ANSWERS

U1.6 Measuring your robot

The robot as a system · University · about 30 min

What this lesson is about

A step response: top speed, time constant and dead band, measured rather than assumed.

- The experiment
- The two numbers
- The third number
- Doing it honestly

Questions

6 marks in all

1. A first order system follows v(t) = v_max * (1 - exp(-t / tau)). This prints the fraction of the final value reached after 1, 2 and 3 time constants. What does it print? [1 mark]

```
import math
for n in (1, 2, 3):
    print(n, round(1 - math.exp(-n), 3))
```

Answer:

```
1 0.632
2 0.865
3 0.95
```

At t = tau the bracket is 1 - 1/e = 0.632, which is where the 63 percent rule comes from. After 3 tau it is 95 percent of the way, the practical meaning of "there".

2. A step response settles at 20 cm/s with a time constant of 0.25 s. What is the speed at t = 0.25 s, in cm/s to one decimal place? [1 mark]

Answer: 12.6 (accept within 0.1). At t = tau the speed is 63.2 percent of the final value: 0.632 × 20 = 12.6 cm/s.

3. Step response readings are taken every 0.1 s from the moment of the step. This estimates v_max from the last five readings and tau from the first reading at or above 63.2 percent of it. What does it print? [1 mark]

```
speeds = [0.0, 6.8, 11.5, 14.5, 16.6, 18.0, 18.9, 19.3, 19.7, 19.9, 20.1, 19.9, 20.0, 20.1,
19.9]
dt = 0.1
v_max = sum(speeds[-5:]) / 5
target = 0.632 * v_max
for i, v in enumerate(speeds):
    if v >= target:
        break
print("v_max:", round(v_max, 1))
print("tau:", round(i * dt, 1))
```

Answer:

```
v_max: 20.0
tau: 0.3
```

The last five average to 20.0, so the target is 12.64. The first reading above it is 14.5 at index 3, so tau prints as 0.3. Interpolating between 11.5 at 0.2 s and 14.5 at 0.3 s puts the real crossing nearer 0.24 s, so a 10 Hz sample rate is coarse for a quarter-second time constant.

4. The time constant is 0.25 s. How long after the step, in seconds, should you wait before readings measure the settled level rather than the rise? [1 mark]

Answer: 0.75 (accept within 0.01). After 3 tau the response is 95 percent of the way there: 3 × 0.25 = 0.75 s.

5. The dead band is 15. A controller's output spends most of its time around 10. What does the robot do? [1 mark]

- ☐ **A** Nothing, which looks exactly like a bug in the controller's arithmetic
- ☐ **B** Moves at 10 percent of its top speed
- ☐ **C** Moves slowly, but only after the lag
- ☐ **D** Moves for half a second and then the deadman stops it

Answer: A. Below the dead band the drive buzzes and does not move. It is a property of the actuator to be designed around, not a software fault.

6. Which of these are part of measuring a step response honestly? [1 mark]

Tick every answer that is true.

- ☐ **A** Repeat the run and report the spread
- ☐ **B** Ignore readings taken during the first 3 tau when finding the level
- ☐ **C** Record the battery level at the time
- ☐ **D** Take the single last reading as v_max, since it is the most settled
- ☐ **E** Use the catalogue value when your runs disagree

Answer: A, B, C. One run is an anecdote, the transient is not the level, and the numbers depend on conditions. Each reading carries noise, so the level is an average of several, and the catalogue is not your robot.

The task: the step response

Run a step to full forward power, plot the speed as `speed`, and print two lines: `v_max:` and `tau:`.

```
from bugbot import *
connect()

speeds = []
forward(100)
```

The hint students can ask for: Ask for full speed from a standing start, and read flow() every tenth of a second into a list. The settled speed is the average of the last few; the time constant is when it first crossed 63 percent of that.

A solution

```
from bugbot import *
connect()

speeds = []
forward(100)
for i in range(30):
    v = flow()[1]
    speeds.append(v)
    plot("speed", v)
    wait(0.1)
stop()

v_max = sum(speeds[-8:]) / 8
tau = 3.0
for i, v in enumerate(speeds):
    if v >= 0.63 * v_max:
        tau = (i + 1) * 0.1
        break
print("v_max:", round(v_max, 1))
print("tau:", round(tau, 2))
```

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