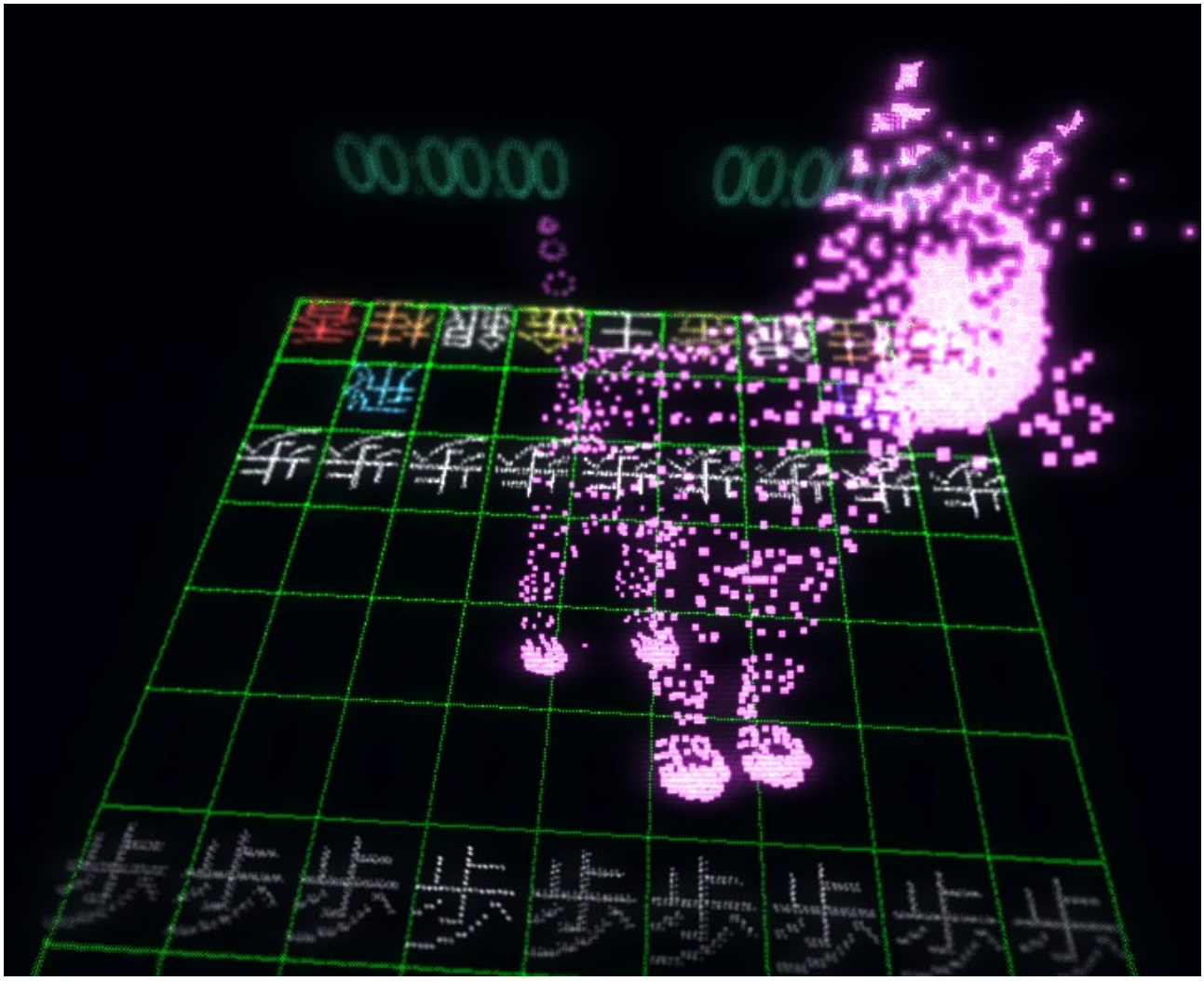
自主プロジェクトレポート

「AR ボードゲーム」



160308 森田雅博

テーマ

今回のプロジェクトでは、将棋盤を真上からwebカメラで撮影し、画像処理することで盤面をデジタル化し、ARコンテンツに取り入れることを目指しました。一番の技術的テーマは画像処理ですが、複数のスレッド、プロセスをたて、それらを運用するための通信が裏のテーマになっています。

