Foosball Robot

03-190294 村上 恵太郎

01/29/2021

1　概要

Foosball(通称テーブルサッカー) の対戦相手となるロボットを制作した．Web カメラで読み込んだ画像からボールの位置を認識し，その結果に基づいてモータを制御してバーを動かしている．以下にデモ動画を作成した．

`https://youtu.be/knL-YN4Qc_c`

また，完成したソースコードは以下のリポジトリに置いてある．

`https://github.com/ketaro-m/foosball_robot`

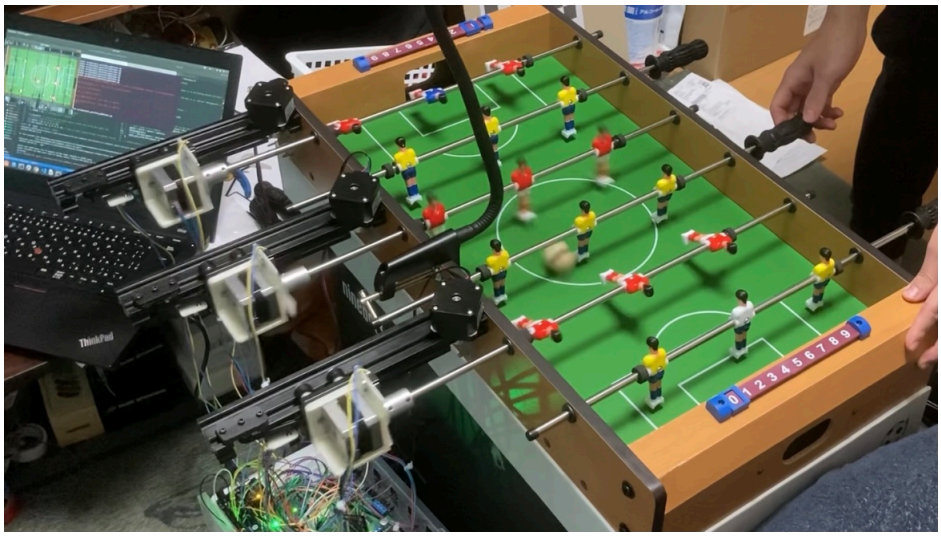

図 1: 作品の全体像

2　実行環境

2.1　ハードウェア

ハードウェアは補遺 A に記載した部品表のものを使用した．

2.2　ソフトウェア

- **Ubuntu 18.04**
- **ROS Melodic Morenia**

ROS packages

- **uvc_camera** (`http://wiki.ros.org/uvc_camera`)
- **camera_calibration** (`http://wiki.ros.org/camera_calibration`)
- **image_proc** (`http://wiki.ros.org/image_proc`)
- **cv_bridge** (`http://wiki.ros.org/cv_bridge`)
- **rosserial** (`http://wiki.ros.org/rosserial`)
- **rqt_image_view** (`http://wiki.ros.org/rqt_image_view`)

3　動作原理

台の真上に設置した Web カメラで盤面の画像を読み込み，色抽出をベースとした画像処理を施すことによってボール位置を認識し，その結果を Arduino が受け取って 6 つ (3 つのバーそれぞれにつき並進と回転) のステッピングモータを制御することによってボールを打ち返す．これを 10 fps のサイクルで繰り返すことによって，ボールに素早く追従する．
この全体像を ROS ノードの構成図で表すと以下のようになる．

