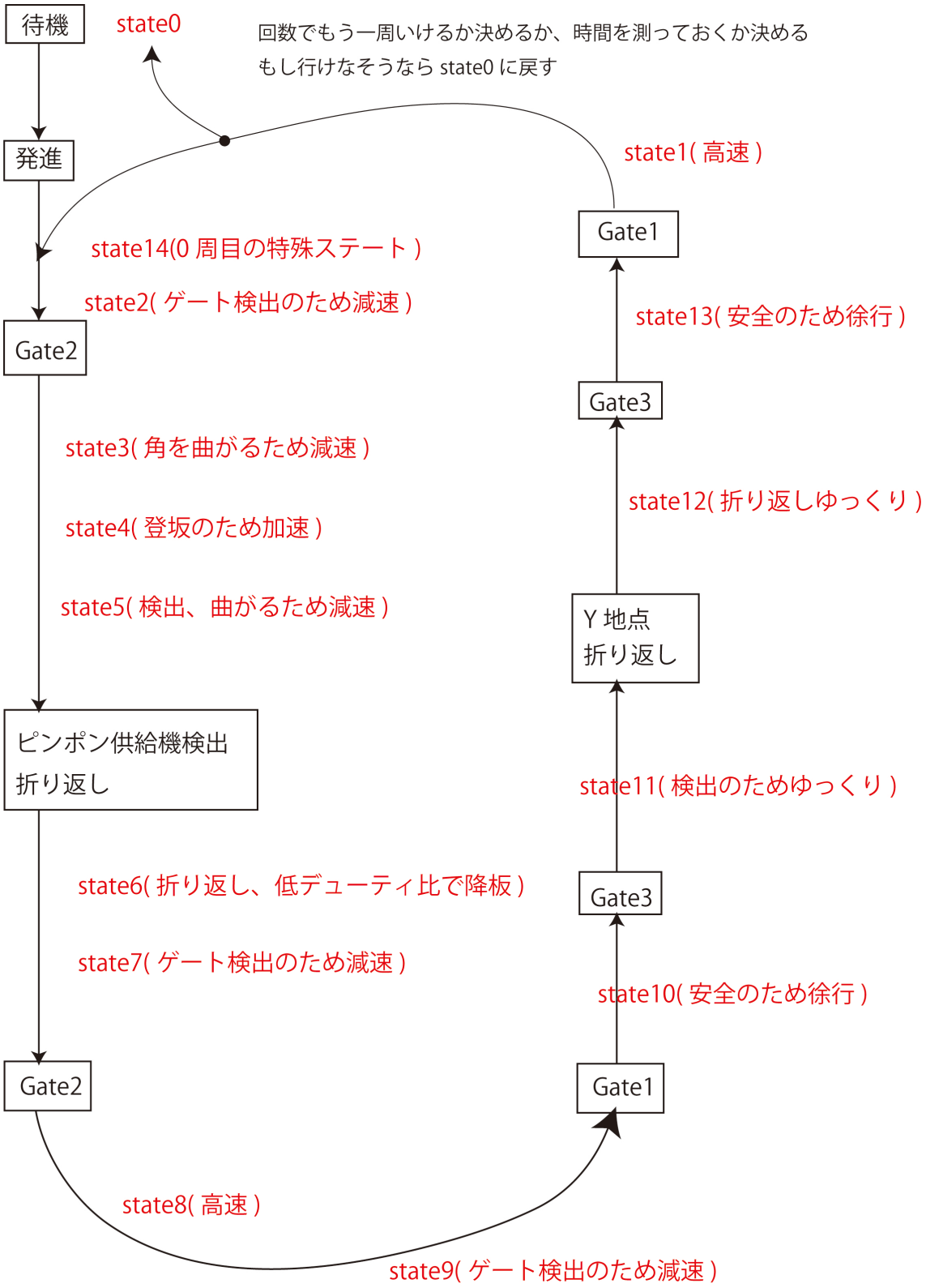
待機
state0
回数でもう一周いけるか決めるか、時間を測っておくか決める
もし行けなそうなら state0 に戻す
発進
state1( 高速 )
Gate1
state14(0 周目の特殊ステート )
state2( ゲート検出のため減速 )
Gate2
state13( 安全のため徐行 )
Gate3
state3( 角を曲がるため減速 )
state12( 折り返しゆっくり )
state4( 登坂のため加速 )
state5( 検出、曲がるため減速 )
Y 地点
折り返し
ピンポン供給機検出
折り返し
state11( 検出のためゆっくり )
state6( 折り返し、低デューティ比で降板 )
Gate3
state7( ゲート検出のため減速 )
state10( 安全のため徐行 )
Gate2
Gate1
state8( 高速 )
state9( ゲート検出のため減速 )

