最終チーム報告書最終チーム報告

16班

1. 実験目的
2. 実験環境
3. ライントレース
   1. ハードウェア(記述者--魚躬)

　　設計の目的：

　チームの戦略としてコーナーで他のチームとの差を付けようと考えた。そのために，センサーからの値を確実に習得しその値をすぐにタイヤへ反映できるようにすること，どんなカーブでも素早く曲がることができるようにすることを目的にハードウェアの開発をした。

　　実際のハードウェアの仕様：

　作成したハードウェアを図1に示す。また，図からは読み取れない仕様を表1に示す。

|  |  |
| --- | --- |
| 重量 | 600g |
| 部品数 | 52個 |
| センサーとタイヤ間の距離 | 10㎜ |
| センサーの地面からの距離 | 9㎜ |
| センサー間の距離 | 32㎜ |
| タイヤ間の距離 | 145㎜ |

　表1：細かいハードウェアの仕様

　　工夫点：

　工夫した点は重量・センサーの位置・タイヤの位置・センサーとタイヤ同士の位置・全体のバランスの5点である。

　重量は部品の数をサンプルハードウェアの110個から52個に減らしたことによって745gから600gまで軽量化することができた。この軽量化を実現できたのはハードウェア全体の安定性を支える重要な部品にのみ丈夫で重い部品を扱い，それ以外の部分には極力部品を付けない，または軽い部品を使うように工夫したからである。

　センサーの位置について説明する。まずセンサーはライトセンサーとカラーセンサーの2つがあり，この2つの距離をサンプルの40㎜から32㎜に近づけることで車線に合わせて常にグレーの値が良い撮れるように工夫した。また，センサーの地面からの高さをサンプルの14㎜から9㎜にしたことでよりセンサーの値を確実に取得し，PID制御がしやすくなるようにした。

　タイヤの位置は，タイヤ同士の距離を145㎜に近づけることで曲がるときによりインコースで曲がることができるように工夫した。

　センサーとタイヤ同士の位置は，センサーとタイヤの距離を10㎜に近づけたことでセンサーから読み取った値をすぐにモーターに反映させるｋとができるように工夫した。

　全体のバランスは図1-(a)を見ると分かるようにハードウェアの後方にボール付きのピンをつけたことで全体の重心をタイヤの近くにすることができた。これによって走行中にハードウェアが滑ってしまうことを防ぎ，より線に沿った走行が行えるようになっている。

　　問題点と改善策：

　ハードウェア自体は当初の予定通りに100％作成できたのでハードウェア自体には問題点はない。しかし，ハードウェアの作成作業の進め方には問題点があった。それはソフトウェアとハードウェアで作業を分担して進行せずにプログラムの作業が終わった後にハードウェアの作業に取り掛かろうとしていたことである。そのため，ソフトウェアの作業中に問題が発生した際にセンサーやモーター自体に原因があることに気づかず，プログラムの改良作業に時間をかけてしまい多くの時間が無駄になったことがあった。このような問題点を以降起こさないようにするために，もの集め競技会に向けた作業ではハードウェアとソフトウェアの作業を分担して，ハードウェアとソフトウェアのどちらに問題が発生しているのかにいち早く気づけるように改善した。

* 1. ソフトウェア
     + - ライントレース

ライントレースの方は頼んだ...



* + - * ペナルティ競技

　以下にペナルティ競技のコースを示す.

