Proportional Braking / Reverse Driving Transition

Robert Atkinson
7 August 2018

We compute where the proportional braking / reverse driving transition should lie

1. Administrivia

1.1. Load-In

We load output we previously computed.

```
Get[NotebookDirectory[] <> "Utilities.m"]
```

```
inputDirectory = FileNameJoin[{NotebookDirectory[], "MotorPhysicsGears.Output"}] <> $PathnameSeparator
```

```
C:\ftc\RobotPhysics\MotorPhysicsGears.Output\
```

```
Get[inputDirectory <> "ParametersUnitsAndAssumptions.m"];
$Assumptions = parameterAssumptions
```

```
{Bafter ∈ ℝ, constvapp ∈ ℝ, constτappafter ∈ ℝ, Jafter ∈ ℝ, i[_] ∈ ℝ, vapp[_] ∈ ℝ, vg[_] ∈ ℝ, α[_] ∈ ℝ,
 θ[_] ∈ ℝ, τ[_] ∈ ℝ, τappafter[_] ∈ ℝ, ω[_] ∈ ℝ, Ke > 0, Kt > 0, L > 0, R > 0, η > 0, ℕ > 0, B ≥ 0, J ≥ 0, t ≥ 0}
```

```
Get[inputDirectory <> "MotorModels.m"];
```

```
Get[inputDirectory <> "MotorTimeDomainFunctions.m"];
```

```
Get[inputDirectory <> "SteadyState.m"];
```

```
Get[inputDirectory <> "Misc.m"];
```

2. Locating the Transition Point

2.1. General

In the steady state (no acceleration), the back EMF is proportional to the steady state velocity Ω (in radians per second).

```
backEmfEqns = {ssEmf == Ke Ω}
```

```
{ssEmf == Ke Ω}
```

2.2. Proportional Braking

We need to know the beginning and endpoints of the proportional braking curve, but here need not assume it is linear.

0% proportional braking is 100% H-bridge state 0. In that state, there is zero torque, as we have an open circuit. In order to determine the transition between proportional braking and reverse driving, rather than dealing with open circuits we will determine for each situation of relevance an equivalent instantaneous DC applied voltage for an H-bridge-less variant of the motor. For the zero-torque situation, we need to examine our voltage differential equation.

```
diffEqns[[3]]
```

```
R i[t] + vg[t] + L i′[t] == vapp[t]
```

If we’re in steady state at velocity Ω, and we instantaneously drop the voltage, to what level should we drop it so that the torque (and thus the current) as averaged over our PID loop execution interval (50ms) is zero? We haven’t yet completed that analysis, so for now we’ll use an approximation. If there is zero torque, there is zero current. Which means that drop across the resistor is zero. After a couple of time constants, the drop across the inductor will also be zero, and we’re left with:

```
diffEqns[[3]] /. {i[t] → 0, i'[t] → 0}
```

```
vg[t] == vapp[t]
```

That is, the voltage we should drop our applied voltage vappt[t] to is (approximately) the current back EMF. This is only approximate, as it doesn’t account for the exponential ramp down of the current, nor does it consider the deceleration of the motor (and consequent drop in the back EMF) that will happen over the PID loop interval, but it’s the best we have on hand for now.

We recall that this is happening at 0% proportional braking, which we wish to be coincident to the origin.

```
prop0Eqns = { controlProp0 == 0, voltProp0 == ssEmf }
```

```
{controlProp0 == 0, voltProp0 == ssEmf}
```

At 100% proportional braking, we have shorted across the motor terminals; the instantaneous applied voltage is zero.

```
prop100Eqns = {voltProp100 == 0}
```

```
{voltProp100 == 0}
```

We define a linear curve for proportional braking, though in the below we only use it for plotting, not in the determination of the transition point.

```
voltProp[control_] := voltProp0 + (voltProp100 - voltProp0) / (controlProp100 - controlProp0) * (control - controlProp0)
voltProp[controlProp100]
voltProp[controlProp0]
```

```
voltProp100
```

```
voltProp0
```

2.3. Reverse Driving

0% reverse driving is almost the same as 100% proportional braking except that there’s a diode eating some of the voltage alongside the resistor and the inductor. Thus, going back to our differential equation above, the instantaneous equivalent applied voltage needs to be adjusted slightly positive to balance.

```
reverse0Eqns = { voltReverse0 == -(0 - diodeDrop) }
```

```
{voltReverse0 == diodeDrop}
```

100% reverse driving applies the full negative battery voltage:

```
reverse100Eqns = { controlReverse100 == -controlMax, voltReverse100 == -batteryVoltage }
```

```
{controlReverse100 == -controlMax, voltReverse100 == -batteryVoltage}
```

2.4. Continuity

We will assume linearity of reverse driving in order to set up our continuity of voltage (we only use that fact in the initial duty cycle range of reverse driving).

```
voltReverse[control_] :=
 voltReverse0 + (voltReverse100 - voltReverse0) / (controlReverse100 - controlReverse0) * (control - controlReverse0)
voltReverse[controlReverse100]
voltReverse[controlReverse0]
```

```
voltReverse100
```

```
voltReverse0
```

Because of the diode voltage drop, when we stitch proportional braking beside reverse driving so as to have a continuous overall voltage curve, there is necessarily a small overlap in the two curves: as we proceed from small to large control signal *magnitude* (remember that proportional braking and reverse driving are placed in the negative control signal range), the reverse driving curve begins before the proportional braking curve ends. That is, Abs[controlReverse0] <= Abs[controlProp100].

