公立はこだて未来大学 2015 年度 システム情報科学実習
グループ報告書

Future University Hakodate 2015 System Information Science Practice

Group Report

プロジェクト名

感じる筋電義手の開発

Project Name

Development of myoelectric prosthetic hand with haptic sense

グループ名

回路計測班

Group Name

Circuit measurement group

プロジェクト番号/Project No.

9-B

プロジェクトリーダ/Project Leader

1013051　加藤大気　Taiki Kato

グループリーダ/Group Leader

1012061　平野連也　Ren-ya Hirano

グループメンバ/Group Member

1013051　加藤大気　Taiki Kato
1013226　山口弘貴　Hirotaka Yamaguchi
1013209　淺田幸哉　Koya Asada
1013218　佐藤聖太　Shota Sato
1013176　佐々木優　Suguru Sasaki

指導教員

櫻沢繁　伊藤精英　髙木清二

Advisor

Shigeru Sakurazawa　Kiyohide Ito　Seiji Takagi

提出日

2016 年 1 月 20 日

Date of Submission

Janualy 20, 2016

概要

筋電義手とは、筋活動時に近傍の皮膚表面に現れる電位信号を利用することによって、装着者の意のままに制御可能な義手である。しかし、一般的な筋電義手では触れた物の感覚をフィードバックすることができない。そのため暗い場所では義手がどのような状態にあるか確認できず、手すりをつかむ、壁伝いなどといった動作はできない。さらに、昨年度までのプロジェクトでは、義手の動作に必要な部品が義手筐体の外部にあった。義手を使用する度に外部の部品に接続しなければならない点は生活に支障を来たす可能性がある。

そこで、以上の二点を改善した義手を制作することをプロジェクトの目的とし、我々回路計測班では組み込み型の義手制御部の作成を目指した。中間発表までに微弱な筋電位を増幅する為のオペアンプをクアッド型としたことで昨年度までの回路と比較して小型化する事に成功した。また、より小さな部品を採用する事により更なる小型化を見込める事から最終発表ではプリント基板上に実装する事で義手内部に組み込むことを目指した。

さらに、サーボモータを利用して指を動かすにあたって、信号を平滑化せずにモータに入力すると意図しない動作を起こす可能性があるため、筋電信号を移動平均法により平滑化するためのプログラミングを行った。その際、外部からの実行命令がなくとも電源を入れるだけでプログラムが自動実行されるように記述し、動作に外部接続を必要としない義手制御部分の作成を目指した。

キーワード　筋電義手, 組み込み

（※文責: 平野連也）

Abstract

The robotic artificial arm is able to be controlled by user's will. The arm uses an electric potential appearing on the surface of skin when the muscles contract. But, users can ' t feel if their hands touch an object. So, they can ' t recognize their situation when they work in the dark place. It means they can ' t walk along a wall or hold a handrail in the dark place. And to control the last year 's hand, it was necessary to connect some parts out of the arm. It may cause a person to have trouble carrying out daily activities.
Therefore, we try to solve these two problems. By midterm, we succeeded in making the smaller EMG measurement circuit than last year's model by using a quad operational amplifier. Then, we try to mount these parts on printed circuit board for final term because it can be expected that the measurement circuit can be smaller.
In addition, to prevent malfunction of the servomotor witch control the finger, we programed for smooth the EMG signal by Moving-average method. And we enabled the program runs automatically when the switch turned on without any other instruction.

Keyword Myoelectric Hand, Built-in

（※文責: 平野連也）

目次

第 1 章　背景

1.1　現状・従来例

ここ数年で筋電義手の開発は急速に進められている。さらに、本物の人の手の動きを目指すことによって、筋電義手は作業用義手や能動義手に比べてより多く様々な作業を簡単に行うことができる。しかし、筋電義手は高価であるため、筋電義手の普及状況は、片腕前腕切断で義手を使用している者の中に占める割合で、アメリカは 25〜40%、ドイツは 70%、イタリアは 16% であるのに対し、日本では 2% にとどまっている [1]。また、筋電義手は触れた物の感覚をフィードバックすることができないため、本物の人の手の動きを目指すことにおいて、動きだけでなく感じる機能がある義手を世界のいくつかの国で研究されている。

（※文責: 佐藤聖太）

1.2　現状における問題点

現在、一般的な筋電義手にはいくつか技術的問題がある。それらの中で本プロジェクトでは「触れた物体の感覚をフィードバックすることができない」という問題に着目した。例えば、触れた物体の圧力を感じることできないと、物体の硬さが判断できないため、掴んだ物体を握り潰してしまう様な問題が起こる。さらに、昨年度までのプロジェクトでは、義手を制御するためにパソコンを経由していた。しかし、義手は人の手の代わりとして使用するのでその都度パソコン使わなければならないのは生活に支障が出る可能性がある。

このように、筋電義手は人の手で当たり前にできることができないため、本物の人の手の動きを目指すために改善すべき点はいくつかある。

（※文責: 佐藤聖太）

1.3　目標の概要

1.3.1　目標設定まで

筋電義手における問題解決のために、大きく分けて

1. 筋電義手の開発
2. 感じる機能の実装

という 2 つについてそれぞれ考えていくことにした。

1.3.2 目標

1.3.1 で分けた 1、2 についてそれぞれ目標を立てた。

1. パソコンを使わず義手を動かすために、筋電位計測回路とマイコンボードを組み込むことにした。そのために、昨年度のプロジェクトで制作されたものよりも小さな回路を作成・設計するとともに、組み込みに適した義手筐体の設計作成を目指した。さらに、小さい回路を作るために、基板をプリント基板で作成することを目標とした。

2.「触れた物の感覚をフィードバックすることができない」という問題について、人が手で感じられる様々な感覚の中から、義手の使用者が義手を装着した腕で物体に触れたことが知覚できない点と、義手で触れた物体の温度を装着者が知覚できない点から、「熱」と「圧力」についての 2 点に着目した。さらにこれらを義手に実装するために、フィードバック装置の設計・制作・実験を行うことにした。

回路計測班では、1 の目標のうち筋電義手の開発における回路の制作・設計について活動を行った。

（※文責: 佐藤聖太）

第 2 章　到達目標

2.1　問題の設定

1.2 の問題の解決のため、回路計測班では、以下を目標とした。

- 義手に組み込むために回路の小型化を実現する。
- 小型化を実現するため、基板をユニバーサル基板から、プリント基板に変更する。

（※文責: 佐藤聖太）

2.2　具体的な手順・課題設定

1. 手が動く仕組みを学ぶ
   課題解決のプロセス：筋収縮の仕組みを学ぶ。
2. 筋電位について学ぶ
   課題解決のプロセス：筋電位について学び、計測を体験する。
3. 目標の設定
   課題解決のプロセス：プロジェクトメンバーと話し合い、目指す筋電義手の仕様を決定する。
4. 計測班で必要になる知識を学ぶ
   課題解決のプロセス：筋電位の計測、信号処理に必要な知識を学ぶ。
5. 必要な回路と電極の作成、使用マイコンの検討
   課題解決のプロセス：屈筋と伸筋から電位をとるため、１チャンネルずつ合計２チャンネル分の回路の作成をした。オペアンプを組み込んだアクティブ電極の作成をした。
6. 筋電位の計測と処理を行う
   課題解決のプロセス：作成した回路・アクティブ電極を使用し、筋電位を測定して義手を動作させる。