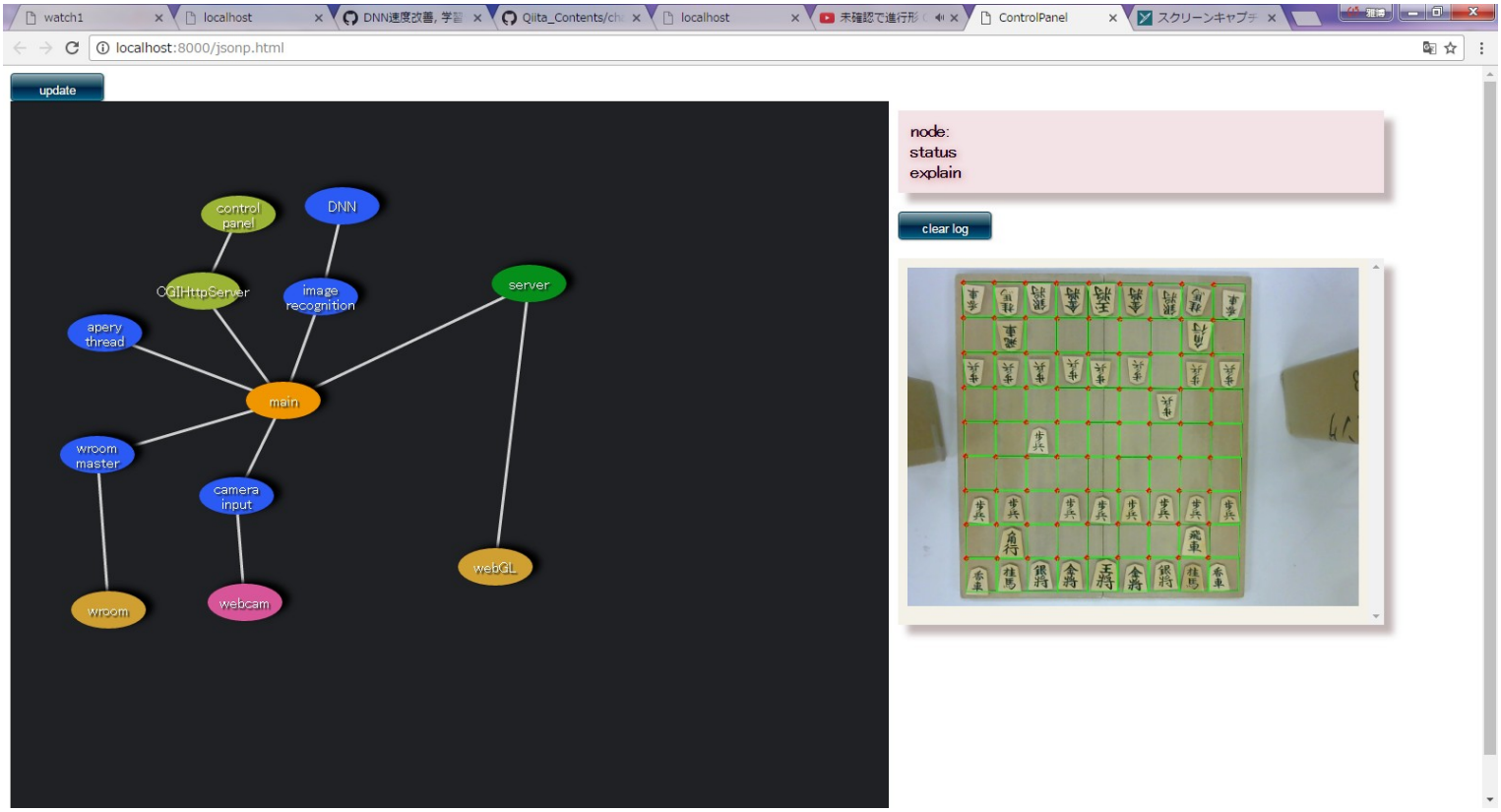
機能としては、人間が将棋盤で一手進めると、画像処理で指し手を判定します。指し手の情報はクラウド上のサーバに送られ、各端末(PC, スマホなど)からサーバにアクセスすることでリアルタイムに盤面を見られるようにしました。こちらの画面はWEBGLのjsライブラリであるThree.jsを利用しました。システムの状態の入出力は、もともとpythonの標準入出力を利用していましたが、見やすくするため、GUIツールとしてのコントロールパネルを作成しました。また、オープンソースになっている将棋ソフトのaperyを組み込み、機械の判断する次の正しい手を取得できるようにしています。さらにwifi通信でローカルPCと通信する独自デバイスを製作しました。

