Controllers

Algorithm 1 Bang-Bang Controller

```
i ← idealDistance
while true do
  z ← readSensor()
  if z < i then
    moveAway()
  else if z > i then
    moveTowards()
  else
    driveStraight()
  end if
end while
```

The function setVel($\dot{x}$, $\dot{\theta}$) takes two arguments, $\dot{x}$ which is the forward velocity, and $\dot{\theta}$ which is the angular velocity (how fast it's turning). In Player, set_cmd_vel() is like setVel, but it takes a few more arguments.

If we are close to our ideal distance, we don't want to move towards, or away as aggressively as if we are further away. We can make our angular velocity, $\dot{\theta}$ proportional to the actual distance (as measured by our sensor, z) we are from the ideal distance (i).

$$\text{Dist} \quad = \quad |z - i| \tag{1}$$

In the P-Controller algorithm, K_P is a constant. Try different values for K_P until you find one that works best. Depending on if you are left, or right wall following, K_P may need to be negative.

Algorithm 2 P-Controller

$i \leftarrow$ idealDistance
while true **do**
  $z \leftarrow$ readSensor()
  setVel(1.0, $K_P \cdot (z - i)$)
end while

If we compare our current sensor reading to the previous sensor reading, we can determine if we've been moving towards or away from our ideal distance. This can help prevent overshooting our target.

$$\frac{\Delta \text{Dist}}{\Delta t} \approx z_{\text{curr}} - z_{\text{prev}} \tag{2}$$

Algorithm 3 PD-Controller

$i \leftarrow$ idealDistance
$z_{\text{prev}} \leftarrow$ readSensor()
while true **do**
  $z_{\text{curr}} \leftarrow$ readSensor()
  setVel(1.0, $K_P \cdot (z_{\text{curr}} - i) + K_D \cdot (z_{\text{curr}} - z_{\text{prev}})$)
  $z_{\text{prev}} \leftarrow z_{\text{curr}}$
end while

K_D is another constant parameter. As with finding K_P in the P-Controller, experiment to find the best K_D for your application. K_D will generally have the opposite sign of K_P.

$$\text{Err}_t = \text{Err}_{t-1} + (z_{\text{curr}} - i) \tag{3}$$
$$\text{Err} = \sum_t z_{\text{curr}} - i \tag{4}$$

One problem with the Integral term is that it is unbounded. It may grow too large, in which case the Integral term dominates the output of the controller. One solution is to provide bounds for e (i.e. add in an "if $e <$ min then $e =$ min , if $e >$ max then $e =$ max").

Algorithm 4 PID-Controller

$i \leftarrow$ idealDistance
$z_{\text{prev}} \leftarrow$ readSensor()
$e \leftarrow 0$
while true **do**
  $z_{\text{curr}} \leftarrow$ readSensor()
  setVel(1.0, $K_P \cdot (z_{\text{curr}} - i) + K_I \cdot e + K_D \cdot (z_{\text{curr}} - z_{\text{prev}})$)
  $z_{\text{prev}} \leftarrow z_{\text{curr}}$
  $e \leftarrow e + (z_{\text{curr}} - i)$
end while