（※文責: 佐藤聖太）

2.3 課題の割り当て

各人の得意分野及び関連性、時間軸のスケジュールを基準に以下のように割り当てた。

平野連也の担当課題は以下のとおりである。

5 月 筋電義手の仕組みについて学ぶため講習を受けた。
6 月 筋電位測定回路の設計製作を行い、電極及び筋電位測定回路の製作を行った。
7 月 中間発表での我々の発表内容をまとめたスライドを作成した。
8 月 eagle の仕様の学習
9 月 eagle 上で筋電回路の設計を開始
10 月 プリント基板の試作を開始
11 月 プリント基板が完成
12 月 最終発表に向け、発表内容をまとめたスライドを作成した。

淺田幸哉の担当課題は以下のとおりである。

5 月 筋電義手の仕組みについて学ぶため講習を受けた。
6 月 アクティブ電極と回路を作成をした。
7 月 中間発表のスライドと原稿を作成した。
8 月 成果発表に向けてこれからの活動の方針を決めた。
9 月 義手に使うバッテリーの検討と購入をした。
10 月 筋電位計測実験のデータをまとめた。
11 月 成果発表のスライドと原稿を作成した。
12 月 成果発表会でプレゼンをした。

佐々木優の担当課題は以下の通りである。

5 月 筋電義手の仕組みについて学ぶため講習を受けた。
6 月 移動平均やモーターの角度制御のためのプログラム作成。
7 月 実装するためのプログラムの修正およびデバッグ用のプログラムの作成。
8 月 実装に必要なアルゴリズムを学ぶ。
9 月 筋電位計測、解析用のプログラム作成。
10 月 プログラムの自動実行のスクリプト作成。
11 月 モーターの角度制御の調整。
12 月 他班の機能追加のためプログラムの修正。

佐藤聖太の担当課題は以下の通りである。

5 月　筋電義手の仕組みについて学ぶため講習を受けた。
6 月　boost.Python を用いたプログラムの作成。
7 月　ポスターの土台を作成した。
8 月　最終成果発表に向けてこれからの活動の方針を決めた。
9 月　触知覚班が使用する圧力センサの動作確認をした。
10 月　ポスターのアウトラインを作成した。
11 月　ポスターの作成を進めた。
12 月　メイン・サブポスターを完成させた。

山口弘貴の担当課題は以下のとおりである。

5 月　筋電義手の仕組みについて学ぶため講習を受けた。
6 月　電極及び筋電位測定回路の製作を行った。
7 月　中間発表での我々の発表内容をまとめたスライドを作成した。
8 月　中間発表までの成果を振り返り、最終発表までの目標を検討した。
9 月　義手に内蔵するのに適切なバッテリーなどのパーツの検討をした。
10 月　電極が追加で必要になったため、必要に応じて作成した。
11 月　小型化された筋電位計測回路を製作した。
12 月　メインポスター本文の英訳を行った。

加藤大気の担当課題は以下の通りである。

5 月　筋電義手の仕組みについて学ぶため講習を受けた。
6 月　電極及び筋電位測定回路の製作を行った。
7 月　中間発表での我々の発表内容をまとめたスライド、ポスターを作成した。
8 月　中間発表までの成果を振り返り、最終発表までの目標を検討した。
9 月　義手に内蔵するのに適切なバッテリーなどのパーツの検討をした。
10 月　電極が追加で必要になったため、必要に応じて作成した。
11 月　筋電位計測実験を行い、その結果を分析した。
12 月　スライド、ポスターの製作を行った。

（※文責: 加藤大気）

2.4 プロジェクトの活動内容

1. リーダーの選定
   プロジェクトを効率的に進めるためにプロジェクトリーダーを選定した。リーダーの役割をプロジェクト全体の進行役とし、話し合いの結果、加藤大気が担当することとなった。

2. 筋電義手についての学習
   筋電義手とはどういうものなのか、インターネットを用いて調べ問題点、従来の例などを調査した。

3. グループの編成
   効率的に作業を進めるため外装班、回路計測班、 触知覚班の３つのグループを編成した。

4. 筋電義手製作のために必要な知識の学習
   グループ毎に筋電義手製作に必要な知識を学習した。

5. 必要物資の調達
   義手を製作するのに必要な部品について調査し、購入した。

6. 義手の製作
   実際に義手を製作した。

7. 中間発表用スライド、ポスターの作成
   中間発表する際に使用するスライドやポスターを作成した。

8. 中間発表
   製作した義手、スライド、ポスターを用いて中間発表を行った。

9. 中間発表の反省
   後期の活動に活かすため前期の活動内容の反省点を全員で話し合い反省点を挙げた。後期の活動ではグループ間の情報共有を積極的に行い。スケジュール管理を徹底することとなった。

10. 最終発表に向けてプロジェクト全体での目標の確認
    最終発表に向けて後期の活動の目標を確認した。結果、改めて目標を感じる筋電義手の製作とし後期の活動に取り組むこととした。

11. 感じる筋電義手の製作
    目標より、圧力を装着者にフィードバックすることができ、部品の内蔵を可能とした義手を製作した。

12. 最終発表用スライド、ポスターの作成
    最終発表する際に使用するスライドやポスターを作成した。

13. 最終発表
    製作した義手、スライド、ポスターを用いて最終発表を行った。

14. プロジェクトの反省
    1 年間のプロジェクト活動の反省を行い、プロジェクト活動を行った意義をメンバー全員で確認した。

（※文責: 山口弘貴）

第 3 章　課題解決のプロセス（概要）

3.1　回路について

電気信号の中でも筋電位は微弱であり、計測するためには筋電位の増幅が必要である。筋電位を計測するためには筋電位計測回路が必要である。私達は筋電位計測回路を製作するためにこれを構成する各回路について学習した。

- 差動増幅回路
  2 つの信号の同相ノイズを除去し、差分を増幅する回路である。今回は 2 つの商用電源ノイズを除去するために用いた。

- ハイパスフィルター
  身体が動いた時に生じるモーションアーチファクトと呼ばれるノイズを減らすため、ハイパスフィルターを用いた。ハイパスフィルターは、特定の周波数以上の信号を通過させ、それ以下の周波数の信号をカットする働きを持つ。正確にフィルタリングするためには高次のフィルターが必要となるが、フィルターの次数を上げるためにはその次数分オペアンプが必要となる。今回は小型の回路製作のため、使用するオペアンプの数は抑えたかったため、1 次にとどめ、80Hz 以下の信号をカットした。