この独自デバイスを利用することで対局中に一方の対局者が相手の対局者に気づかれることなくaperyの次の一手を取得することを想定しています。

概要

主な処理はpythonでかいています。GUIのコントロールパネルを作成するにあたって、pythonのみで実装するのは困難であると判断し、GUIはhtmlとしwebコンテンツにしました。以下がそのキャプチャになります。左の部分はノードとそれらの通信路を表しています。ノードは、ひとつひとつのプロセスまたはスレッドを表しています。右側は左側でクリックして選択したノードの状態や出力が表示されます。

ローカルPC上の実装(主にpython)はhttps://github.com/moritanian/ShogiRecognition/ に、クラウドサーバの実装は https://github.com/moritanian/ShogiRecognitionServer にあげており、リポジトリを分けています。



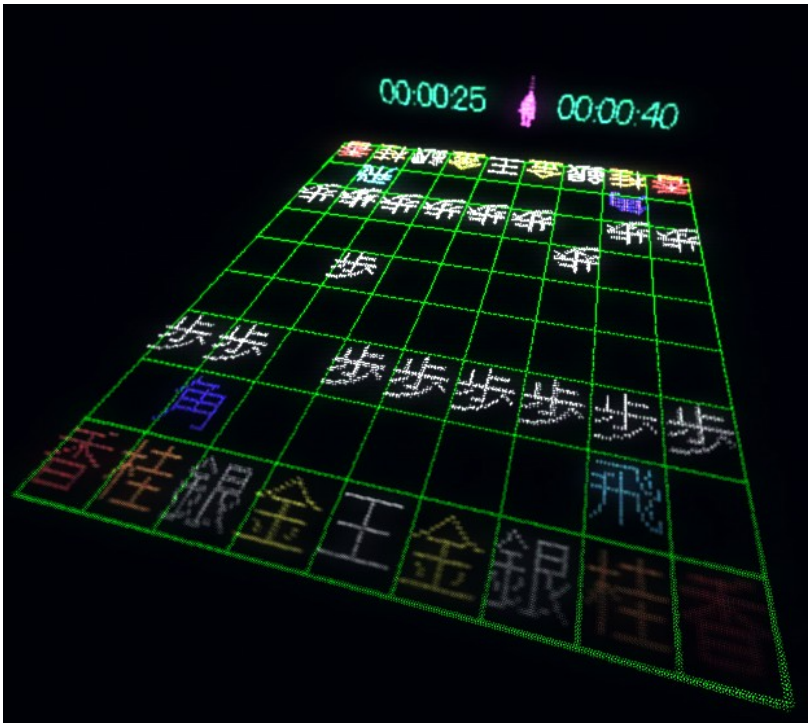
処理の流れとしては、「webcam」で盤面を撮影し、「camera input」で決められた時間ごとに画像として取得します。「image recognition」で画像処理しますが、この際、場合によって機械学習を利用します。これらは image_e.py, koma_recognition.py に実装しています。画像処理によって盤面が進んだ場合、その情報を「server」に蓄えます。サーバとの通信は http 通信になります。実装には python の urllib ライブラリを使用しました。(WWW.py)