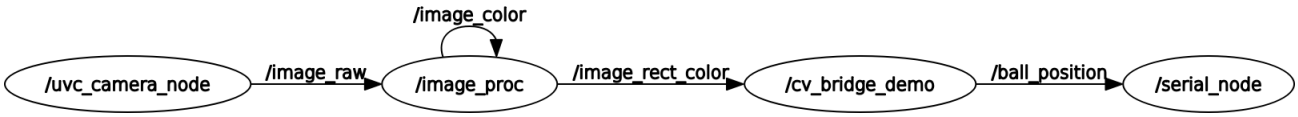

図 2: ROS ノード

4　動作結果

まず，L6470_SPI_stepMoter_sketch.ino を Arduino に書き込む．次に，以下にあるように launch/stepper_camera_driver.launch を実行してカメラノードなどの複数の ROS ノードを立ち上げた後，最後に scripts/ball_position_publisher.py を実行して画像処理ノードを立ち上げると，図 2 にあるようなパイプラインが繋がって全体が動くようになる．

```
$ roslaunch launch/stepper_camera_driver.launch
$ python scripts/ball_position_publisher.py
```

5　制作過程

5.1　ハードウェア設計

既製品の Foosball 台を使用し，それに合うように部品を選定して組み合わせた．リニアモジュールとステッピングモータを接続する部分，ゼロ点 (モータ制御のセクションで説明する) 用のマイクロスイッチを取り付ける部分は 3D プリンタを用いて作成した．

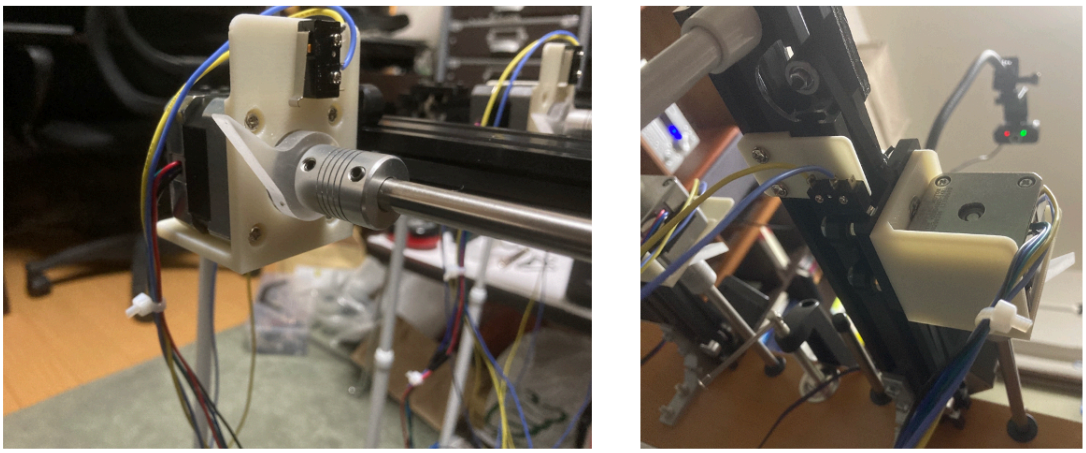

図 3: 3D プリンタで作成した部品

5.2 画像処理

5.2.1 カメラキャリブレーション

Web カメラから直接入ってくる画像にはレンズ歪みがあるので，これを補正する内部パラメータを求める必要がある．このためのプログラムは用意されているため，以下のサイトに従ってチェッカーボードをカメラの前で動かすことで，画像のように補正することができる．詳細な手順は，README.md(`https://github.com/ketaro-m/foosball_robot/blob/main/README.md`) に記載している．

- **camera_calibration** (`http://wiki.ros.org/camera_calibration`)
- **How to Calibrate a Monocular Camera** (`http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration`)

図 4: キャリブレーション結果

5.2.2 ボール検出

ボールの検出には HSV カラーフィルタを用いる．
画像処理の一連の流れは以下の画像の通りで，まず初めにフィールド内 (画像内でのフィールドのピクセルは事前に設定しておく) に HSV フィルターをかけ，特定の HSV 値の領域を抽出する．次に，メジアンフィルタ，膨張・収縮処理を施すことによって，ノイズ除去とボール領域のクロージングを行う．最後に画像中の最大領域を取得することによって，ボールの領域のみを検出する．