- 非反転増幅回路
  入力された電圧を増幅する回路で、微弱な筋電位の信号を増幅するために用いた。この回路から出力される電圧は入力された電圧と同位相となる。

- 積分回路
  入力電圧を時間で積分して出力する回路で、得られる積分波形は筋電位の大きさと筋活動量にほぼ比例している。ここで構成した積分回路は入力した電圧の符号を反転させる。負電圧のみの信号は、今回使用する Raspberry Pi 2 model B には入力不可能である。

- 反転増幅回路
  入力された電圧の符合を反転させ、増幅する回路。積分回路で負の値になった電圧を正の値にし、増幅するため用いた。

以上の回路を組み合わせて今回使用する筋電位計測回路を作成した。

（※文責: 加藤大気）

3.2　プリント基板とは

前期の活動では筋電位増幅回路を義手筐体に組み込むことができなかった。その反省から後期の活動では、同回路をプリント回路として制作し筐体への内蔵を図った。前期に使用した部品と同等の性能を有するより小型な表面実装部品で筋電位検出回路を構成する事で同回路の大幅な小型化を実現できると考えた為である。JIS 規格における関連用語の定義を以下に示す。

プリント基板　プリント配線と，プリント部品及び（又は）搭載部品とから構成される回路。
プリント配線　回路設計に基づいて，部品間を接続するために導体パターンを絶縁基板の表面又は表面とその内部に，プリントによって形成する配線又はその技術。

回路設計を行う CAD ソフトは様々存在するが公立はこだて未来大学（以下、本校と表現する）が所有する基板加工機では CAD ソフト Eagle が推奨されているため同ソフトを用いて設計を進めた。図？ に回路設計中の画面の一部を示す。

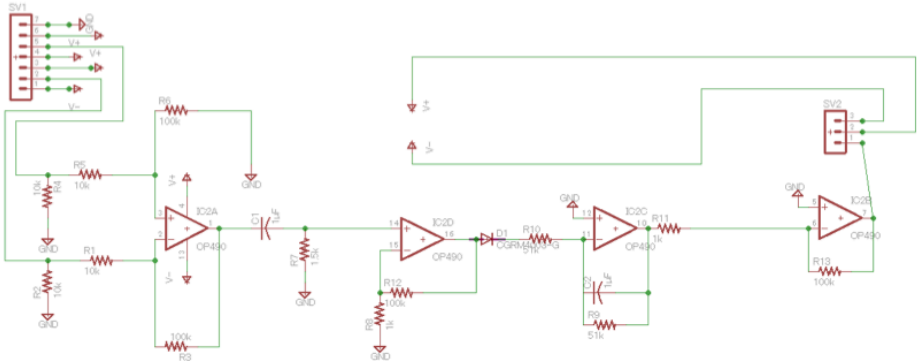

図 3.1　eagle による設計回路

また、図 3.1 に示す回路をソフト内にあらかじめ登録されている部品のパッケージ情報と照らし合わせ、部品の配置をレイアウトしたものを図 3.2 に示す。この時、組み込み時に配線が容易にできるよう入出力の端子を片側に集中させた。以上が CAD ソフト Eagle での作業である。

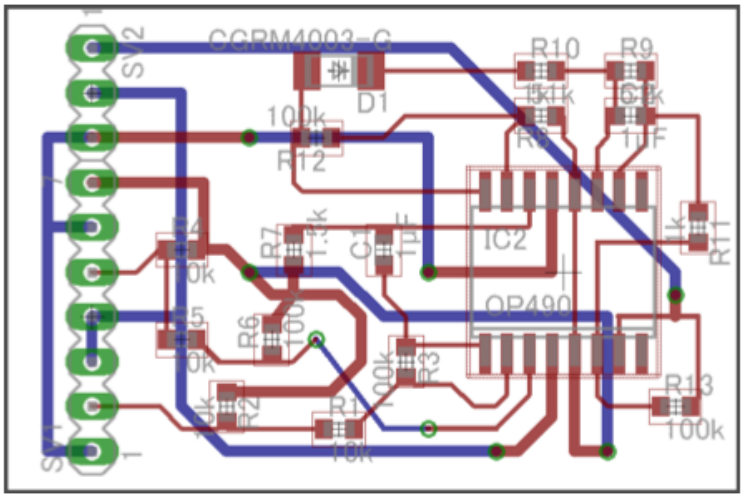

図 3.2　eagle による設計回路

次に、本校基板加工機に接続された PC 内のソフト Design Pro を使用した。生基板を加工するために図 3.2 で示した配線パターンを縁取りする形で加工線を追加し、プリント基板に加工した。生基板の素材には絶縁性と強度に優れるガラスエポキシ基板を採用した。加工した後、部品をはんだ付けしたプリント基板を図 3.3 に示す。