「webGL」はサーバにアクセスする web クライアントを指します。サーバをクラウドにおくことで、www につながるあらゆるデバイスからアクセスできるようになります。右下画像が表示例です。

先ほどのコントロールパネルは「Control Panel」に対応しており、「CGIHttpServer」を介してメインと通信しています。CGIHttpServer は CGI に対応した python の簡易サーバです。これら関係ファイルはまとめて、/server 以下においています。

「apery thread」では apery の実行ファイルを立ち上げ、apery と通信をします。(apery_call.py)

「wroom」は、独自デバイスを表し、「wroom master」と wifi 通信します。(wroom_master.py)



実装方法

主な部分の実装方法について記述します。

「Image Recognition」

画像処理では、「画像中の将棋盤の位置の特定」、「将棋盤のマス目（9＊9）に分割」、「各マス目に駒があるかどうかの二値判定」、マス目に駒があった場合「先手か後手（見方の駒か敵の駒かは駒の向きで決まる）かの判別」、「駒の種類の判定」と、順番に行います。

「将棋盤の位置の特定」では、マス目の直線をハフ変換を用いて抽出し、盤の縦横それぞれ 10 この直線を取り出します。

「駒の有無の判定」は、マス目の領域に分割した画像を canny のエッジ抽出し、エッジの多さで識別しました。適切に閾値を設けることでほぼ100%の識別率を達成できました。

「駒の先後の判定」は、マス目の領域に分割した画像をランダムハフ変換によって、駒の輪郭の直線を抽出します。斜めのラインの傾きの方向とその位置から判定します。ここまでの処理結果が以下になります。片側しか取り出せていないものもありますが、その位置（中心に対して左右いずれか）と傾きから先後を判定できました。

「先後の判定」までは画像処理で対応できますが、「駒の種類の判定」は、文字認識となり、機械学習を使用しますが、将棋は40枚の駒があり、それを実用レベルで正しく認識するだけの認識率を達成するのは技術的に厳しいと判断しました。実際は将棋のルールの制約で、正しい手は限られるため、「駒の先後判定」の段階まででほとんど対応できます。この観点で将棋の有効手を分類すると以下の3種類になります。

1, 駒の移動、駒とりなし

2, 駒の移動、駒とりあり

3, 駒打ち

1は手の進む前と後で駒がなくなるマス一箇所、出現するマス一箇所なので、「駒の有無の識別」の段階までで移動元と移動先のマスを特定できます。

2は手の進む前と後で駒がなくなるマス一箇所、先後が入れ替わるマス1箇所で、前者が移動元のマスで後者が移動先のマスになります。よって「先後の識別」までで移動元と移動先のマスを特定できます。

1,2 の駒の移動は、移動先で成る可能性がある場合はさらに成りの判定として「駒の種類の識別」が必要になります。

3 は手の進む前後で駒が出現するマス一箇所です。持駒の種類が複数ある場合、どの駒を打ったのか識別するために「駒の種類識別」が必要になります。

以上をまとめると、「駒の種類識別」の段階まで、すなわち機械学習が必要なのは、

a, 成る可能性のある駒移動

b, 持駒の種類が複数ある場合の駒打ち

の 2 とおりの場合のみとなります。今回の実装ではこの二通りの場合以外は「駒の先後判定」の段階までで指した手を判定するようにしています。

ただ、このままでは手やその他外乱を駒があると誤識別し、手が進んだと認識してしまう場合がたまに発生しました。そこで前の画像と現在の画像の差分をとり、その差分画像を領域分割、ラベリングし領域の数と大きさで手などが入っていると判定した場合、そこで処理を終了するようにしました。この二重チェックをもって外乱対策としました。

「DNN」

機械学習には 3 層の DNN を利用しました。python のライブラリである chainer を用いました。

```
# 多層パーセプトロンの定義
model = FunctionSet(  l1=F.Linear( input_size, input_size),
                      l2=F.Linear(input_size, output_size))
```