Here, we choose to use braking in the overlap. Thus, here our transition point is controlProp100.

```
controlTransition = controlProp100;
controlOther = controlReverse0;
```

Our continuity requirement is that at the transition, the two curves must coincide.

```
ctyEqns = { voltProp[controlTransition] == voltReverse[controlTransition] }
```

```
{voltProp100 ==
  voltReverse0 + ((controlProp100 - controlReverse0) (-voltReverse0 + voltReverse100)) / (-controlReverse0 + controlReverse100)}
```

2.5. Linearity

All other things being equal, what we want is that the overall voltage vs control signal in the negative control range be as linear as possible. The shapes of the two curves, and the voltages at their endpoints is fixed; all we have to control is the transition threshold. We assume that the duty cycle response curves of both modes is linear. (While they're not entirely so, they're close, and we need to start somewhere. If we actually knew the curves we'd do a least squares fit.) Then, the overall concatenation can best be fit as linear by matching the slopes of the proportional braking as a whole and the reverse driving as a whole.

```
slopeEqns = {
  slope == (voltProp100 - voltProp0) / (controlProp100 - controlProp0),
  slope == (voltReverse100 - voltReverse0) / (controlReverse100 - controlReverse0)
 }
```

$$\left\{\text{slope} == \frac{-\text{voltProp0} + \text{voltProp100}}{-\text{controlProp0} + \text{controlProp100}},\ \text{slope} == \frac{-\text{voltReverse0} + \text{voltReverse100}}{-\text{controlReverse0} + \text{controlReverse100}}\right\}$$

2.6. Summary

We put all our equations together.

```
(eqns = Join[backEmfEqns, prop0Eqns, prop100Eqns, reverse0Eqns, reverse100Eqns, ctyEqns, slopeEqns]) // prettyPrint
```

ssEmf = Ke Ω
controlProp0 = 0
voltProp0 = ssEmf
voltProp100 = 0
voltReverse0 = diodeDrop
controlReverse100 = −controlMax
voltReverse100 = −batteryVoltage

$$\text{voltProp100} = \text{voltReverse0} + \frac{(\text{controlProp100}-\text{controlReverse0})\,(\text{voltReverse100}-\text{voltReverse0})}{\text{controlReverse100}-\text{controlReverse0}}$$

$$\text{slope} = \frac{\text{voltProp100}-\text{voltProp0}}{\text{controlProp100}-\text{controlProp0}}$$

$$\text{slope} = \frac{\text{voltReverse100}-\text{voltReverse0}}{\text{controlReverse100}-\text{controlReverse0}}$$

2.7. Solving

We solve these equations.

```
solveFor = {controlTransition, controlOther, slope, voltProp0,
   voltProp100, voltReverse0, voltReverse100, controlProp0, controlReverse100, ssEmf};
soln = uniqueSolve[eqns, solveFor];
soln // prettyPrint
```

controlProp100 → $-\frac{\text{controlMax Ke }\Omega}{\text{batteryVoltage+Ke }\Omega}$

controlReverse0 → $\frac{\text{controlMax diodeDrop}-\text{controlMax Ke }\Omega}{\text{batteryVoltage+Ke }\Omega}$

slope → $\frac{\text{batteryVoltage+Ke }\Omega}{\text{controlMax}}$

voltProp0 → Ke Ω
voltProp100 → 0
voltReverse0 → diodeDrop
voltReverse100 → –batteryVoltage
controlProp0 → 0
controlReverse100 → –controlMax
ssEmf → Ke Ω

The control transition point we seek is the following:

```
controlTransition /. soln
```

$-\frac{\text{controlMax Ke }\Omega}{\text{batteryVoltage + Ke }\Omega}$

3. Verification

3.1. Check

Let us verify that at the control transition point, the two curves coincide.

```
voltProp[controlTransition] - voltReverse[controlTransition]
% /. soln
% // FullSimplify
```

```
voltProp100 - voltReverse0 -
  ((controlProp100 - controlReverse0) (-voltReverse0 + voltReverse100)) / (-controlReverse0 + controlReverse100)
```

