ANSWERS

U10.2 Velocity profiles

Following a trajectory · University · about 30 min

What this lesson is about

The trapezoid, the triangle when there is no room to reach cruise, and what jerk costs you.

- The trapezoid
- The BugBot has no acceleration limit
- Jerk
- Choosing the numbers

Questions

7 marks in all

1. A trapezoidal move covers 80 cm with an acceleration limit of 10 cm/s/s and a cruise speed of 15 cm/s. How long is the cruise (flat top) phase, in seconds to 2 decimal places? [1 mark]

> **Answer:** 3.83 (accept within 0.01). t_acc = 15 / 10 = 1.5 s and d_acc = 15 x 15 / (2 x 10) = 11.25 cm. The flat top covers 80 - 22.5 = 57.5 cm at 15 cm/s, which is 3.83 s.

2. What does this print? It works out the whole duration of the same profile. [1 mark]

```
D, A, V = 80.0, 10.0, 15.0
t_acc = V / A
d_acc = V * V / (2 * A)
t_flat = (D - 2 * d_acc) / V
print(round(2 * t_acc + t_flat, 2))
```

> **Answer:**
>
> ```
> 6.83
> ```
>
> Two ramps of 1.5 s each plus a 3.83 s cruise gives 6.83 s.

3. The same limits (10 cm/s/s, 15 cm/s) are used for an 8 cm move. What peak speed does the profile reach, in cm/s to 2 decimal places? [1 mark]

> **Answer:** 8.94 (accept within 0.01). Two ramps to 15 cm/s would need 22.5 cm, more than the 8 cm available, so the profile is a triangle with peak sqrt(a d) = sqrt(10 x 8) = 8.94 cm/s.

4. A trapezoid generator works on long moves, but every short move overshoots. What is the most likely bug? [1 mark]

- ☐ **A** It always assumes a cruise phase, even when 2 d_acc does not fit inside the distance
- ☐ **B** The acceleration limit is too low
- ☐ **C** It samples the profile too often
- ☐ **D** The deceleration ramp uses a different a from the acceleration ramp

> **Answer:** A. When the ramps cannot fit, the flat top works out negative and the profile promises distance it has no room for. That case must become a triangle.

5. The BugBot's drive is a first order lag with a time constant of 0.25 s. Roughly what initial acceleration does a step command to 20 cm/s produce, in cm/s/s? [1 mark]

> **Answer:** 80 (accept within 1). A first order lag starts towards its target at (target / tau): 20 / 0.25 = 80 cm/s/s. Nothing in the hardware limits it.

6. The drive has no acceleration limit of its own. Which are good reasons to impose one anyway? [1 mark]

Tick every answer that is true.

- ☐ **A** Demanding more than friction supplies makes the robot slip, which dead reckoning cannot see
- ☐ **B** A step demand is a current spike that can sag the battery and reset the robot
- ☐ **C** A profile the machine can follow leaves an error you can reason about
- ☐ **D** It makes the move finish sooner than a step command would
- ☐ **E** The drive rejects step commands above 10 cm/s

> **Answer:** A, B, C. Traction, current and predictability are all reasons. A limit never makes a move faster, and the drive accepts any step.

7. A robot arm rings for a moment at the start and end of every trapezoidal move. What change addresses the cause? [1 mark]

- ☐ **A** Limit the jerk with an S-curve so the acceleration ramps instead of switching instantly
- ☐ **B** Raise the cruise speed so the move is over sooner
- ☐ **C** Use a triangle profile instead of a trapezoid
- ☐ **D** Sample the trapezoid at a finer time step

> **Answer:** A. A trapezoid's acceleration steps at each corner, which is infinite jerk and excites anything springy. An S-curve ramps the acceleration at a small cost in duration.

The task: a trapezoidal profile

Build a trapezoidal profile that covers 80 cm with an acceleration limit of 10 cm/s/s and a cruise speed of 15 cm/s, sampled every 0.05 s. Plot `v` and `s` at every sample, with a `wait(0.1)` after every second one so the chart has a time axis (the simulator runs in 0.02 s steps, so a `wait(0.05)` would last 0.04 s), and print `peak:`, `duration:` and `distance:`, the last being the area under your own speed profile.

```
from bugbot import *
connect()

DISTANCE, A_MAX, V_MAX, DT = 80.0, 10.0, 15.0, 0.05
```

The hint students can ask for: Reaching the cruise speed takes v/a seconds and covers v squared over 2a. If two of those fit inside the distance there is a flat top in the middle; if they do not, the profile is a triangle and the peak is lower than the cruise speed. Integrate your own profile to check it.

A solution

```
from bugbot import *
connect()

DISTANCE, A_MAX, V_MAX, DT = 80.0, 10.0, 15.0, 0.05

t_acc = V_MAX / A_MAX                       # how long it takes to reach cruise
d_acc = 0.5 * A_MAX * t_acc * t_acc         # and how far that takes
if 2 * d_acc > DISTANCE:
    # no room for cruise: a triangle, with a peak lower than V_MAX
    t_acc = (DISTANCE / A_MAX) ** 0.5
    peak, t_flat = A_MAX * t_acc, 0.0
else:
    peak = V_MAX
    t_flat = (DISTANCE - 2 * d_acc) / V_MAX
duration = 2 * t_acc + t_flat

def speed(t):
    if t < 0 or t > duration:
        return 0.0
    if t < t_acc:
        return A_MAX * t
    if t < t_acc + t_flat:
        return peak
    return max(0.0, peak - A_MAX * (t - t_acc - t_flat))

n = int(round(duration / DT))
step = duration / n
s = 0.0
for i in range(n):
    v = speed((i + 0.5) * step)             # the middle of the step, so the corners
integrate cleanly
    s += v * step
    plot("v", v)
    plot("s", s)
    if i % 2 == 1:
        wait(0.1)

print("peak:", round(peak, 2))
```

```
print("duration:", round(duration, 2))
print("distance:", round(s, 1))
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.