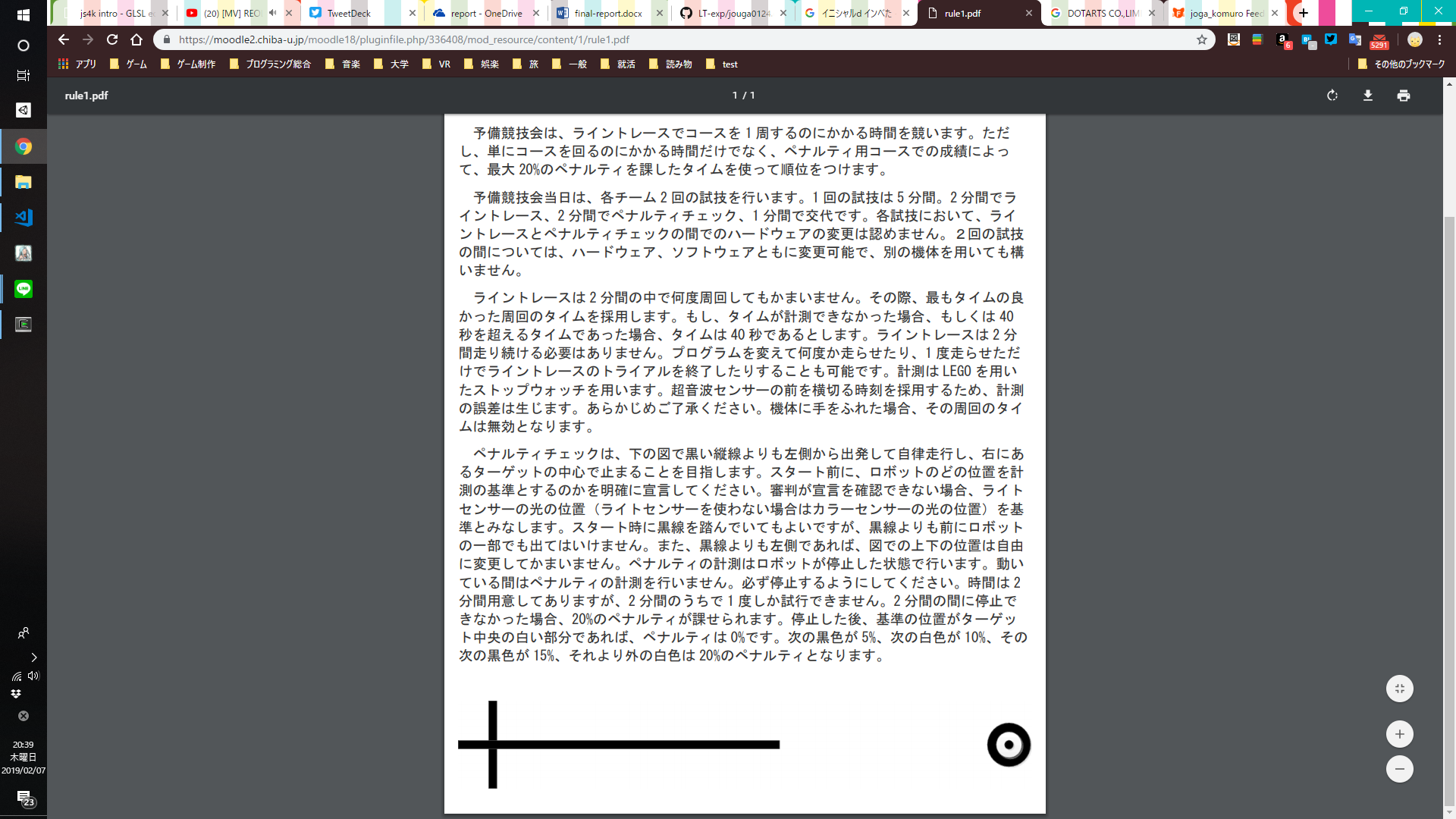


図. ペナルティ競技のコース

最初自分はState管理で5つのStateに分けるプログラムを書いた. それは左の十字から右の二重円に向かって黒いライン上を走る状態, 白い領域を走る状態, 二重円の外側の黒い線を感知した状態, 円内の白い領域を感知した状態, 円の中央の黒円を感知した状態に分けた.

しかし, これには問題があって, 1つでもStateを通らないと永遠に最後のStateにはたどり着けないというものだったり, 円の線をしっかり感知するには車体のセンサー2つともがしっかり線を認識しなければならないといけないという内容のものであった.

