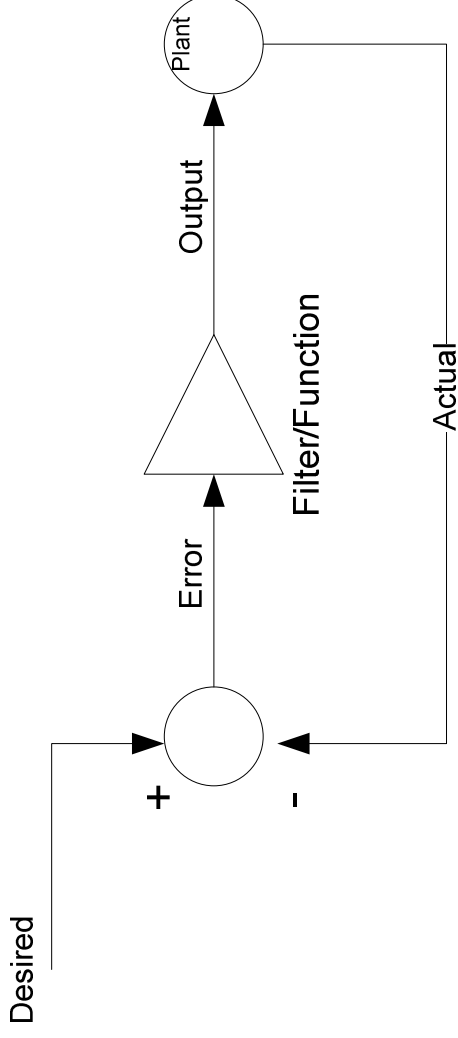


Basic Control



Desired:

- Position
- Velocity
- Force
- Temperature
- Etc

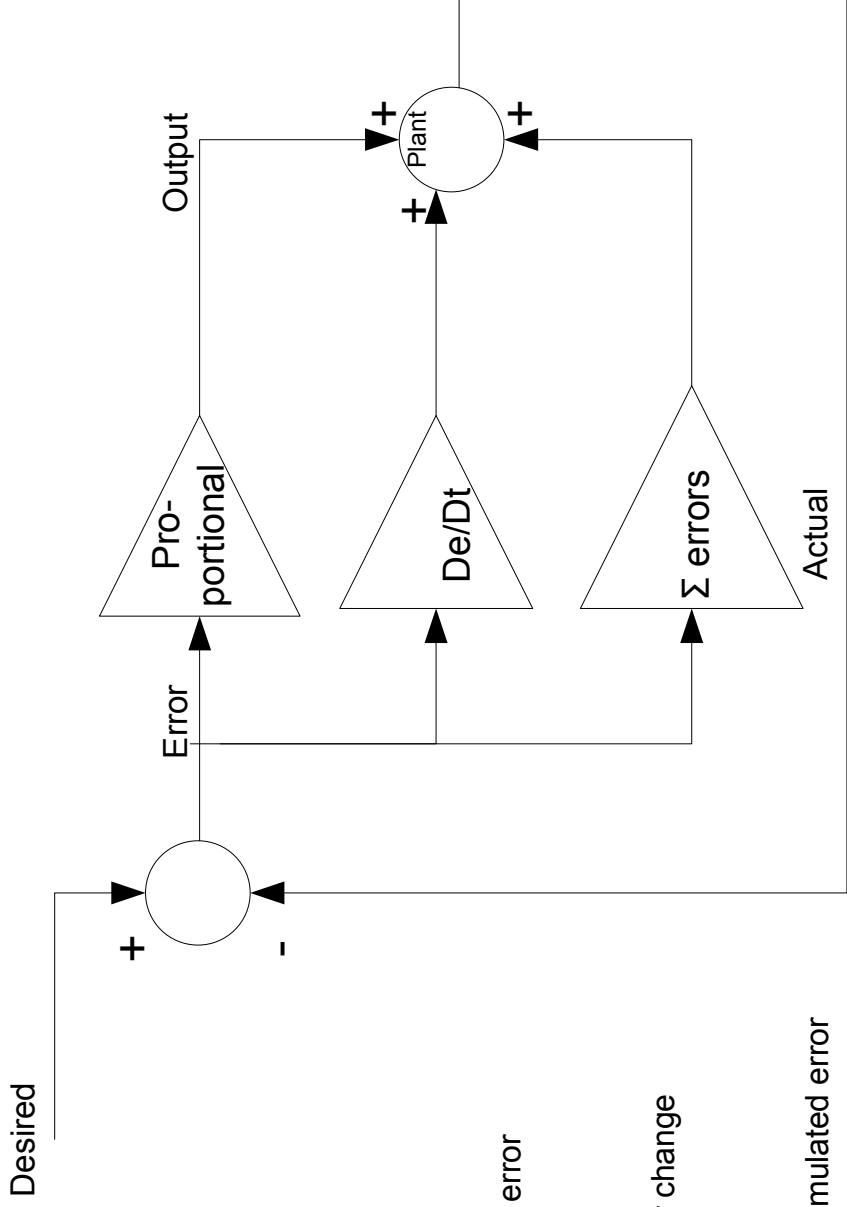
Plant:

- Robot
- Fan
- Lights
- Pump
- Etc

Filter/Function

- transforms error
- over time into an
- output signal that
- drives the plant

Basic PID Control



Proportional

Output proportional to error

Derivative

Output based on error change

Integral

Output based on accumulated error

Practical PID

Simple code; fixed period (dt) simplifies

$$\text{Error}_n = \text{Desired}_n - \text{Actual}_n$$

$$\text{Output}_n = K_p * \text{error}_n + K_d * (\text{Error}_n - \text{Error}_{n-1}) + K_i * i\text{Error}$$

$$i\text{Error}_n = i\text{Error}_{n-1} + \text{Error}_n$$

Works well for linear processes or non-linear over narrow ranges.

Assumes Output linearly affects the plant.

Derivative term dampens response (opposes inertia)

Integral term eliminates constant error (friction)

Variable period would require time (dt) value in all terms

Basic PID Tuning

- Set all gains zero
- Increase K_p until oscillations occur
- Set K_p to $2/3$ of oscillating value
- Increase K_d until there is no overshoot and response is snappy
- Increase K_i (slowly) to remove residual error.

DC Motors, theory

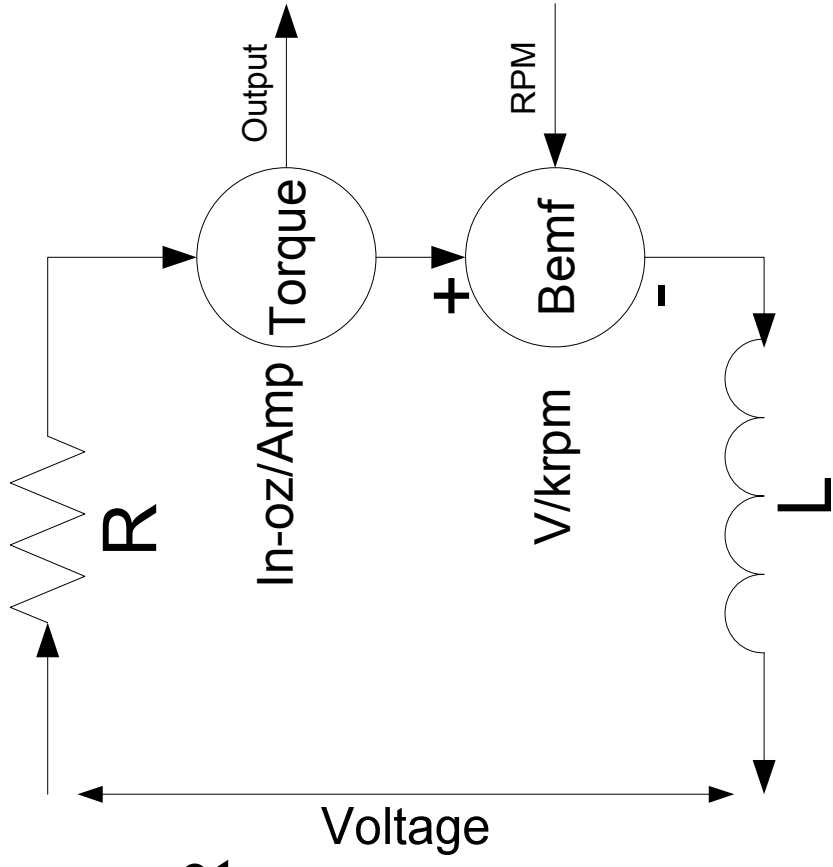
Google “H-Bridge Demystified” (2003)