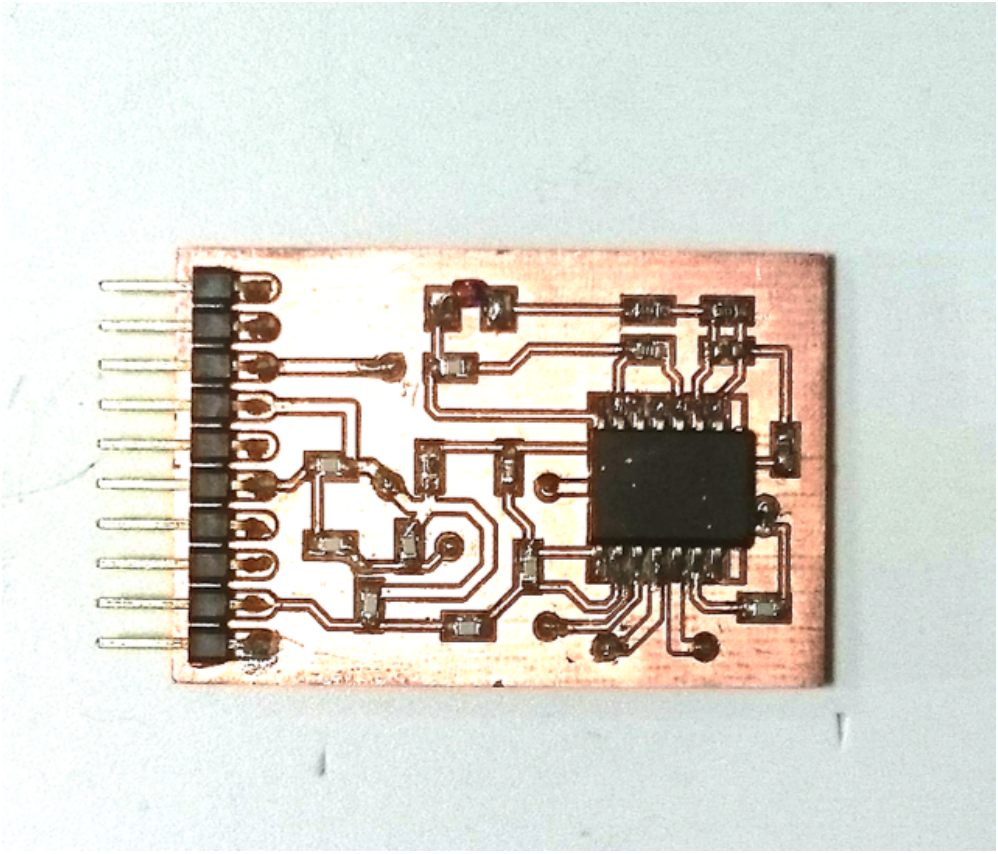

図 3.3　はんだ付けしたプリント基板

（※文責: 平野連也）

3.3　プログラムについて

プログラムでは、筋電位計測回路から入力された値を 0.05 秒間隔で確保し移動平均を取った。その後移動平均を取った値が設定した閾値よりも高い場合にモータを制御し、それぞれ屈曲、伸展の制御を行った。下に筋電位の入力及びサーボモーターへの出力を行っているプログラムを示す。

```
import sys
import time
import RPi.GPIO as GPIO
from Adafruit_PWM_Servo_Driver import PWM
from Vector import *

chnum = 8
bitrange = 12

v1 = Vector([0])
v2 = Vector([0])
```

```
limit = 800
vector_len = 20
flag = 1
t1 = 0
t2 = 0
pwx = 0
pwt = 4000
counter = 601

pwm = PWM(0x40)
servoMin = 300
servoMax = 600

fw = open(“hogehoge.csv","w")


def setServoPulse(channel, pulse):
    pulseLength = 1000000
    pulseLength /= 60
    print "%d us per period" % pulseLength
    pulseLength /= 4096
    print "%d us per bit" % pulseLength
    pulse *= 1000
    pulse /= pulseLength
    pwm.setPWM(channel, 0, pulse)

pwm.setPWMFreq(60)

pwm.setPWM(0, 0, servoMin)
pwm.setPWM(1, 0, servoMin)
pwm.setPWM(2, 0, servoMin)
pwm.setPWM(3, 0, servoMin)
pwm.setPWM(4, 0, servoMin)

spi_clk  = 11
spi_mosi = 10
spi_miso = 9
spi_ss   = 8


GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)


GPIO.setup(spi_mosi, GPIO.OUT)
GPIO.setup(spi_miso, GPIO.IN)
GPIO.setup(spi_clk,  GPIO.OUT)
GPIO.setup(spi_ss,   GPIO.OUT)
GPIO.setup(13, GPIO.OUT)
GPIO.setup(6, GPIO.OUT)
pwmV = GPIO.PWM(13, 10)
```

```
while True:
    time.sleep(0.05)
    counter += 1

    for ch in range(chnum):
        GPIO.output(spi_ss,   False)
        GPIO.output(spi_clk,  False)
        GPIO.output(spi_mosi, False)
        GPIO.output(spi_clk,  True)
        GPIO.output(spi_clk,  False)


        cmd = (ch | 0x18) << 3
        for i in range(5):
            if (cmd & 0x80):
                GPIO.output(spi_mosi, True)
            else:
                GPIO.output(spi_mosi, False)
            cmd <<= 1
            GPIO.output(spi_clk, True)
            GPIO.output(spi_clk, False)
        GPIO.output(spi_clk, True)
        GPIO.output(spi_clk, False)
        GPIO.output(spi_clk, True)
        GPIO.output(spi_clk, False)


        value = 0
        for i in range(bitrange):
            value <<= 1
            GPIO.output(spi_clk, True)
            if (GPIO.input(spi_miso)):
                value |= 0x1
            GPIO.output(spi_clk, False)


        if ch > 0:
            sys.stdout.write(" ")
        GPIO.output(spi_ss, True)

        sys.stdout.write(str(value))
        fw.write("%s"%value)

        if ch!= 7:
            fw.write(",")

        if ch == 0:
            v1.addVector(value)
            v1.control_len(vector_len)
        if ch == 1:
            v2.addVector(value)
            v2.control_len(vector_len)
```

```
        if v1.fuckingTF(limit):
            pwm.setAllPWM( 0, servoMax)
        if v2.fuckingTF(limit):
            pwm.setAllPWM( 0, servoMin)
        if ch == 7:
            if value <= 4000:

                GPIO.output(6, 0)
                if flag == 1:
                    pwmV.start(30)
                    pwt = 4000 - value
                    pwt = pwt / 100
                    t1 = t1 + 1
                    if t1 > pwt:
                        flag = 0
                if flag == 0:
                    pwmV.start(0)
                    t2 = t2 + 1
                    if t2 > value / 400:
                        flag = 1
                        t1 = 0
                        t2 = 0
            if value > 4000:
                pwmV.start(0)
                frag = 1
                t1 = 0
                t2 = 0
    fw.write("\n")
    sys.stdout.write("\n")
    if counter == 600:
        break

fw.close()
```

以上に示したプログラムは python2.7 で記述している。このプログラムを起動時にスクリプトとして実行し筋電位の計測を行っている [2]。次に上記のプログラムで使用している値を確保し移動平均をかけているプログラムを記す。

```
class Vector:

    def __init__(self, vector):
        self.vector = vector
        self.vector_len = len(vector)

    def addVector(self, vector):
        self.vector.append(vector)

    def control_len(self, length):
        if len(self.vector) > length:
            del self.vector[0:len(self.vector) - length]

    def fuckingTF(self, limit):
```

```
        ave = sum(self.vector)/len(self.vector)
        if ave > limit:
            return True
        return False

    def PressureTF(self, Plimit):
        ave = sum(self.vector)/len(self.vector)
        if ave > limit:
            return True
        return False
```