そこで, Stateを極端に減らし, 精度を上げるために十字から円までの移動はセンサーを使わず, 左右のモーターでPI制御をして移動するようにした. その時のプログラムを以下に示す.

|  |
| --- |
| // センサーが検知した色を黒, 白, グレーで分ける  int CalcColor(const int cval, const int lval, const int range) {  if (cval < clow + range && lval < llow + range) {  return 0; // 黒  }  else if (cval > chigh - range && lval > lhigh - range) {  return 1; // 白  }  else {  return 2; //グレー  }  }    int spd\_limit(int val){  if(val > 10) return 10;  else if(val < -10) return -10;  else return val;  }    void algorithm\_straight(void)  {  enum StraightState  {  None,  WhiteStraight,  ArrivedFirstCircle,  ArrivedSecondCircle,  StateNum,  };    // センサーで取得した値を元に計算した色  enum CalculatedColor  {  Black,  White,  GRAY,  ColorNum,  };  enum StraightState myState = None; // 現在のロボットの状態  enum CalculatedColor nextColor = ColorNum;  int range = 140; // 完全一致の黒色白色を感知するのは難しいので許容する範囲    // 左右のモーターの回転数  int RmotorCount = 0;  int LmotorCount = 0;  int subMotorCount = RmotorCount - LmotorCount;  int isAdjusting = 0;    int BASE\_POW = 30;  int distance = 1500, finish = 0;  int Rdeg, Ldeg, error, turn;  int prev\_err, integral = 0, derivative = 0;  int tol = 30;  double kp = 100.0;  double ki = 25;  double kd = 0;    /\*---------------発進動作-----------------------\*/  nxt\_motor\_set\_count(Rmotor, 0);  nxt\_motor\_set\_count(Lmotor, 0);  do  {  wai\_sem(Stskc);  lval = get\_light\_sensor(Light);  cval = get\_light\_sensor(Color);  nextColor = CalcColor(cval, lval, range);  RmotorCount = (int)nxt\_motor\_get\_count(Rmotor) / 360;  LmotorCount = (int)nxt\_motor\_get\_count(Lmotor) / 360;  if ((RmotorCount >= 6 && LmotorCount >= 6) && (myState == None))  {  myState = WhiteStraight;  }  // Calculating angle difference  Rdeg = nxt\_motor\_get\_count(Rmotor);  Ldeg = nxt\_motor\_get\_count(Lmotor);  error = Ldeg - Rdeg;  // PID  integral = integral + error;  derivative = error - prev\_err;  turn = kp \* error + ki \* integral + kd \* derivative;  // Control  // motor\_set\_speed(Rmotor, BASE\_POW+speed\_limit(turn), 1);  motor\_set\_speed(Rmotor, BASE\_POW, 1); // Master  motor\_set\_speed(Lmotor, BASE\_POW - spd\_limit(turn), 1); // Slave  prev\_err = error;  // Display  display\_goto\_xy(0, 0);  display\_string("LRdeg: ");  display\_int(Ldeg, 4);  display\_int(Rdeg, 4);  switch (myState) {  // 白領域走行  case WhiteStraight:  // 白領域を走行中に黒を感知したらステートを1つ目の円感知後に変更  if (nextColor == Black)  {  myState = ArrivedFirstCircle;  }  break;  // ↓結構ハードコーディングだけどあまりいい案が浮かなばなかった...  // 最初の黒円到達後  case ArrivedFirstCircle:  dly\_tsk(100);  myState = ArrivedSecondCircle;  break;  // 2個目の黒円到達後  case ArrivedSecondCircle:  myState = None;  motor\_set\_speed(Rmotor, 0, 1);  motor\_set\_speed(Lmotor, 0, 1);  break;  default:  break;  }  } while (1); |

図. 本番時のペナルティ競技のプログラム

結果的に練習時にはこれで円の中央に止まることができたのだが本番では何故か中央の黒円を素通りしていったのだ. この原因は2個目の黒円到達後にStateを初期値であるNoneにしているためdo While()文がまた最初から走査されてしまい, 結果的にそのまま走り抜けてしまったということであった. ここでStateをStateNumに変えておけばちゃんと止まったと考えられる. 結果的にイージーミスが大きなミスにつながってしまったためコードを書く時の良い教訓になった.