- $I = V/R$
- $V = I \cdot R$
- $dI/dT = V/L$
- $V = L \cdot dI/dT$
- $W = I \cdot V$
- $W = I^2 \cdot R$
- $W = V^2/R$
- $I =$ current (amps)
- $V =$ volts
- $L =$ inductance
- $W =$ power (watts)
- $R =$ resistance (ohm)

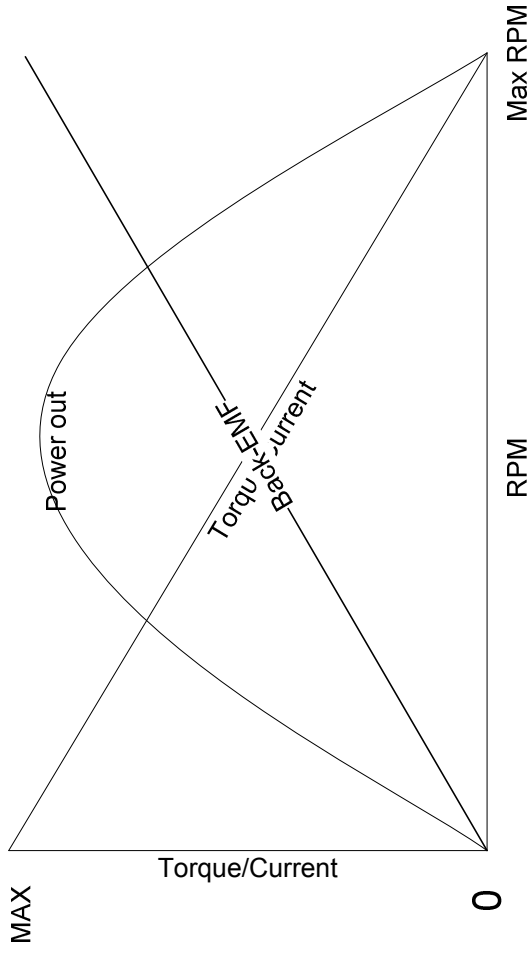
DC Motor Model

- Torque (force) \sim Current
- Max Current = Voltage/R
- Max RPM = $V/Bemf$

In general:
Torque $\sim (V - Bemf)/R$



DC Motor; Charts