上記のプログラムで指定した長さの値を確保し、フィルターをかけることで平滑化を行っている。また、閾値を超える入力が行われたかの判定も行っている [2]。

```
#!/usr/bin/python
import re
import smbus

# ============================================================================
# Adafruit_I2C Class
# ============================================================================

class Adafruit_I2C(object):

  @staticmethod
  def getPiRevision():
    "Gets the version number of the Raspberry Pi board"
    # Revision list available at: http://elinux.org/RPi_HardwareHistory#Board_Revision_History
    try:
      with open('/proc/cpuinfo', 'r') as infile:
        for line in infile:
          # Match a line of the form "Revision : 0002" while ignoring extra
          # info in front of the revsion (like 1000 when the Pi was over-volted).
          match = re.match('Revision\s+:\s+.*(\w{4})$', line)
          if match and match.group(1) in ['0000', '0002', '0003']:
            # Return revision 1 if revision ends with 0000, 0002 or 0003.
            return 1
          elif match:
            # Assume revision 2 if revision ends with any other 4 chars.
            return 2
        # Couldn't find the revision, assume revision 0 like older code for compatibility.
        return 0
    except:
      return 0

  @staticmethod
  def getPiI2CBusNumber():
    # Gets the I2C bus number /dev/i2c#
    return 1 if Adafruit_I2C.getPiRevision() > 1 else 0

  def __init__(self, address, busnum=-1, debug=False):
    self.address = address
```

```
    # By default, the correct I2C bus is auto-detected using /proc/cpuinfo
    # Alternatively, you can hard-code the bus version below:
    # self.bus = smbus.SMBus(0); # Force I2C0 (early 256MB Pi's)
    # self.bus = smbus.SMBus(1); # Force I2C1 (512MB Pi's)
    self.bus = smbus.SMBus(busnum if busnum >= 0 else Adafruit_I2C.getPiI2CBusNumber())
    self.debug = debug

  def reverseByteOrder(self, data):
    "Reverses the byte order of an int (16-bit) or long (32-bit) value"
    # Courtesy Vishal Sapre
    byteCount = len(hex(data)[2:].replace('L','')[::2])
    val       = 0
    for i in range(byteCount):
      val    = (val << 8) | (data & 0xff)
      data >>= 8
    return val

  def errMsg(self):
    print "Error accessing 0x%02X: Check your I2C address" % self.address
    return -1

  def write8(self, reg, value):
    "Writes an 8-bit value to the specified register/address"
    try:
      self.bus.write_byte_data(self.address, reg, value)
      if self.debug:
        print "I2C: Wrote 0x%02X to register 0x%02X" % (value, reg)
    except IOError, err:
      return self.errMsg()

  def write16(self, reg, value):
    "Writes a 16-bit value to the specified register/address pair"
    try:
      self.bus.write_word_data(self.address, reg, value)
      if self.debug:
        print ("I2C: Wrote 0x%02X to register pair 0x%02X,0x%02X" %
         (value, reg, reg+1))
    except IOError, err:
      return self.errMsg()

  def writeRaw8(self, value):
    "Writes an 8-bit value on the bus"
    try:
      self.bus.write_byte(self.address, value)
      if self.debug:
        print "I2C: Wrote 0x%02X" % value
    except IOError, err:
      return self.errMsg()

  def writeList(self, reg, list):
    "Writes an array of bytes using I2C format"
    try:
```

```
      if self.debug:
        print "I2C: Writing list to register 0x%02X:" % reg
        print list
      self.bus.write_i2c_block_data(self.address, reg, list)
    except IOError, err:
      return self.errMsg()

  def readList(self, reg, length):
    "Read a list of bytes from the I2C device"
    try:
      results = self.bus.read_i2c_block_data(self.address, reg, length)
      if self.debug:
        print ("I2C: Device 0x%02X returned the following from reg 0x%02X" %
         (self.address, reg))
        print results
      return results
    except IOError, err:
      return self.errMsg()

  def readU8(self, reg):
    "Read an unsigned byte from the I2C device"
    try:
      result = self.bus.read_byte_data(self.address, reg)
      if self.debug:
        print ("I2C: Device 0x%02X returned 0x%02X from reg 0x%02X" %
         (self.address, result & 0xFF, reg))
      return result
    except IOError, err:
      return self.errMsg()

  def readS8(self, reg):
    "Reads a signed byte from the I2C device"
    try:
      result = self.bus.read_byte_data(self.address, reg)
      if result > 127: result -= 256
      if self.debug:
        print ("I2C: Device 0x%02X returned 0x%02X from reg 0x%02X" %
         (self.address, result & 0xFF, reg))
      return result
    except IOError, err:
      return self.errMsg()

  def readU16(self, reg, little_endian=True):
    "Reads an unsigned 16-bit value from the I2C device"
    try:
      result = self.bus.read_word_data(self.address,reg)
      # Swap bytes if using big endian because read_word_data assumes little
      # endian on ARM (little endian) systems.
      if not little_endian:
        result = ((result << 8) & 0xFF00) + (result >> 8)
      if (self.debug):
        print "I2C: Device 0x%02X returned 0x%04X from reg 0x%02X" % (self.address, result & 0xFFFF, reg
```

```
      return result
    except IOError, err:
      return self.errMsg()

  def readS16(self, reg, little_endian=True):
    "Reads a signed 16-bit value from the I2C device"
    try:
      result = self.readU16(reg,little_endian)
      if result > 32767: result -= 65536
      return result
    except IOError, err:
      return self.errMsg()

if __name__ == '__main__':
  try:
    bus = Adafruit_I2C(address=0)
    print "Default I2C bus is accessible"
  except:
    print "Error accessing default I2C bus"
```

上記のプログラムにより I2C 通信を制御している [2]。

```
#!/usr/bin/python

import time
import math
from Adafruit_I2C import Adafruit_I2C

# ===========================================================================
# Adafruit PCA9685 16-Channel PWM Servo Driver
# ===========================================================================

class PWM :
  # Registers/etc.
  __MODE1              = 0x00
  __MODE2              = 0x01
  __SUBADR1            = 0x02
  __SUBADR2            = 0x03
  __SUBADR3            = 0x04
  __PRESCALE           = 0xFE
  __LED0_ON_L          = 0x06
  __LED0_ON_H          = 0x07
  __LED0_OFF_L         = 0x08
  __LED0_OFF_H         = 0x09
  __ALL_LED_ON_L       = 0xFA
  __ALL_LED_ON_H       = 0xFB
  __ALL_LED_OFF_L      = 0xFC
  __ALL_LED_OFF_H      = 0xFD

  # Bits
  __RESTART            = 0x80
  __SLEEP              = 0x10
```

```
  __ALLCALL            = 0x01
  __INVRT              = 0x10
  __OUTDRV             = 0x04

  general_call_i2c = Adafruit_I2C(0x00)

  @classmethod
  def softwareReset(cls):
    "Sends a software reset (SWRST) command to all the servo drivers on the bus"
    cls.general_call_i2c.writeRaw8(0x06)        # SWRST

  def __init__(self, address=0x40, debug=False):
    self.i2c = Adafruit_I2C(address)
    self.i2c.debug = debug
    self.address = address
    self.debug = debug
    if (self.debug):
      print "Reseting PCA9685 MODE1 (without SLEEP) and MODE2"
    self.setAllPWM(0, 0)
    self.i2c.write8(self.__MODE2, self.__OUTDRV)
    self.i2c.write8(self.__MODE1, self.__ALLCALL)
    time.sleep(0.005)                                       # wait for oscillator

    mode1 = self.i2c.readU8(self.__MODE1)
    mode1 = mode1 & ~self.__SLEEP                 # wake up (reset sleep)
    self.i2c.write8(self.__MODE1, mode1)
    time.sleep(0.005)                             # wait for oscillator

  def setPWMFreq(self, freq):
    "Sets the PWM frequency"
    prescaleval = 25000000.0    # 25MHz
    prescaleval /= 4096.0       # 12-bit
    prescaleval /= float(freq)
    prescaleval -= 1.0
    if (self.debug):
      print "Setting PWM frequency to %d Hz" % freq
      print "Estimated pre-scale: %d" % prescaleval
    prescale = math.floor(prescaleval + 0.5)
    if (self.debug):
      print "Final pre-scale: %d" % prescale

    oldmode = self.i2c.readU8(self.__MODE1);
    newmode = (oldmode & 0x7F) | 0x10             # sleep
    self.i2c.write8(self.__MODE1, newmode)        # go to sleep
    self.i2c.write8(self.__PRESCALE, int(math.floor(prescale)))
    self.i2c.write8(self.__MODE1, oldmode)
    time.sleep(0.005)
    self.i2c.write8(self.__MODE1, oldmode | 0x80)

  def setPWM(self, channel, on, off):
    "Sets a single PWM channel"
    self.i2c.write8(self.__LED0_ON_L+4*channel, on & 0xFF)
```

```
    self.i2c.write8(self.__LED0_ON_H+4*channel, on >> 8)
    self.i2c.write8(self.__LED0_OFF_L+4*channel, off & 0xFF)
    self.i2c.write8(self.__LED0_OFF_H+4*channel, off >> 8)

  def setAllPWM(self, on, off):
    "Sets a all PWM channels"
    self.i2c.write8(self.__ALL_LED_ON_L, on & 0xFF)
    self.i2c.write8(self.__ALL_LED_ON_H, on >> 8)
    self.i2c.write8(self.__ALL_LED_OFF_L, off & 0xFF)
    self.i2c.write8(self.__ALL_LED_OFF_H, off >> 8)
```