$-\text{diodeDrop} - \left((-\text{batteryVoltage} - \text{diodeDrop})\left(-\frac{\text{controlMax Ke }\Omega}{\text{batteryVoltage + Ke }\Omega} - \frac{\text{controlMax diodeDrop} - \text{controlMax Ke }\Omega}{\text{batteryVoltage + Ke }\Omega}\right)\right) \Big/ \left(-\text{controlMax} - \frac{\text{controlMax diodeDrop} - \text{controlMax Ke }\Omega}{\text{batteryVoltage + Ke }\Omega}\right)$

```
0
```

3.2. In Real Life

We examine what this looks like for an actual motor. We choose a NeveRest 60. We ignore the gearing since we want the Ke relative to the pre-gearbox armature velocity, not the post-gearbox shaft velocity.

```
aMotor = motorParameters["AM 60 A", Geared → False] // siUnits // clearUnits
constants = {diodeDrop → 7/10, Ke → (Ke /. aMotor), batteryVoltage → 12, controlMax → 32767}
```

$\langle\left| R \to \frac{33}{10}, L \to \frac{347}{500000}, Ke \to \frac{533}{500}, Kt \to \frac{533}{500}, J \to \frac{1041}{100000000}, B \to \frac{33}{1000}, N \to 1, \eta \to 1, Jafter \to 0, Bafter \to 0, const\tau appafter \to 0 \right|\rangle$

$\left\{diodeDrop \to \frac{7}{10}, Ke \to \frac{533}{500}, batteryVoltage \to 12, controlMax \to 32767\right\}$

There are three points of relevance to us.

```
points = {
  Z → {controlProp0, voltProp0},
  T → {controlProp100, voltProp100},
  R → {controlReverse100, voltReverse100}
 }
```

```
{Z → {controlProp0, voltProp0}, T → {controlProp100, voltProp100}, R → {controlReverse100, voltReverse100}}
```

The maximum velocity of this motor is just over 10 radians / second.

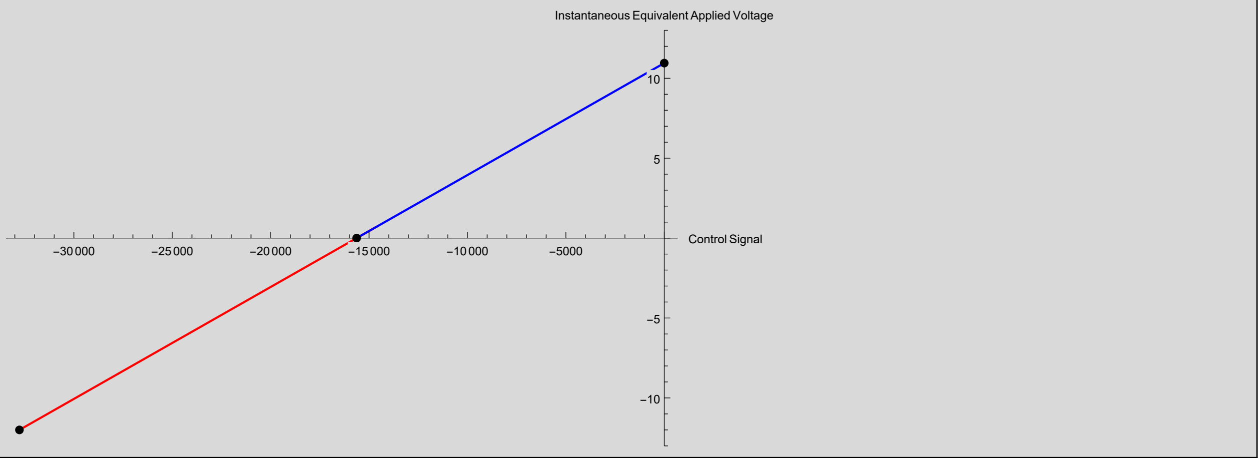
```
ssVel /. ss /. aMotor /. constvapp → batteryVoltage /. constants // N
```

```
10.2726
```

At that velocity, what part of the negative control range is devoted to proportional braking?

```
speed = {Ω → 10.27}
all = Join[soln, constants, speed];
p1 = Plot[Evaluate[voltReverse[control] //. all], {control, controlReverse100, controlReverse0} //. all,
    PlotStyle → Red, AxesOrigin → {0, 0}, PlotRange → Evaluate[{-batteryVoltage - 1, batteryVoltage + 1} //. all],
    AxesLabel → {HoldForm[Control Signal], HoldForm[Instantaneous Equivalent Applied Voltage]},
    ImageSize → Large];
p2 = Plot[Evaluate[voltProp[control] //. all], {control, controlProp100, controlProp0} //. all, PlotStyle → Blue];
p3 = ListPlot[{Z, T, R} /. (points //. all), AxesOrigin → {0, 0}, PlotStyle → Black];
Show[p1, p2, p3]
```

```
{Ω → 10.27}
```



Answer: about half of it. Good. It will be less for lesser velocities, but still a good chunk.

4. Revision History

- 2018.08.07 Initial version.