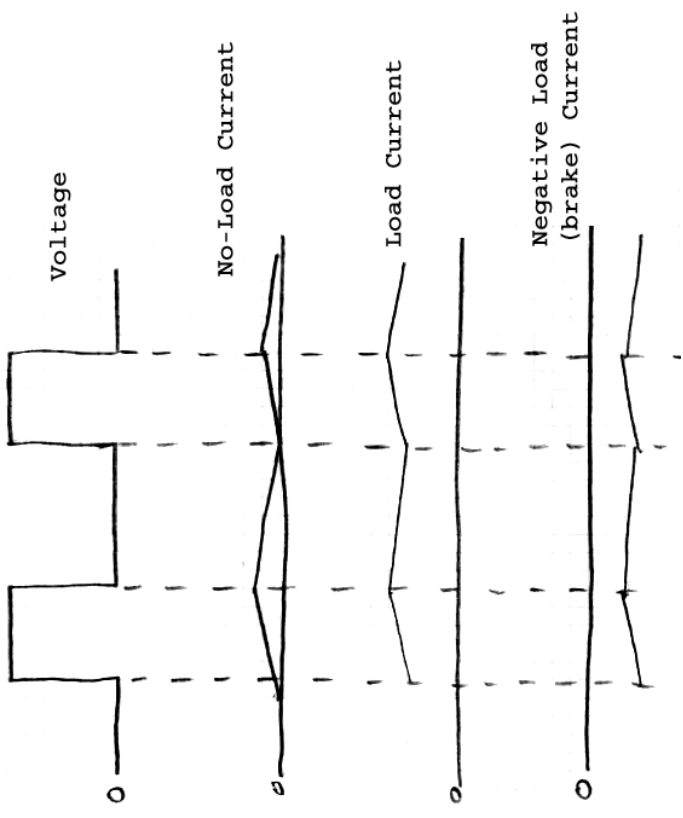
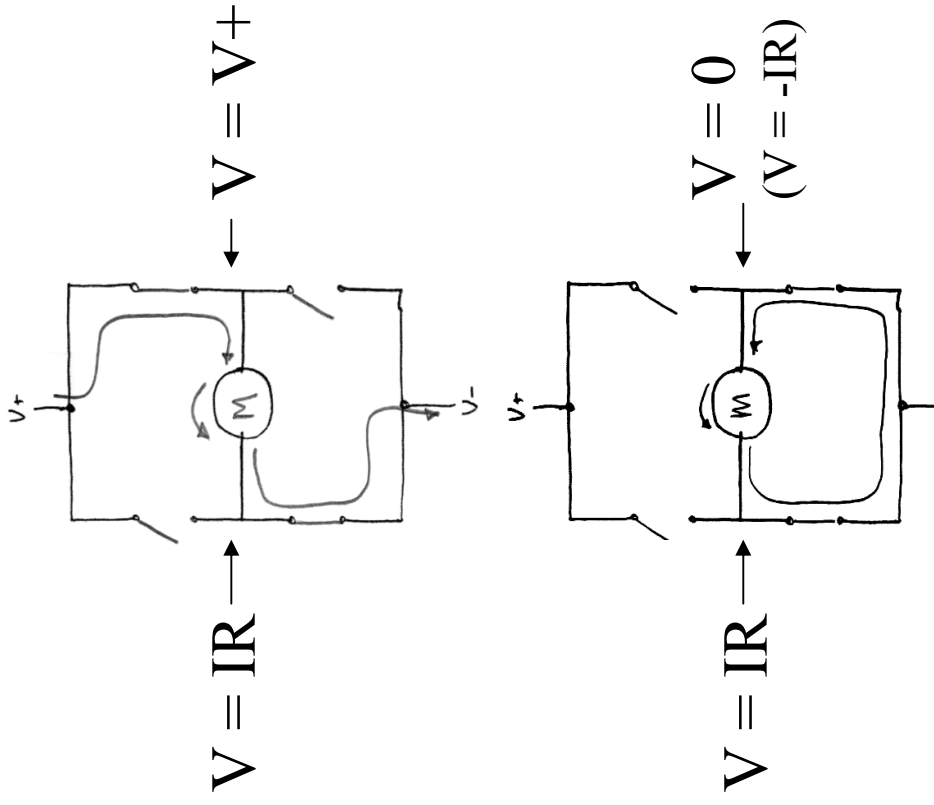
- Current/Torque maximum at stall
- Back EMF increases with RPM
- Max speed determined by Voltage
- Friction reduces maximum speed

DC Motor Output (PWM)

- Pulse Width Modulation
- Applies full voltage for a variable amount of time (from 0 to 100%)
- Inductance effectively converts pulses to constant voltage.
- See “H-Bridges Demystified”