1. もの集め競技会
   1. ハードウェア

　　第一回目の物集目競技会では, ソフトウェアとハードウェアの開発を両立するためにNXTは置くだけで本体に設置できるようにした(図1). これにより, NXT本体の取り外しの自由度が高まった. 本体の構造としては, ステージの特性を考察してカラーセンサーを本体の左部分に設置するようにした. また, アルゴリズム設計の負担を減らすよう, シンプルな動作でオブジェクトを回収できるような本体設計を目指して, アーム部分は低く設計した. これにより, オブジェクトに直進するだけでオブジェクトを回収できるようにした. シンプルに, 最低限の動作を行えるような本体設計をめ実現した. しかし, 壁際でのアームの上下運動の際にアームが壁に引っかかったり, オブジェクトに対して進入する角度によっては上手く回収できないという問題もあった.

　　第二回の物集め競技会では, 第一回における本体の問題点の改善とさらなる改良を目指した. まず, NXTを本体の一部として組み込む設計に変えた. NXTにもパーツを拡張する部分があるので本体拡張の自由度が上がった. 次に, アームなどの本体の前方部分を大幅に改良した. 本体の両側に設置したタッチセンサーを誤作動なく反応させるために, タッチセンサーの先端から前方に向けて拡張している部分の強度を上げた. これにより, タッチセンサーの誤作動を防ぎながらオブジェクト回収にも役に立つようになった. また, 第一回で課題となった壁際でのアームの動作において引っ掛かりが生じてしまうことについては, 第一回ではアームの根元が本体上部にある事が原因であったと考えたのでそれを地面に近くなるよう設計し直した. これにより壁際でアームが引っかかる事がなくなった.

　　いずれの物集め競技会もシンプルさをモチーフに設計を進めた. 簡単な部品の取り外しなどが可能であったことにより効率的に作業を進める事ができた.