学習アルゴリズムとして adam を選択し、初期配置画像15枚ほどで学習させた結果、認識率（未学習データ）は9割ほどでした。前段に畳み込みとプーリングの層を入れてみましたが、認識率の改善はみられませんでした。そのため、仮に盤面判定をすべて機械学習でやると駒は40枚あるので毎回平均で4マス分は誤識別します。前述のように機械学習を利用する回数を最低限にすることで実用に近いレベルにしました。

「CGIHttpServer」

前述のとおり、web コントロールパネル用に python でローカルサーバを立てています。server/cgi-bin/panel.py, server/cgi-bin/socket_slave.py に記述しています。このサーバと python のメインの部分は完全に別プロセスになっておりこの間は TCP/IP のソケット通信で繋ぐようにしました。さらに python サーバとコントロールパネルは http で非同期通信することでデータを常に更新しています。この部分は jquery を用いて ajax 通信で実装しています。通信するデータ形式は可変なので JSONP でやりとりしました。

「Control Panel」

コントロールパネルは、システムの状態の見える化にこだわりました。各ノードは、対応するシステムからの応答がなくなるとグレーアウトし、特定のグレーアウトしたノードはダブルクリックで立ち上がるようにしています。また、システム操作とは関係がありませんが、ノードをドラッグすることで自由に移動させることができるようにしています。

これらの実装は都合のよいライブラリが見つからないので、jquery を使って独自に実装しています。また、ノードの影がノードの周りを回ったり、ノードの説明タブがノードクリック時に点滅スライドするなどの演出をとりいれていますが、これらは処理を軽くするため、css で対応しました。スタイルシートだけでもかなりこみいった演出ができることを学べました。

「Server」

盤面データを蓄積するサーバは、php で記述しました。サーバのルーティングの学習の意味を含めて既存のフレームワークを利用せず、独自のフレームワーク(もどき)を構成しています。ここではルーティング、MVC の分離、データベース接続の最低限の仕組みを実装しています。

このサーバをローカル pc におくと、ローカルネットワーク内でしか接続できません。一般の WEB アプリケーションのようにどこからでもアクセスできるようにするため、外部サーバにおくようにしました。今回は Heroku を利用しました。URL は、https://shogirecognitionserver.herokuapp.com で現在もアクセスできるようになっています。

また、さまざまなアプリケーションサービスからデータを使用できるように、データのやりとりは api 形式にしています。以下の例のように json を返すようにしています。

- api/game/[game_id] # 指定した game_id の対局情報を取得

{"result":true,"list":
[{"id":"2","first_player_name":"","second_player_name":"","description":"","epoch":"10","winner":"0","start_time":"2017-01-06 17:23:30","end_time":null}]}

- api/game/[game_id]/game_record/show/[手数] # 指定した手数の指し手の情報を取得

{"result":true,"records":
[{"id":"32","game_id":"2","epoch":"1","position":"6757","target":"1","is_promotion":"0","revival":"0","kihu":"?26?\n","create_time":"2017-01-06 17:23:42"}]}