サーボモーターの角度の制御をしている [2]。

（※文責: 佐々木優）

3.4 筋電位計測実験

回路班では製作した回路の性能実験や義手の動作確認の際、被験者の前腕に製作した電極を装着し、そこから筋電位の情報を得ている。この時毎回被験者毎に筋電位を読み取りやすい前腕部の部位を探すという行為を行っていた。この行為を被験者毎に行うのは手間であり、スムーズに実験を行うためには人間の前腕部のどの部分が電極の装着に適しているのかを調査する必要があると考え以下の実験を定義し行った。

実験目的　筋電位を計測する際人間の前腕のどの部分が電極の装着に適しているのかを調査する。

実験方法　20 歳から 23 歳の男性 10 人の右腕前腕に電極を装着した状態で手の伸展屈曲運動を行うよう教示し、筋電位が基準値を越えた回数と数値を記録した。数値はで 0 から 4095 の値を取り、基準値を実際にプログラム上で義手を動かす際の閾値である 300 とした。計測する区域（図 3.4）は手首から肘までの区域を 20% ずつに分割し、屈筋は腕の手のひら側、伸筋は手の甲側で計測を行った。

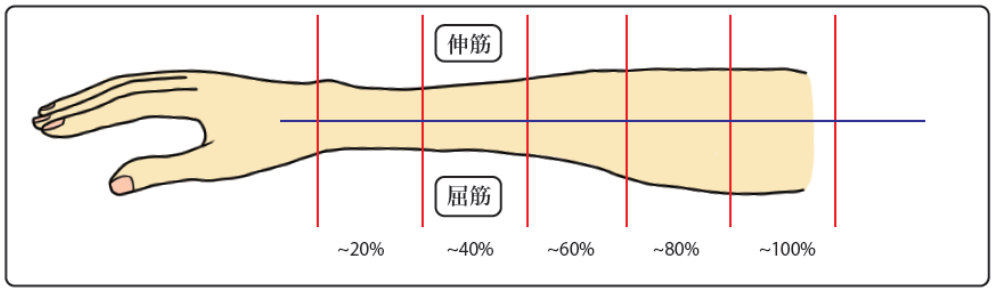

図 3.4　測定部分の区域分け

実験結果　基準値を超えた回数に着目した場合（図 3.5）、屈筋側は 40% 部分で最も回数が多く伸筋側は 60 80% 部分で回数が多かった。基準値を超えた時の値に着目した場合（図 3.6）、屈筋側は手首に近いほど筋電位の値は大きく、伸筋側は 40〜80 部分の範囲で大きい値を取り、手首に最も近い 20% 部分と肘に最も近い 100% 部分の値は小さい値となった。

考察　実験結果より、前腕から筋電位を計測する場合、屈筋側は 40% 部分に、伸筋側は 60 80% 部分に電極を装着することでより正確に筋電位を計測できることがわかった。これは人間が指を曲げる動作を行う際に動いている浅指屈筋、深指屈筋という筋肉がその部位に対応していてかつ他の筋肉に計測を妨げられずらい部位であったことを示していると考えられる。