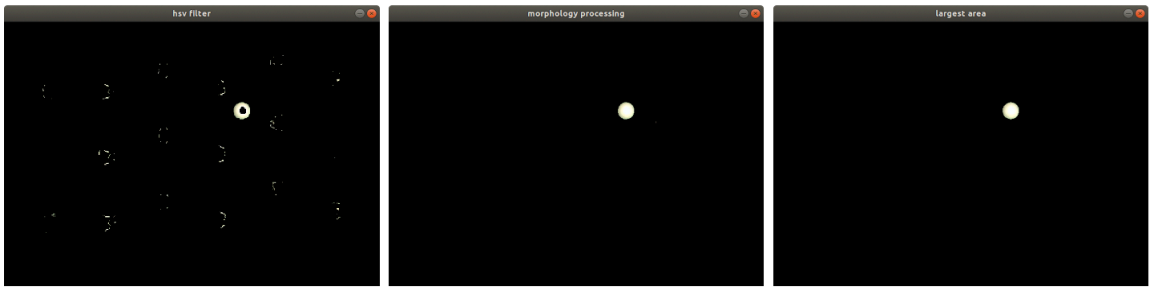

図 5: 画像処理

ボールの HSV 値は適切に設定しておく必要がある．しかし，環境によって光の当たり具合が若干異なり適切な HSV 値が変わってしまうため，このチューニング用のプログラムも作成した．
カメラノードを立ち上げた状態で script/hsv.py を実行すると画像が表示され，ボールをクリックするとそのピクセルの HSV 値が得られる．このプロセスを何回か行ったり，ボール内の様々なピクセルを選択することで，適切

な HSV 値の領域をチューニングする．
この値を scripts/ball_position_publisher.py のプログラム上部に設定して以下のようにプログラムを立ち上げると，画像のように適切にボール位置を検出することができていることが分かる．また，このプログラムはその座標を ROS トピック (/ball_position (opencv_apps/Circle 型)) として publish する役割も担っている．
こちらも，詳細な手順は，README.md(https://github.com/ketaro-m/foosball_robot/blob/main/README.md) に記載している．

```
$ roslaunch launch/track_ball.launch
$ python scripts/ball_position_publisher.py
$ rostopic echo /ball_position
```

図 6: 検出結果

5.3 モータ制御

ステッピングモータの制御には，L6470 というチップの乗ったモータドライバを使用した．SPI 通信という方式が取られ，8bit ごとにデータを区切って送信することで，データシート (https://www.st.com/resource/en/datasheet/l6470.pdf) に載っている様々なコマンドを送って制御できる．これを実現する Arduino スケッチのライブラリは，このブログ (http://spinelify.blog.fc2.com/blog-entry-41.html) に載っているものを使用した．
複数のモータを制御するためには，モータドライバをデイジーチェイン接続する必要がある．デイジーチェイン接続とは，図 7 にあるように (このブログ (https://garchiving.com/control-multiple-motors-with-arduino/) から引用した)，複数 byte を順々に押し出してそれぞれのモータドライバに所望の byte 値を書き込む方式である．このデイジーチェイン接続を考慮した複数モータの制御ライブラリは存在しなかったため，上述のライブラリを参考にしながら，全てのライブラリを書き直す作業を要した．これにより，デイジーチェインという直列の通信方法であっても，Arduino スケッチのメインプログラム内では，特定のモータのみを特定のコマンドで駆動するような形式でプログラムできるようにした．
今回の作品ではモータをデフォルト値よりもかなり高速に動かす必要があった．これはモータドライバのあるレジスタ値を設定することで実現できるのだが，その値に伴って電圧値も適切に設定する必要があった．また，モータを速く動かしすぎると脱調を起こしてしまう．これを防ぐためにも，各種レジスタ値をチューニングし，加速度やその立ち上がりの角度を適切に設定する作業が必要であった．この適切な値はモータの数を増やす度に若干変更されるため，その都度チューニングする必要があった．
どうしても脱調を起こしてしまうことは防ぎきれず自己位置がずれてしまう，また，ボールがプレイヤーに強く当たるとモータが逆方向に回転してずれてしまう．これを補正するためには，図 3 にあるようにマイクロスイッチによるゼロ点を用意した．原理としては，脱調が検出されたら，そのモータを脱調復帰モードに切り替えてマイクロ

