About P loops

Well it's going to be more about the p loops *I used*. It's pretty simple, `power = Kp * error`, where `error = target position - current position.` I usually set the current position to be the average value of left and right motor rotation (need to reset before movement) for straight line movements.

It might take a long time for the robot to settle since the power will be close to 0 when the robot is close to the target. The simple (**and bad**) solution I used when I first got started is to add or subtract the power by 5 (depending on the direction of movement) so it won't take forever for the robot to reach the target. If well-tuned, the robot settles quickly. However, one big flaw is that if the robot overshoots the target, it will take it a significant amount of time for it to correct itself.

Update:
A better way to ensure abs(minimum power) is always greater than 5% would be to set power = 5 or -5 when the power is 0 < power < 5 or -5 < power < 0

A lot of times the power calculated by Kp * error will be greater than 100 or smaller than -100. In order for the heading correction algorithm to work as intended, we need to limit those powers to 100 or -100 if they are greater or smaller than that. For example, `if(power > 100); power = 100.`

pseudo code for a simple moving forward p loop (2m drive):

```
//reset motor rotations
left motor reset rotation;
right motor reset rotation;
//calculate error and power
current position = (left motor rotation + right motor rotation) /
2;
error = target position - current position;
power = Kp * error;

//while loop to keep everything updated
while (absolute value of error > error tolerance) {
 //update a bunch of things
   current position = (left motor rotation + right motor rotation)
/ 2;
  error = target position - current position;
  power = Kp * error;

  //limiting power
  If (power > 100){
    power = 100;
  }
  If (power < -100){
    power = -100;
  }

  send power to motors;
  task sleep for 10 ms;
}
stop the motors;
```

I would recommend PID instead of p loops. Here's the document I used when I was learning about PID:
https://docs.google.com/document/d/1D61dinGqP__CHzRfc9yNugnS9K-JTa395_r0qP7L5W8/edit

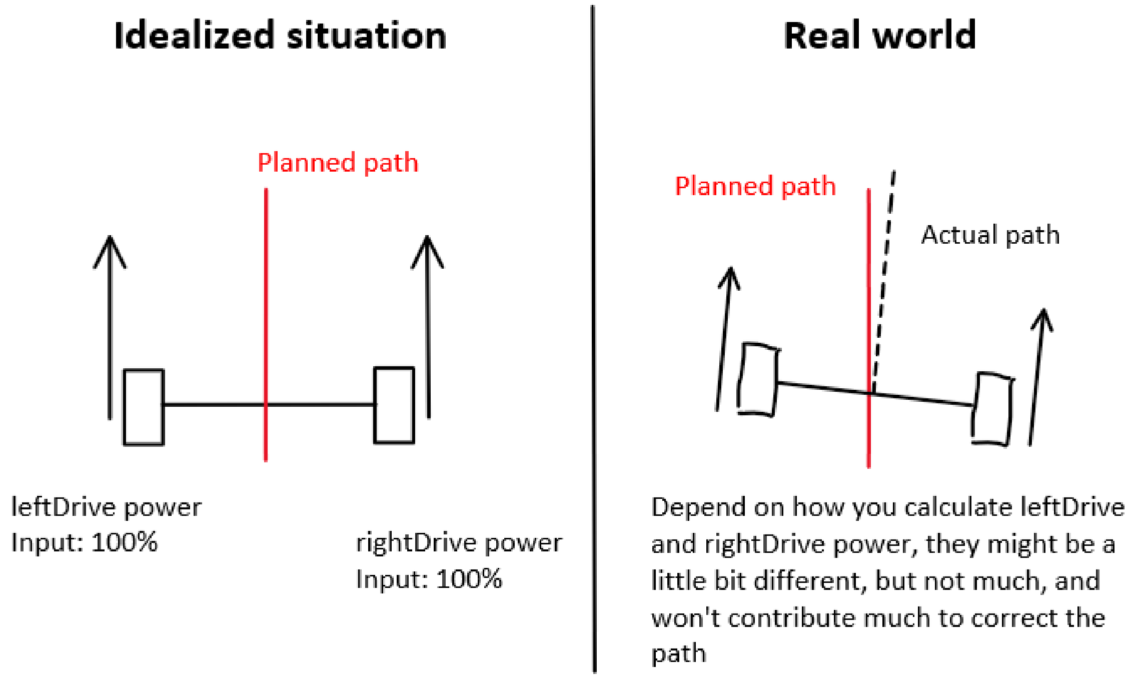
Since I'm only using the P controller, different movements might need different Kp values. So create a P loop function and give each movement a Kp value that works best for this particular movement. A Further target usually requires a smaller Kp.