（※文責: 加藤大気）

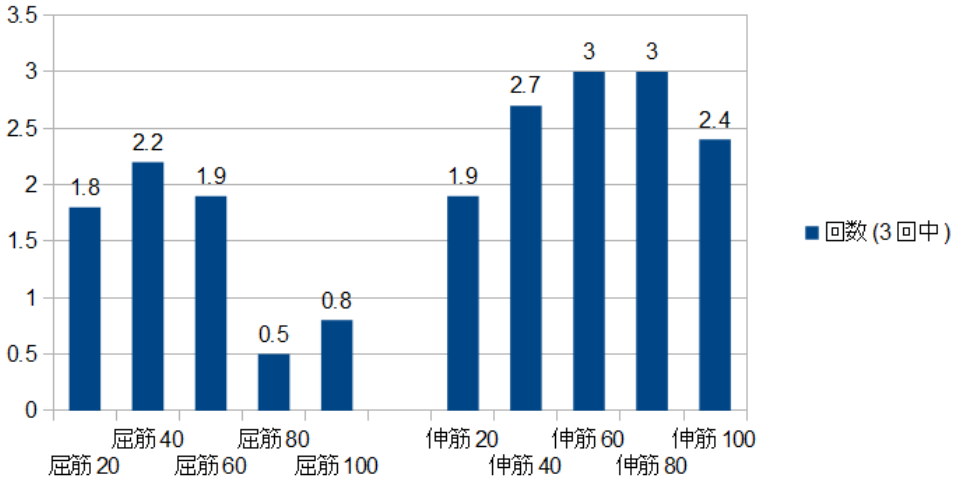

図 3.5　閾値を超えた回数

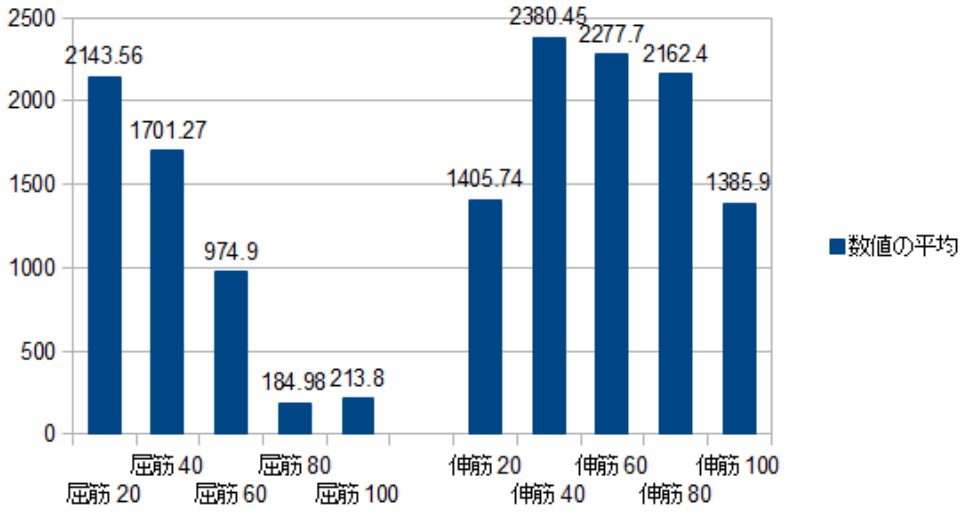

図 3.6　筋電位の平均

第 4 章　グループ内でのインターワーキング

4.1　スライドの作成

4.1.1　スライドの概要

成果発表会で発表するスライドでは, 中間発表のスライドで各班の役割を明確に区分してまとめた形式を変更した。「感じる筋電義手の開発」という目的を、「感じる機能をつける」と「筋電義手の開発」の二つの視点に分けて実現させていくコンセプトでスライドを作成した。回路計測班では主に「筋電義手の開発」の部分で回路の制作過程や制作した回路の特徴をまとめた。

4.1.2　回路計測班で担当したスライド

- 筋電位計測回路
  制作した筋電位計測回路がどのような役割を持っているのか説明した。
- 回路の制作過程
  回路の制作過程を写真を使って 2 ページで説明した。
- 完成した回路
  プロジェクト活動の前期と後期で制作した回路と、2 つの回路の大きさを比較するために 500 円玉の写真を載せた。義手に使う部品をすべて内蔵するために、前期で制作した回路よりもさらに小さく回路を制作したことを示した。

4.2　成果発表までの成果

中間発表までに実現できなかった義手筐体への回路の内蔵を、ユニバーサル基板からプリント基板に変更し小型化することで達成することができた。また、義手が単体で動作できるようにプログラムを変更した。

（※文責: 浅田幸哉）

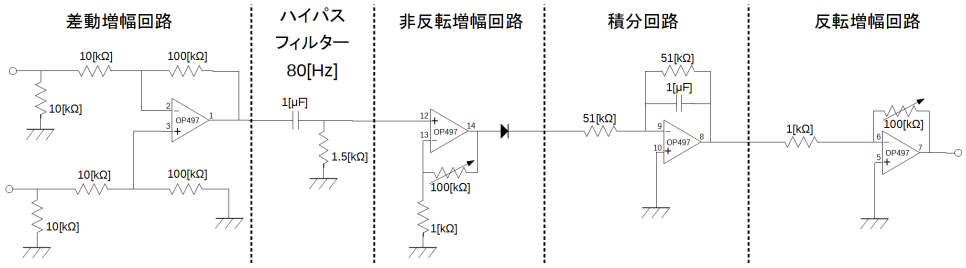

図 4.1　製作した筋電位計測回路

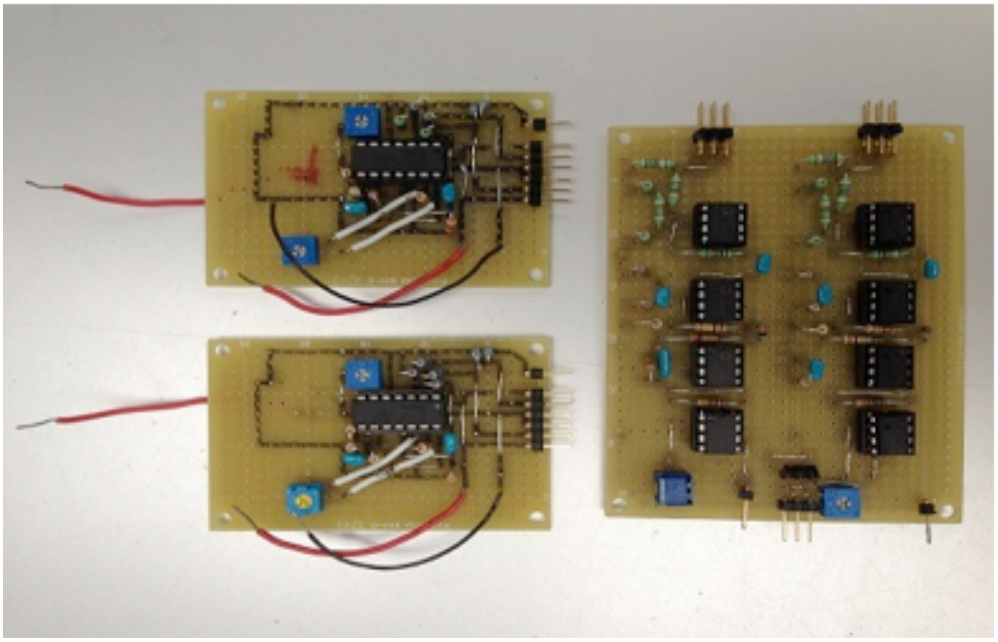

図 4.2　今回製作した回路（左側）と前年度の回路（右側）

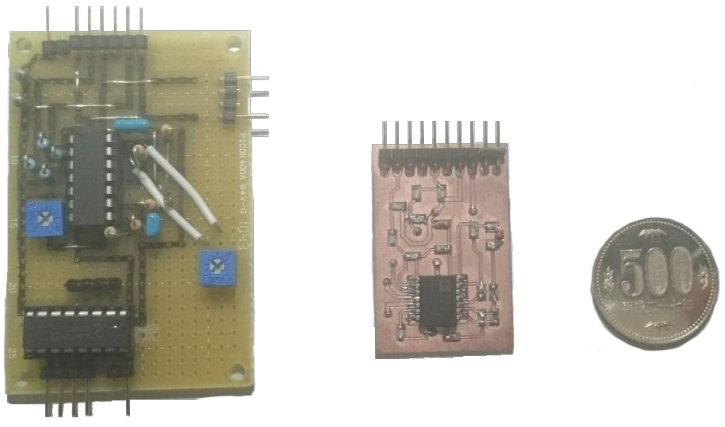

図 4.3　ユニバーサル基板、プリント基板、五百円玉の比較

第 5 章　結果

5.1　成果

今年度の我々の目標は、義手本体への組み込みを考慮した筋電位測定回路の小型化であった。中間報告の時点では、ユニバーサル基板を使用して回路を製作したが、それでもなお大きく、組み込みには至らなかった。それを踏まえて、後期は基板加工機を使用して生基板上に非常に密に回路を設計することで、前期に製作したものの更に半分以上小さくすることができ、義手本体に組み込むことができた。