図 7: デイジーチェイン接続

スイッチが押される位置まで動かし，押された瞬間にモータドライバの位置レジスタの値を所定の値にリセットするというメカニズムである．また，脱調検出がされなくても，スイッチが押される位置にモータが来たときは位置をリセットし，自己位置がずれてしまったが脱調検出がなされなかった場合にも定期的に修正するようにした．脱調とは，モータが機械的に入力パルスに追従できず，ストール (失速) する現象である．これは，モータの起電力という形で電気的に検出でき，今回使用したモータドライバにはエラー検知用のレジスタという形で組み込まれていた．よって，これらを Arduino の入力ピンに結線して監視することで，脱調検出を可能とした．この検出判定の閾値などのパラメータは，何回か試してみて決定する必要があった．

Arduino スケッチの概要

- L6470_SPI_stepMoter_sketch.ino ← メインプログラム
- L6470_commands.ino ← モータコマンドライブラリ
- L6470_commands_multi.ino ← モータコマンドライブラリ (デイジーチェイン対応)
- foosball_commands.ino ← ゲーム操作コマンドのライブラリ

メインプログラム (L6470_SPI_stepMoter_sketch.ino) は，同じディレクトリ内にあるライブラリをインポートする形で利用している．まずセットアップでは，各種ピンの設定，SPI 通信の立ち上げ，ROS ノードの立ち上げなどを行い，メインループの中で恒常的に脱調検出ピン，ゼロ点ピンの監視を行っている．さらに，画像処理ノードによって publish されたボール位置を subscribe しており，その結果に基づいたコマンド操作をして，バーを操縦している．
バー操縦のコマンドは比較的シンプルで，まず，図 6 にあるようにフィールドを分割することで，ボールが相手側にあるか自分側にあるかを判別し，相手側にあった場合はディフェンス，自分側にあった場合はオフェンス動作をする．ディフェンスの場合は，横方向はボール位置に追従するようにスライドし，ボールよりも後ろにいるプレイヤーは足でボールを止められるようにバーを回転する．ボールよりも前のプレイヤーは攻撃に備えて足をあげておく．オフェンスの場合は，ボール位置にいるバーはキックモーションを行い，それよりも後ろはディフェンスと同じ要領で防御の構えをし，それよりも前にいるバーは後ろから来るボールの通路を空けるように構える．基本的にはこのようなコマンドは全てモータの回転角を指定することで実現でき，ボール位置の publish と同じ周波数，つまり 10 fps でモータの回転角を上書きしていくようなサイクルで指示を出していくが，キックモーションは 1 回のサイクルよりも長いサイクルを要する．この場合や，脱調検出がされたことによって脱調復帰モードになっているモータには指示がいかないように，特定のモータにだけコマンド指示が送られるように工夫した．具体的には，グローバル変数として 6bit の変数を用意し，各 bit にそれぞれのモータの状態フラグを表現して，その情報も含めてモータに指示が送られるように工夫した．各モータフラグを配列として表現するとポインタの受け渡しなどが煩雑になったが，このように int 型の変数を 2 進数と見なして受け渡すことで，プログラムを簡潔にしたり，メモリの節約をすることができた．