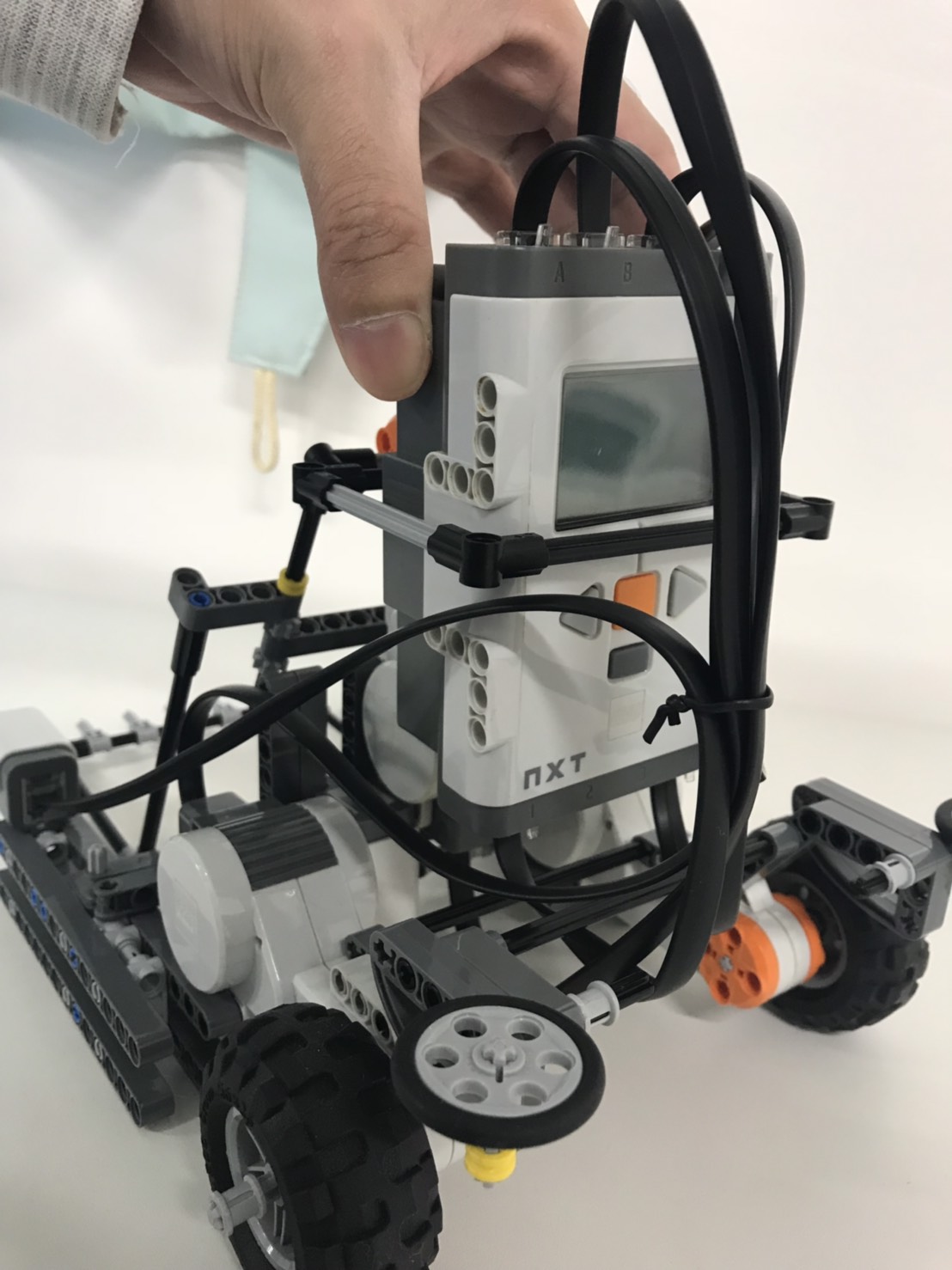


図1. ハードウェア本体

* 1. ソフトウェア
  2. 第一回目

第一回目の物集め競技会では1つも物を集めることはできなかった. 以下に原因を示す.

* + - 1. タスク管理, センサー処理などの細部の動作に時間をかけすぎた.
      2. 旋回用の関数がなかったため, 車体の方向を変える動作に柔軟性が足りなかった.
      3. 実験回数が足りなかった.

結局基本の軸となるアルゴリズムよりも細部の機能に時間をかけすぎたたため物集め本体のアルゴリズムにかける時間が足りなくなり中途半端な結果になってしまったと考えられる.

次に具体的な第一回目の物集め競技会のアルゴリズムを説明する.

物集め全体のアルゴリズムはハードコーディングで技術的に説明することはなにもしていないためこのアルゴリズムで使われているmov\_func()関数とarm\_func()関数について説明する.

|  |
| --- |
| void arm\_func(int POW, const int DEG)  {  // TODO: 上げ下げのオプション  // 正：下，負：上  int Adeg;  nxt\_motor\_set\_count(Amotor, 0);  if(DEG < 0)POW = -2\*POW;  while(1){ // <= OR >= (?)  wai\_sem(Stskc);  Adeg = nxt\_motor\_get\_count(Amotor);  if(DEG < 0 && Adeg <= DEG) break;  else if(DEG > 0 && Adeg >= DEG) break;  motor\_set\_speed(Amotor, POW, 1);  // アーム角度表示  display\_goto\_xy(1, 1);  display\_string("Arm:");  display\_int(nxt\_motor\_get\_count(Amotor), 4);  display\_update();  motor\_set\_speed(Amotor, 0, 1);  } |

図. arm\_func()関数

arm\_func()関数は第一引数にモーターのパワー, 第二引数に上げ下げのdegreeを受け取る. 内部ではDEGの正負でアームの上げ下げの区別し, モーターの回転数がDEGの絶対値を超えたらモーターを止めることでアームを動かす仕組みになっている.