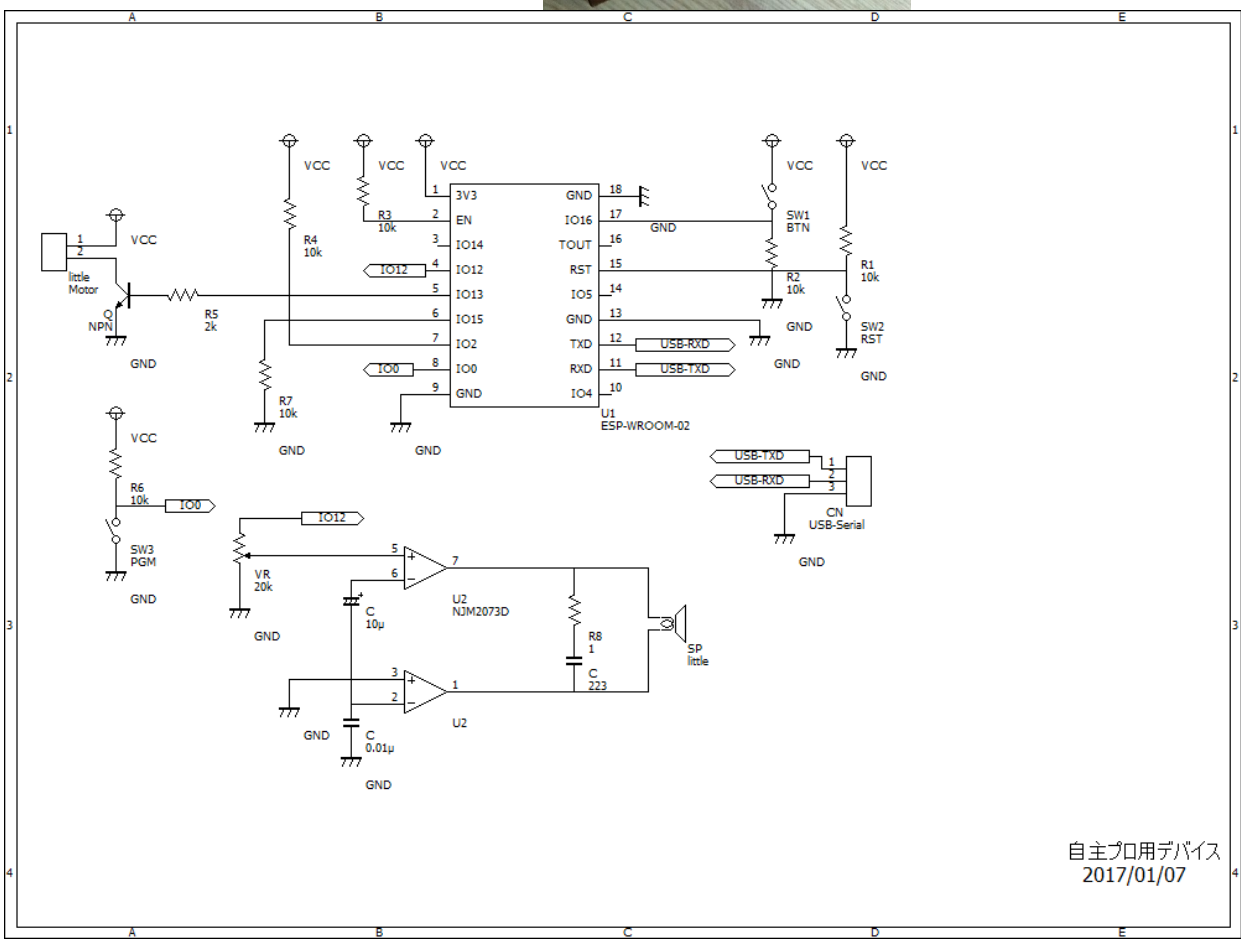
web クライアントとしてアクセスした場合の例は概要のほうで提示しましたが、webGL を用いています。表示しているすべてのオブジェクトはパーティクルで構成しています。three.js ライブラリのポストエフェクトを用い、ブラーなどをかけることでパーティクルが演出できることがわかりました。コードはライブラリの公式サンプルの https://threejs.org/examples/#webgl_points_dynamic を参考にしました。

また、ユーザエージェントによって PC とスマホでページを分けることで、スマホからアクセスした場合、スマホの傾きによって画面内のオブジェクトが回転するようにしました。姿勢取得によく用いられるのは加速度センサですが、今回はスマホに搭載されている3軸の磁気センサの値を使用しました。地磁気の方向から姿勢を取得します。加速度センサを用いて重力加速度の向きから姿勢を取得する方法に対して、磁気センサを使う方法は並進加速度の影響を受けないというメリットがあります。まわりの電化製品などの磁気による影響を受けるかと考えていましたが、ほとんど影響はみられませんでした。他に、スマホページは負荷軽減のため、PCページに比べて描画するパーティクルの数を減らしています。