6　苦労したこと，反省，感想

スターリングエンジン設計演習を通して機械設計や加工は経験したが，ソフトウェアや電子部品も含めて一人で設計する経験はほとんど初めてだった．自分で加工した部品は少ないが，データシートを正しく読み，適切な部品を選定する過程は非常にためになった．
ハードウェアを含んだものだと，理論上は正しく動くはずでも実機では動作しないということが多々起こり，またそのデバッグも大変だった．例えば，モータを複数に増やすと電圧・電流が足りなくなって動作をしなくなる，Arduino Nano ではメモリが少なく，ROS ノードを立てるような処理を加えるとメモリが圧迫されて動作が不安定になる，といったバグは発見するまでに時間を要した．しかし，理論を学んだだけでは気づけないようなバグを身を持って体験できたのは貴重な経験だったと思う．
この制作過程で最も大変だったのは，複数のモータを適切に制御することだった．特に，今回は非常に素早く動かす必要があったので，モータの特性から考えて各種のパラメータを調整する過程が大変だった．また，複数のモータにした途端に動かなくなったり，計算のオーバーフローが起きていたり，突き詰めていけば必ずどこかに特定の原因があるものだが，それを見つけるまでに多くの時間を費やしてしまった．しかしながら，こうしたエラーは体系的に学べるものではなく，このように逐一経験していくことでしか学べないので，今後の糧になると思う．
当初の題目は「FC DQN」であったように，目標は強化学習を用いてモータ制御も学習させることだったが，上述のモータ制御に時間がかかってそこまで辿り着くことはできなかった．モータ制御のセクションで説明したようなルールベースのプログラムでも対戦相手としては楽しめるものになったし，1ヶ月強の制作期間で作るものとしては及第点だと思うが，せっかくなら学習にも着手したかった．この辺りは，制作のスケジュールの問題であり，反省して次に活かせるポイントである．例えば，部品の発注は到着するまでに時間がかかるから早めに済ますなどの優先順位をつけることはもちろん，期限までに小さなマイルストーンを置き，小さなステップに分解しながら進めることで進捗管理やプラン B への切り替え判断を迅速にするなどの工夫も必要だと感じた．この辺りは，エンジニアリングの世界では重要なことであると思うので，今後に生かしていきたい．

お願い

総じて自主プロは非常に良いプログラムだと思いますが，費用の全てが学生の自己負担であることは，この演習の唯一の欠点だと思います．僕が上述のように経験できたデバッグ作業は，ハードウェアを積極的に使うことで得られるものであるし，僕もその観点から費用を捻出することにためらいはなかっのびですが，金銭的な問題から制作物を制限してしまう学生も少なからずいるのではないかと思います．少額であっても学生の学びの質を高めることになり，自主プロがより良い演習になると思うので，制作費用の支援の検討をしていただければ後輩のためになると思います．

謝辞

オンラインという難しい状況の中，多くの先生方のご協力のもと，この作品を完成させることができました．
コンセプト段階から相談に乗っていただけた岡田先生，モータ制御や機械設計についてアドバイスをくださった浅野先生，3D プリンタの印刷を手伝っていただいた西川先生には特に感謝を申し上げます．
その他，相談に乗ってくださった先生方，学科同期の皆さん，ありがとうございました．

補遺 A　部品表

以下に使用した部品の一覧を記載する．(部品名にハイパーリンクを付与している．)

部品名	個数
Foosball 台	1
木球	1
ThinkPad (学科 PC)	1
Arduino Mega	1
AC アダプタ (24V 2.7A)	1
DC ジャック端子変換	1
ブレッドボード，各種ジャンパピン	
ステッピングモータリニアモジュール	3
ステッピングモータ 42SHD4002-24B	3
L6470 ステッピングモータドライバ	6
M3×6 ねじ (ステッピングモータ固定用)	12
カップリング	3
ブラケット	3
M5 六角ナット, ワッシャー	3
M5×20 皿木ネジ (*Foosball* 台とリニアモジュールの接続用)	3
M5×8 皿頭小ねじ (リニアモジュールと 3*D* プリンタ部品の接続用)	12
マイクロスイッチ	6
M2×12 ねじ，ナット (マイクロスイッチ固定用)	12
M4×10 ねじ (マイクロスイッチ台座固定用)	6
M3×6 いもねじ (ステッピングモータ軸と 3*D* プリンタ部品の固定用)	6
USB カメラ	1
カメラスタンド	1