
137 laf=s.get()
138 tm=v1.get()
139 if data.isdecimal()==True and fre.isdecimal()==True and laf.isdecimal()==True and tm.isdecimal()==True:
140 ser1.write('a'.encode('ascii')) # arduinoへ開始の合図を送る。
141 ser2.write('a'.encode('ascii'))
142 ser1.write(fre.encode('ascii'))
143 ser2.write(fre.encode('ascii'))#送電側と受電側の送るデータの量を合わせるため、
144 #あえて周波数を送る。送らなかった場合、送電側と受電側の出力にずれが生じるから。
145 ser1.flush() # バッファ内の待ちデータを送りきる。
146 ser2.flush()
147 print("send incese_fre:"+data+" first_fre:"+fre+" last_fre:"+laf)
148 print("frequency transmission_ep receiving_ep")
149 L.append("increase_frequency:"+data+" first_frequency:"+fre+" last_frequency:"+laf)
150 L.append("frequency transmission_ep receiving_ep")
151 print("send:"+fre+"kHz")

```

```
152 L.append(fre+"kHz")
153 L1=[]
154 L2=[]
155 Lmsave=[]
156 Lmsend=[]
157 Lmreceive=[]
158
159
160 if int(data)==0:
161     x=4
162     else:
163     x=2
164     t=1
165     else:
166     print("error")
167     v.set("")
168     u.set("")
169     s.set("")
170     v1.set("")
171     def stop_com(self):
172     global x
173     x=3
174
175
176 def connect(self):
177     self.start_connect()
178     send_button.configure(state=tk.NORMAL)
179     stop_button.configure(state=tk.NORMAL)
180     send_entry.configure(state=tk.NORMAL)
181     default_entry.configure(state=tk.NORMAL)
182     saveas_button.configure(state=tk.NORMAL)
183     max_entry.configure(state=tk.NORMAL)
184     time_entry.configure(state=tk.NORMAL)
185     connect_button.configure(state=tk.DISABLED)
186     saveas_combo.configure(state=tk.NORMAL)
187
188 def saveas():
189     global L
190     global data
191     global Lmreceive
192     global Lmsave
193     global Lmsend
194     secomb=vc.get()
195     if secomb=='all':
196     save(L)
197     else:
198     if data=='0':
199     print("error!!")
200     else:
201     if secomb=='sweep:fre-send-receive':
202     save(Lmsave)
203     elif secomb=='sweep:fre-send':
204     save(Lmsend)
205     elif secomb=='sweep:fre-receive':
206     save(Lmreceive)
207
208 def save(a):
209     filename=tkFileDialog.asksaveasfilename(defaultextension=".csv",filetypes=[("csv","*.csv*")])
210     with open(filename,'w') as fout:
211     fout.write("\n".join(a))
212
213
214 #周波数をclock_genelaterに送る
215 #ストップするときの関数
216 def stop_data():
217     global ser1
218     global ser2
219     global fre
```

```
220 ser1.write('b'.encode('ascii')) # arduinoへ終了の合図を送る。
221 ser2.write('b'.encode('ascii'))
222 ser1.flush() # バッファ内の待ちデータを送りきる。
223 ser2.flush()
224 ser1
225 print("--stop--")
226 L.append("stop")
227 fre='0'
228 time.sleep(1)
229
230 root=tk.Tk()
231 font=tkFont.Font(size=24)
232 ser=Ser()
233 v=tk.StringVar() # tk.TK()の後に書く。
234 u=tk.StringVar()
235 s=tk.StringVar()
236 v1=tk.StringVar()
237 vc=tk.StringVar()
238
239 # ボタン入力
240 connect_button=tk.Button(root, text='connect', font=font, height=2, padx=20, command=ser.connect)
241 connect_button.grid(row=0, column=0)
242 send_button=tk.Button(root, text='send', font=font, height=2, padx=20, command=ser.send_com)
243 send_button.grid(row=0, column=1)
244 send_button.configure(state=tk.DISABLED)
245 stop_button=tk.Button(root, text='stop', font=font, height=2, padx=20, command=ser.stop_com)
246 stop_button.grid(row=0, column=2)
247 stop_button.configure(state=tk.DISABLED)
248 #entry
249 send_entry=tk.Entry(root, font=font, textvariable=v)
250 send_entry.grid(row=1, column=1, columnspan=2)
251 send_entry.configure(state=tk.DISABLED)
252 default_entry=tk.Entry(root, font=font, textvariable=u)
253 default_entry.grid(row=2, column=1, columnspan=2)
254 default_entry.configure(state=tk.DISABLED)
255 max_entry=tk.Entry(root, font=font, textvariable=s)
256 max_entry.grid(row=3, column=1, columnspan=2)
257 max_entry.configure(state=tk.DISABLED)
258 time_entry=tk.Entry(root, font=font, textvariable=v1)
259 time_entry.grid(row=4, column=1, columnspan=2)
260 time_entry.configure(state=tk.DISABLED)
261
262 #label
263 label1=tk.Label(root, font=font, text='increase_frequency')
264 label1.grid(row=1, column=0)
265 label1_Hz=tk.Label(root, font=font, text='Hz')
266 label1_Hz.grid(row=1, column=3)
267 label2=tk.Label(root, font=font, text='first_frequency')
268 label2.grid(row=2, column=0)
269 label2_Hz=tk.Label(root, font=font, text='kHz')
270 label2_Hz.grid(row=2, column=3)
271 label3=tk.Label(root, font=font, text='last_frequency')
272 label3.grid(row=3, column=0)
273 label3_Hz=tk.Label(root, font=font, text='kHz')
274 label3_Hz.grid(row=3, column=3)
275 label4_time=tk.Label(root, font=font, text='Measurement_Time')
276 label4_second=tk.Label(root, font=font, text='ds')
277 label4_time.grid(row=4, column=0)
278 label4_second.grid(row=4, column=3)
279
280 #セーブボタン
281 saveas_button=tk.Button(root, text='save', font=font, height=2, padx=20, command=saveas)
282 saveas_button.grid(row=0, column=3)
283 saveas_button.configure(state=tk.DISABLED)
284
285 #COMBOBOX
286 Comb=['all', 'sweep:fre-send-receive', 'sweep:fre-send', 'sweep:fre-receive']
287 saveas_combo=ttk.Combobox(root, values=Comb, textvariable=vc)
```