また, アームの上げ下げ角度をディスプレイに表示する機能も付いている.

|  |
| --- |
| void mov\_func(const int POW, int RATIO, const int DEG)  {  //正：右，負：左  int Ldeg, Rdeg, turn;  int cur\_err, prev\_err, integral, derivative;  double kp = 70.0;  double ki = 0;  double kd = 0;  nxt\_motor\_set\_count(Lmotor, 0);  nxt\_motor\_set\_count(Rmotor, 0);  prev\_err = integral = derivative = 0;  do{  // PID制御  Ldeg = nxt\_motor\_get\_count(Lmotor);  Rdeg = nxt\_motor\_get\_count(Rmotor);  cur\_err = Ldeg-Rdeg-RATIO;  integral = integral + cur\_err;  derivative = cur\_err - prev\_err;  turn = kp \* cur\_err + ki \* integral + kd \* derivative;  motor\_set\_speed(Lmotor, POW - spd\_limit(turn), 1); // -spd\_limit(turn)?  motor\_set\_speed(Rmotor, POW + spd\_limit(turn), 1);  prev\_err = cur\_err;  // モーター角度表示  display\_goto\_xy(1, 2);  display\_string("Lmotor:"); display\_int(Ldeg, 4);  display\_goto\_xy(1, 3);  display\_string("Rmotor:"); display\_int(Rdeg, 4);  display\_update();  if(POW < 0 && Ldeg <= DEG) break;  else if(POW > 0 && Ldeg >= DEG) break;  } while(1);  motor\_set\_speed(Lmotor, 0, 1);  motor\_set\_speed(Rmotor, 0, 1);  } |

図. mov\_func()

第一引数にマスターモーターのパワーを第二引数に曲がり具合を第三引数にどのぐらいの回転数mov\_func()を実行するかという値が与えられている. 第二引数が0であればまっすぐに進み, 正であれば右寄り, 負であれば左寄りというようになっている. 内部ではPID制御によってマスターモーターとスレイブモーターのパワーを切り替えることで与えられた値に準ずる正確な走行が可能となる. また, デバッグ用としてディスプレイに現在の左右のモーターの回転数を出力する機能も付いている.

また競技会では使用しなかったが, 色センサーの判定アルゴリズムについて説明する.

|  |
| --- |
| typedef enum  {  BLK = 1 << 0, // 黒  BLU = 1 << 1, // 青  GRN = 1 << 2, // 緑  CYA = 1 << 3, // シアン  RED = 1 << 4, // 赤  MAG = 1 << 5, // マゼンタ  YEL = 1 << 6, // 黄  WHT = 1 << 7, // 白  RTP = 1 << 8, // 右押す  RTR = 1 << 9, // 右離す  LTP = 1 << 10, // 左押す  LTR = 1 << 11, // 左離す  DIS = 1 << 12, // 移動距離  POS = 1 << 13, // アーム位置  // 以下ライトセンサー?  } EBits;  static int COL\_THRES[] = {400, 350, 320};  S16 col[3];  U8 CBits = 0;  ecrobot\_get\_nxtcolorsensor\_rgb(Color, col);  CBits = bin(col[0], COL\_THRES[0], 2) |  bin(col[1], COL\_THRES[1], 1) |  bin(col[2], COL\_THRES[2], 0);  // フラッグをクリアしてからセットする  clr\_flg(Fsens, ~(BLK | BLU | GRN | CYA |  RED | MAG | YEL | WHT)); (?)  switch (CBits)  {  case 0:  set\_flg(Fsens, BLK);  break;  case 1:  set\_flg(Fsens, BLU);  break;  case 2:  set\_flg(Fsens, GRN);  break;  case 3:  set\_flg(Fsens, CYA);  break;  case 4:  set\_flg(Fsens, RED);  break;  case 5:  set\_flg(Fsens, MAG);  break;  case 6:  set\_flg(Fsens, YEL);  break;  case 7:  set\_flg(Fsens, WHT);  break;  } |

図. カラーセンサー

このようにビット演算を使用して色を計算した. 閾値はソースコード上のCOL\_THRESであり, 実際の競技場の色を調べながら決めることで正確な判定ができるようになった.