H-Bridge/Inductor operation

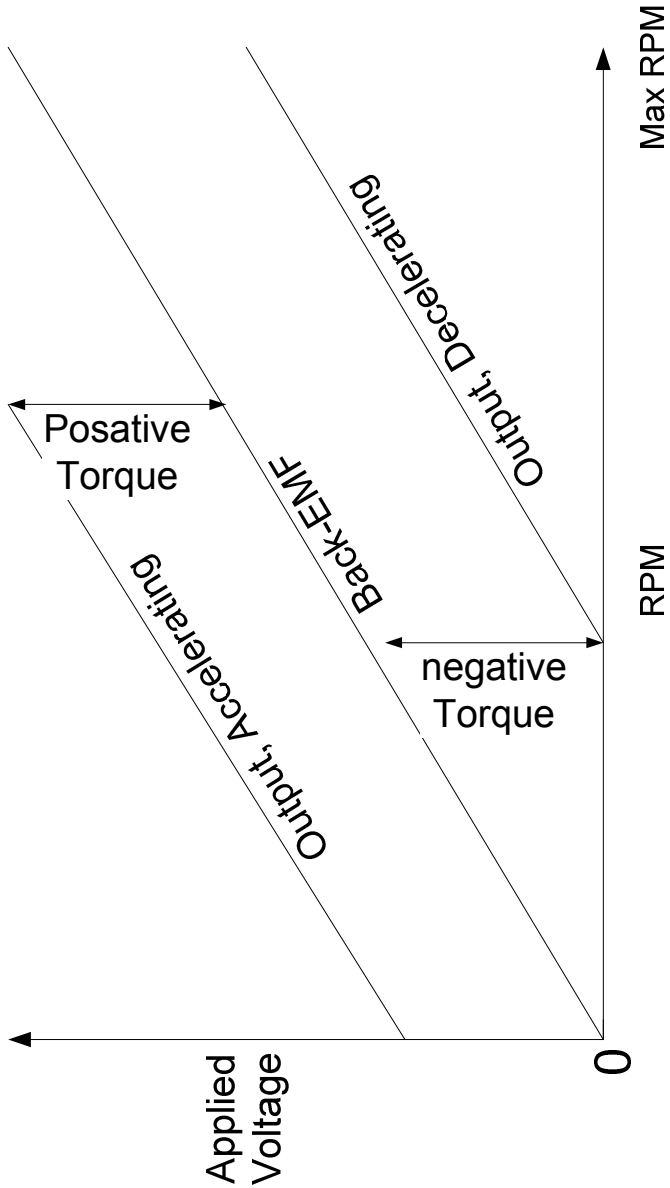
from *H-Bridge Demystified*



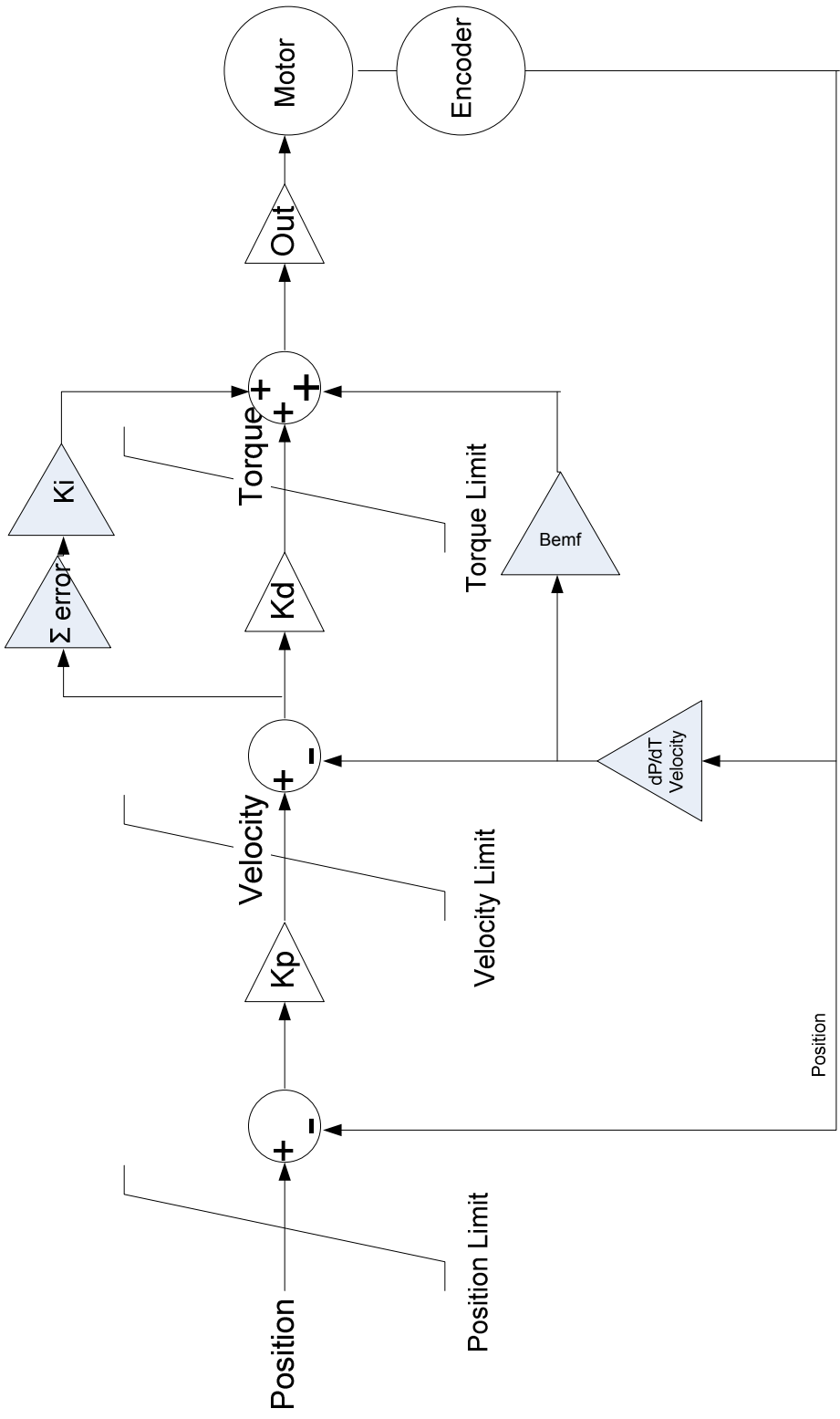
More theory (Yawn)