# これは適当に話し合った結果をまとめたものなので、修正点は言ってください

## 1　ステート毎の挙動についての概略と注意すべき点

- state0
  静止ステート
- state14
  最初の周は例外的に低速で行く。1 周目以降のこの区間と別々にした理由は、ゲート検出のための減速を行う state2 を必要としないため。そのまま state3 に入る。
- state3
  ゲート検出後、角を曲がるために低速を維持するステート。
- state4
  登板用ステート。デューティ比を限界まで上げるべきと考えられる。やはり一定時間後に減速をする。ただし、もし限界までデューティ比を限界まで上げてもそこまでスピードがでないのだとすれば登板区間終わりのカーブで脱輪することはないと考えられるので、state4 に遷移させずほっといてもピンポン球センサの検出が可能かもしれない。これは実験して決める。もし無理なら減速。
- state5
  減速ステート。登板区間終わりのカーブを曲がる目的と、ピンポン球供給機の検出するための徐行が目的。検出し次第折り返す。登板でスピードが出ないのならばまあほっといてもカーブは曲がれるだろうが、曲がった後の平らな部分で加速しないようにデューティを変更することは必要だろう。
- state6
  X 地点から降板区間までは距離がないので、この区間は「降板部分用の低いデューティ比で走行」してステート分けしなくてもあまり変わらない気がする。
- state7
  降板で加速しているだろうから、ゲート 2 を検出するために減速する・・・と思ったけど、もしかしたらここも state6 に統合してもいい気がする。すでに低いデューティ比で走行してるから、これ以上低くできるのか？ってことと、降板終わりからゲートまでの間は坂道ではないので勝手に減速してくれるのではないか？ってことを考えました
- state8
  高速で動く。直線で稼ぐ？ただし、一定時間後に減速してゲート 1 の検出を出来るようにする必要がある。
- state9
  ゲート検出のために減速するステート
- state10
  蛇行してるからゆっくり走ればいいと思われる。ゆっくり走ればきっとゲート 3 も検知できるはず。
- state11
  すぐにゲートがあるのでゆっくり走ればいいだろう。state11 から state12 は時間がかからないはずなので、もし state11 になったあとにしばらく折り返し地点の検出がされなかった場合は state12 に自動

で変更してしまうべきだろう。

- state12
  折り返したあとにゲート 3 につくまでのステート。ここも距離が短いので低速で走ればいいだろう。
- state13
  ゆっくりはしる。ゲート 1 を感知したときに、現在何周したかを数えておいて停車するためのステートメントに移るかを決める。もしまだ回れそうなら回る。もし回るなら state1 に入る。
- state1
  高速で動く。一定時間後に state2 に入りゲート検出のため減速する。
- state2
  state1 で高速になっていると考えられるので、最初のカーブを曲がるときに脱輪しないように減速するステート。state1 に入って一定時間経過後[*1]に自動的に state2 に入る。ゲートを検出したら state3 に入る。

## 2 注意すべきこと

- センサの誤作動として、同じゲートを複数回検出してしまう、いわゆるチャタリングのような状態に陥る可能性があること。すなわちソフトウェアの方でこれを防止する必要がある。具体的にはチャタリング防止のように、入力信号をパルス波に変換して、この値が立ち上がった瞬間から一定時間割り込みを禁止すれば良い？
- 周回数は、とりあえずプログラムには`#define n 数`とでもしておく。データを取って、もし一周するのにかかる時間の統計の分散があまりに大きかったら、しかたなく時間を見てまわるのをやめるかやめないか考えるプログラムをいれる。まあ実際そんなにきつくは無いと思うけど。
- センサーの検出に関しては、フラグとして flg でも定義しておいて、閾値を超えた瞬間に flg=1 となり、そのとき state を変更する。

[*1] 実験して最適な時間を測る