また、前年度は義手の制御を外部で行っていたが、今年度は制御を Raspberry Pi 上で行い、それを義手に内蔵することで、義手の動作を行うすべてのパーツを組み込むことに成功した。

（※文責: 山口弘貴）

5.2　作業手順と評価

我々回路計測班は、筋電位測定回路を設計し基板上に作成する者と、その回路を利用して筋電位を Raspberry Pi に取り込み、義手を制御するプログラムを作成する者の二手に分かれて作業を行った。前者は、筋電位測定回路の理論を学んだ上で、筋電位を取得するための電極や回路の設計・制作を行った。後者は、筋電位測定回路を通して得られた筋電位を利用し、義手の動作を制御する Python プログラムを作成した。

筋電位の取り込みは、A/D 変換器を経由することで問題なく行うことができた。義手の制御の状況も良好だった。

（※文責: 山口弘貴）

第 6 章　今後の課題と展望

6.1　プロジェクトの成果

前年度までは、筋電位計測回路は大きすぎて義手に組み込むことはできず、義手の制御も外部接続したパソコン上で行っていた。

今年度は、プリント基板を使用して回路を小型化し、義手の制御も Raspberry Pi 上で行うことによって、完全に組み込むことに成功した。

6.2　プロジェクトにおける各人の役割

6.2.1　佐々木

主に Raspberry Pi 上で実行する筋電位計測や、義手の動作を制御するためのプログラムを作成し、また手の動作を再現するモーターの制御を行った。

6.2.2　佐藤

プログラム製作の補助と、前後期のプロジェクトのポスター作成を行った。

6.2.3　平野

CAD ソフトウェアを用いて、筋電位計測回路のレイアウトを考案し、設計を行った。

また、筋電位はアナログ信号であるが、Raspberry Pi はデジタル信号しか取り込むことができないため、A/D 変換する方法を検討し、実装した。

6.2.4　加藤

必要に応じて電極の追加製作を行った。筋電位計測実験を行いその結果を分析した。

6.2.5　淺田

必要となった場合に備えて電極の追加製作を行った。また、筋電位計測実験を行いデータをまとめた。

6.2.6　山口

前後期通じて、筋電位を取得するための電極や、筋電位計測回路の製作を行った。また、後期では義手の動作に必要な Raspberry Pi などの機器の電源となるバッテリーや電池ボックスの選定も行った。発表時のポスターに掲載する日本語文の英訳も行った。

6.3　成果発表フィードバックについて

我々は一年間のプロジェクト活動をもとにポスター、スライドなどの発表資料を制作し、成果発表会で発表を行った。その際に聴衆の方々に発表内容の評価をしていただいた。以下に良かった点と悪かった点の一覧を示す。

良かった点

- デモムービー、実演があったのと目標がわかりやすかった
- 問題設定、目標からの解決までがしっかりしていた
- 素人でも義手内部構造がわかったので、わかりやすい内容だった
- 実験から今後の課題を見つけていて素晴らしいと思いました
- 一年前にできていなかった内蔵に挑戦していた
- 実験で気付けたことを活かせるような研究をしている
- 実験の内容やその様子がわかりやすかった
- 見やすい動画で説明してくれてわかりやすかった
- 前期に作った基盤を小型化するために組み直すのはすごい気合を感じた
- 実験内容、反省などがわかりやすくまとめられてた
- 課題設定からの結果の流れが良かった
- 展望がしっかりしており、計画は十分だとおもった
- 製作課程は細かくわかりやすかった
- 知識のないでも部品の内容がよく理解できた
- 義手の制作に高度な技術が入っている
- 声が大きくて、資料がわかりやすかった
- 質疑応答にもしっかり答えることができていた
- 動画がわかりやすかった
- 義手の大切さ、尊さを学ぶことができた
- 成果物わかりやすかった
- 振動子で気付かせるという着眼点は面白い
- メカとしての工夫はうまく伝わった
- 比較対象を設けており、わかりやすかった

悪かった点

- スライド一枚の情報が少ない
- パーツの説明が一行しかなく、パーツごとの関係が少しわかりにくかった
- スライドの文字をすこし大きくしてほしい
- 目的をもう少し話してほしい
- 実用段階になく、中間からの進展がなかった
- 物をつかめているわけではない
- 研究開始時の目標設定をもっとしっかりすべき
- 背景薄オレンジに白抜き文字は見にくい
- 専門的な説明で少しわかりづらかった
- 情報技術的なアプローチも必要そう
- フィードバック方法を考えましょう
- 物を握る感覚と振動の整合性、他にも方法がありそう
- 目的がおかしかった
- 実用化にはセンサー等の改良が必要である
- 精度向上の課題がある
- 温度から圧力へ変更理由を明確にしてほしい

以上のことから、良かった点と悪かった点についてまとめた。

良かった点のまとめ　実演やビデオ等の発表資料は高評価を受けた。義手のパーツ内蔵と振動でフィードバックする方法はユニークで面白いという意見があった。

悪かった点のまとめ　作成した義手の動作、特にフィードバックの精度についてはまだまだ改善の余地があるという意見があった。振動よりも分かりやすいフィードバック方法があるのではないかと指摘された。

以上のまとめから実演とビデオの内容、フィードバックの発想については高評価が得られたが、今後の課題としてフィードバック精度の向上とその方法については他の方法を調査・検討し、装着を前提とした実用段階に入っていく必要があることも分かった。

（※文責: 淺田幸哉）

6.4　今後の課題

筋電義手の屈曲、伸展の制御に成功し、ものを掴んでいる感覚のフィードバックにも成功したが、物を掴むという動作の制御では十分な成果を得られなかった。そのため指の角度をどのように制御するか、どのような形で指を曲げることにより物を掴むことができるかの手法を考えていく必要がある。そのために信号処理、制御だけでなく触知覚班と連携して、研究および開発を今後の課題として行っていく必要がある。

今年度は、義手本体に筋電位計測回路や制御用のマイコン、モーター、バッテリー、電池などを全て組み込むことができた。しかし、パーツを組み込むことだけを考えて実際に使用するときのことはあまり考えられていなかったため、義手の重さがおよそ 2 キログラムと非常に重くなってしまった。

軽量化の上で一番の足かせとなっているのは、おそらくバッテリーと電池だと思われる。これらを 1 つのバッテリーあるいは電池にまとめることができなかった理由は、Raspberry Pi や触知覚班で使用する機材は電圧が違うため、それぞれに適した電源を用意する必要があるためである。

（※文責: 山口弘貴）

参考文献

[1] 陳隆明(ちんたかあき). 義手の可能性 ―従来の義手と筋電義手―. http://medicalfinder.jp/doi/abs/10.11477/mf.6002100483.

[2] adafruit-16-channel-pwm-servo. https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi/library-reference