「Wroom」

独自デバイスは wifi モジュール、スピーカ、振動モータ、スイッチで構成しています。wifi モジュールは、arduino ライクなプログラムが可能で、TCP/IP スタックを積んだ技適取得済みのモジュールである ESP-WROOM02 を使用しています。概観と中身、全体の回路図は以下です。

ESP-wroom02 は arduino 環境で開発することができるので、プログラムのコンパイル、書き込みは arduino と同じように簡単に行えました。スペックは、CPUクロックが 80Mhz, フラッシュメモリ 4M で RAM が 36KB と arduino uno（ATMEGA328P）よりも性能は上になります。



wifi モジュールの動作としては、wifi のアクセスポインタに接続し、その後ローカルPCに接続し TCP/IP 通信を行います。コードは　wroom/wroom_mypwm.c　になります。ローカルPCのIPアドレスは変化するため、wifi モジュール側に IP アドレスを設定する必要がありました。そこで、設定時は PC と独自デバイスを有線でつなぎ、UART 通信で IP アドレスを設定するようにしました。設定した IP アドレスは eeprom に記録し電源を切っても同じ IP アドレスに接続するように対応しました。

機能としては、

1、扇子で盤面のマスを指し示し、独自デバイスのスイッチを押すと、その指したマスが apery の次の一手の移動先と一致した際に振動モータが振動し、対局相手にばれずに正解手を入手できる機能

2、単にスイッチを押すとスピーカからの音で次の一手を知らせる機能

の二つを仕込みました。

1に関しては盤のみの画像と扇子の写った画像の差分画像を領域分割、ラベリングし、領域の大きさから扇子部分を識別し指し示すマスを取得します。

2に関しては音声データの通信に苦労しました。apery の出力する次の一手を棋譜にし、(例:「先手3三角成る」)それを音声合成 WebAPI を用いて音声データに変換します。今回は VoiceText WebAPI (https://cloud.voicetext.jp/webapi/docs/introduction) というサービスを利用しました。

wav 形式の音声データを wifi 通信で独自デバイス側に送ります。独自デバイス側は二つの buffer を用意し、一方に送られたデータを書き込み、もう一方の buffer から24 khz の周期で読み出し pwm 出力します。一方の bffer を最後まで読み終わると役割を逆にし、読み込んでいた buffer に受信データを書き込み、受信データを書き込んでいた buffer からデータを読みこみ pwm 出力します。 これらを実装し、動作確認すると途中で watch dog timer が作動する等でリセットがかかったり、バッファーの書き込みと読み込みの反転の周期と一致すると思われるノイズが聞こえました。バッファーのサイズを小さくしたり、電源の安定化など対策をしましたが、解決できませんでした。原因としては、wifi モジュールのスペック的に難しいのではないかと考えています。

結果

自主プロ開始時に構想していた機能はほぼすべて対応することができました。1点、対応できなかった機能が駒のリアルタイム学習です。現在の実装では駒の種類識別は事前学習した結果を利用しているため、異なる駒や環境下で実行するためには学習しなおす必要があります。そこで本番中に、機械学習による駒の識別が必要になる盤面が現れるまで(機械学習を使用せず画像処理のみで盤面を判定できる状況において)カメラ画像を教師データとして1から学習し、機械学習による識別が必要になった際に学習結果を利用するという構想を立てていました。が、chainer による機械学習を導入したのが発表の2日前で、そこまで対応しきれませんでした。

自主プロの発表を終えて感じたのは、技術的に詰めることよりひと目でインパクトがでるようにすることのほうが重要であるということです。発表時間が短いこともあり、うまく自分の製作物をアピールできなかったと感じています。今後は人に見せるということを意識してものづくりにとりくみたいです。

また、裏のテーマとして通信を扱いましたが、TCP\IP, プロセスとスレッドの違いなどの理解が深まりました。

今回の経験をもとに、今後は WebRTC を用いて複数デバイス間のリアルタイムなアプリケーションの構築をやりたいと考えています。