- Force = Mass*Acceleration
- Acceleration = Force/Mass
- Increasing RPM = Decreasing Torque
- However, deceleration torque goes up!
(Bemf works FOR you, not against)
- Simple PID works poorly in this case

Output Control



Cascade PID loop



Code to implement

```
• tVelocity = p->Position - p->PrevPosition;
• P->PrevPosition = p->Position;
• p->VelSp = LimitStuff( (p->PosSp - p->Position) * p->Kp, p->VelLim);
• VelocityError = p->VelSp - tVelocity;
• NewTorque = ((VelocityError * p->Kv) + (p->IntError * p->Ki));
• p->TorqSp = LimitStuff(NewTorque, p->TorqLim);
• if (NewTorque == p->TorqSp)
    p->IntError += VelocityError;
• Drive = p->TorqSp + (tVelocity * tKem);
```

Simple, eh?

- There is no position limit
- Velocity error is the difference between current velocity & setpoint
- The requested torque is Velocity error * Kd limited to defined values
- Finally, the back-EMF is added to the requested torque to get a PWM output.
- Details were left out: Most numbers are fractions...

Cascade PID details

- All values in encoder ticks & loop rate
- Fixed loop rate removes a lot of math
 - E.g. ticks/dT to get velocity
- I never use integral terms
- I almost never use position, just velocity.
- I make Kv variable increasing with Velocity Setpoint.

Variable K_d , what?

- Typical robot has slop in drive train
- When still, slop effectively reduces inertia
- Velocity term oscillates
- When moving, slop is taken up and inertia felt by PID algorithm increases (the mass of the robot)
- The faster the robot goes, the more friction and the higher the gain

Other robot considerations

- **Feedback**
 - Encoder (Position, Velocity)
 - Potentiometers (Absolute position)
 - Tachometers (Velocity only)
- **Following a path/Trajectory**
 - Drive train slop makes this difficult
 - Angles are not precise depending upon slop
 - Average distance traveled is accurate

Other Considerations

- Yaw (direction) best handled with Gyro
- Velocity control, using distance traveled and Gyro heading works really well
- Gyro heading errors * gain +/- left right velocity (simple P controller on velocity)
- Gyros drift, Use compass to correct, or only use over short periods of time.