* 1. 第二回目

第二回目の物集め競技会では第一回目の失敗を反省し作業に取り掛かったが, 第二回目でも物はあまり集められなかった. 新たな原因を以下に示す.

1.フィールドの少しの環境変化でうまく動作しないことが多くあった.

結果的に練習環境では成功することもあったが失敗することもあり不安定な状態で競技に臨んだ結果, 本番環境ではうまくいかなかったのが一番の敗因だろう.

次に第一回目の競技会では実装されていなかったsteering()関数について説明する.

|  |
| --- |
| void steering(int direction, float angle)  {  // direction (-1 : 左 | 1 : 右)  // angle (ステアリング角度の絶対値)  // 動かすモーターの回転数を格納  int moving\_motor\_count = 0;  // 動かすモーターと止めるモーターのポート  const int moving\_motor = (direction == 1) ? Lmotor : Rmotor;  const int stopping\_motor = (direction == 1) ? Rmotor : Lmotor;  // 初期化  nxt\_motor\_set\_count(Rmotor, 0);  nxt\_motor\_set\_count(Lmotor, 0);  // 旋回開始  nxt\_motor\_set\_speed(moving\_motor, -75, 0);  nxt\_motor\_set\_speed(stopping\_motor, 0, 1);  while (moving\_motor\_count < angle)  {  moving\_motor\_count = -nxt\_motor\_get\_count(moving\_motor);  }  // 旋回終了  nxt\_motor\_set\_speed(moving\_motor, 0, 1);  } |

図.steering()

steering()関数は第一引数に左回転か右回転かを判定するためのdirectionを受け取り, 第二引数に旋回するときに動かすモーターの回転数を受け取る.

内部ではdirectionの値によって動かすモーターと止めるモーターを決め, 動かすモーターがangle以上になったら旋回終了という単純なアルゴリズムになっている.

mov\_func(), arm\_func(), steering()にしろ, どれも関数が呼ばれるタイミングによって回転数も大きく変わるためある程度の誤差が生じるためその誤差の対処もしておくようするべきであったと考える.

1. 全体の考察とまとめ

全体的に言えることは初めから細部にこだわりすぎたということだろう.

今回のように時間が限られている開発の中ではまず競技全体のアルゴリズムをハードコーディングでも完成させることで最低限のプロダクトを用意し, 精度を上げる過程で細部のセンサー処理などの実装にとりかかるべきであったと思う.

自分たちはチーム内でのタスク管理が少し曖昧だったためその辺りの認識をそろえる必要があったと考えた. (鈴丸)

1. 感想

鈴丸 : 結果は確かに悪かったけど, 決してサボってたわけでもなくて皆で色々意見を出しながら開発していくのは新鮮で楽しかった. PC上で動かすプログラムと違って実際の環境などが関係してくるため安定しない状態だったし, 期限も厳格に決まっているため色々と腑に落ちないところもあるがそれでも結果をちゃんと残せたチームがいるのはすごいと思った. できたらまたリベンジとかしたいなぁ...

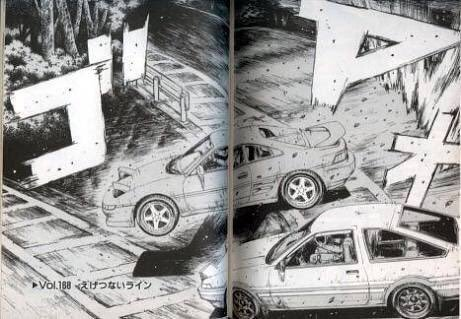


「疲れた...」



「肉球パンチだにゃ～～😼👊」





「インベタのさらにインというのは空中に描くラインだ!!🚘」

参考文献