```

288 vc.set(Comb[0])
289 saveas_combo.grid(row=0,column=4)
290 saveas_combo.configure(state=tk.DISABLED)
291
292 root.after(100,maindef)
293 root.mainloop()

```

ソースコード 2: 送電側 arduino

```

1
2 #include <si5351.h>
3 #include <Wire.h>
4 #include<MsTimer2.h>
5 Si5351 si5351;
6
7 unsigned long long freq = 50000000ULL;
8 /*出力周波数50kHz(これをいじって周波数を変える)freq×0.01=周波数Hz*/
9 unsigned long long pll_freq = 70500000000ULL; /*PLL周波数(いじるな)*/
10
11 String data;
12 float data0 = 0;
13 float f = 0;
14
15 void setup() {
16
17   Serial.begin(57600);
18   MsTimer2::set(100, flash);
19
20   bool i2c_found;
21   /*I2C通信ができるかどうかブール値を入れる変数*/
22   i2c_found = si5351.init(SI5351_CRYSTAL_LOAD_8PF, 0, 0);
23   /*I2C通信を確認(ライブラリreadme参照)*/
24   if (!i2c_found) {
25     Serial.println("Error:I2C");
26   }
27
28   si5351.init(SI5351_CRYSTAL_LOAD_8PF, 0, 0);
29   /*振動子負荷容量(使うモジュールが8pFなのでこれ)*/
30   si5351.set_freq_manual(freq, pll_freq, SI5351_CLK0);
31   /*出力周波数,PLL周波数,設定先出力ピン設定*/
32   si5351.set_phase(SI5351_CLK0, 0);
33   /*位相(今回特に意味はない)*/
34   si5351.pll_reset(SI5351_PLLA);
35   /*PLLをリセット(使う前に一回リセット)*/
36   si5351.update_status();
37   /*si5351のステータスを読む(今回特に使っていない)*/
38
39   while (Serial.available() == 0);
40 }
41
42
43
44
45 void flash(void) {
46   int i = analogRead(0);
47   f = i * 5.0 / 1023.0;
48   Serial.println(f,4);
49   //有効数字を4桁にする.
50   //シリアルモニターは初期設定では小数第二位までしか読み取れないから.
51 }
52
53 void loop() {
54   char aizu = Serial.read();
55   if (aizu == 'a') {
56     MsTimer2::stop(); //新しいduty比に変更されるまでflash関数を止める
57     aizu = 'c';
58     receive_duty_data();

```

```

59 MsTimer2::start();
60 }
61
62 else if (aizu == 'b') {
63   MsTimer2::stop();
64   si5351.set_freq(4000000, SI5351_CLK0);
65   /*信号を止める*/          /*!!!!!!set_freq(0)!!!これでは止まらない!!!!*/
66 }
67
68 else if (aizu == 'c') {
69   //pass
70 }
71 }
72
73 void receive_duty_data() {
74   data = Serial.readString();
75   data0 = data.toFloat();
76   unsigned long long freq = data0 * 1000000;
77   /*1=0.01Hzなので末尾に00をつける.入力単位をキロにしたいので末尾に10^3をつける.*/
78   si5351.set_freq(freq, SI5351_CLK0);          /*周波数セット*/
79   si5351.pll_reset(SI5351_PLLA);          /*念のためPLLをリセット*/
80   si5351.update_status();
81 }

```

ソースコード 3: 受電側 arduino

```

1  #include<MsTimer2.h>
2  float f = 0;
3  String data;
4  float data0 = 0;
5  void setup() {
6    Serial.begin(57600);
7    MsTimer2::set(100, flash);
8    while(Serial.available() == 0);
9  }
10 void flash(void){
11   int i = analogRead(0);
12   f = i * 5.0 / 1023.0;
13   Serial.println(f,4);
14   //有効数字を4桁にする.
15   //シリアルモニターは初期設定では小数第二位までしか読み取れないから.
16 }
17 void loop() {
18   char aizu = Serial.read();
19   if(aizu == 'a'){
20     MsTimer2::stop();
21     aizu = 'c';
22     receive_duty_data();
23     MsTimer2::start();
24   }
25   else if(aizu == 'b'){
26     MsTimer2::stop();
27     aizu='c';
28   }
29 }
30 }
31 void receive_duty_data() {
32   data = Serial.readString();
33   data0 = data.toFloat();
34 }
35 }

```