About the algorithm I used

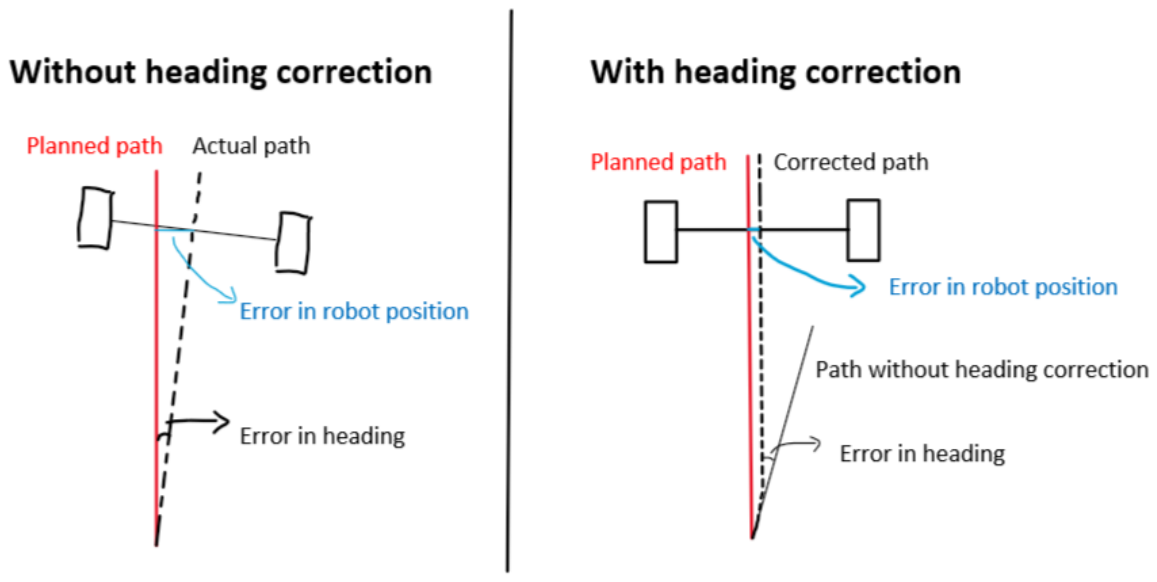
Advantages: it's simple and easy
Disadvantages: cannot eliminate error completely
More advanced algorithms: odometry, motion profile, pure pursuit controller

*This heading correcting algorithm can be applied to all straight line functions. The straight line function itself does not have to be PID or p loop, even if you set the forward speed to be a constant you can still add this heading correction algorithm to it and make it more accurate

Even if PID loops are applied to every drive motor, the high starting speed can drift the robot easily.


Idealized situation
Planned path
leftDrive power
Input: 100%
rightDrive power
Input: 100%
Real world
Planned path
Actual path
Depend on how you calculate leftDrive and rightDrive power, they might be a little bit different, but not much, and won't contribute much to correct the path

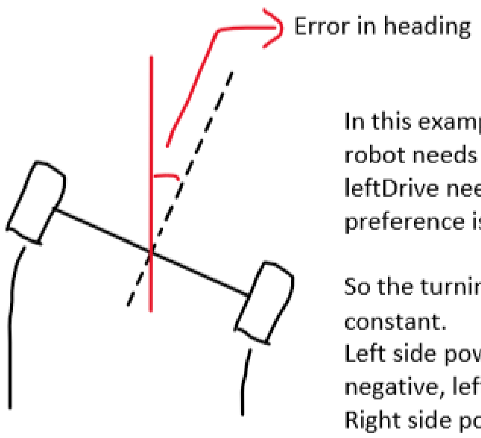
This would result in inaccuracy in position and heading when the robot reaches the end point. But if we somehow correct the heading of the robot as soon as it begins to drift, the error can be reduced significantly.



And this is exactly what I used for my old program. It does not eliminate the error completely, but with the help of sonar and other sensors, the whole routine can be quite accurate and consistent.

So how do we correct the heading *while* the robot is moving forward?

When leftDrive and rightDrive have the same power, the robot moves in a straight line. When they have different power input, the robot travels along a curve or turns. So, if we calculate a "turning power" based on the difference between the planned heading (which should be the heading of the robot before it moves since we are trying to get it travel in a straight line) and the actual heading, and add this power to the power moving the robot forward, the robot will corrects its heading while moving forward.



In this example, the robot's heading is off by about 25 degrees, and the robot needs to turn counter clockwise by 25 degrees. This means that leftDrive needs to slow down, and rightDrive needs to speed up. My preference is that clockwise is positive, counter clockwise is negative.

So the turning power in this situation = - 25 * Kp, where Kp is a positive constant.
Left side power = forward power + turning power (turning power is negative, left side needs to slow down)
Right side power = forward power - turning power (turning power is negative, right side needs to speed up)

Pseudo code for heading correction:

```
//calculate how much the heading is off
heading error = target heading – current heading;
//if you choose to use p loop instead of PID…
turning power = Kp_turning * heading error;
//combine turn power with forward power
left side power = forward power + turning power;
right side power = forward power - turning power;
```

Something you need to be careful about this turning power, is that if you add this to maximum or minimum power (100 or -100), the heading correction would be slower since only one side is speeding up/slowing down.

So, if `(forward power + turning power > 100)` or `(forward power – turning power < -100)`, the other side needs to be speed up/slow down more in order to make the heading correction process as fast as usual.

Pseudo code:

```
//example code if left side power is greater than 100
Left side power = forward power + turning power;
Right side power = forward power - turning power;
If (left side power > 100){
 //this means right side's power needs to be further reduced in order for left side to catch up
 //the amount of reduction equals (left side power - 100)
 //so the new right side power equals:
 right side power -= (left side power - 100);
}
```

The correction code for when power is less than -100 is similar.