ANSWERS

U10.8 Time scaling a path

Following a trajectory · University · about 40 min

What this lesson is about

Cubic and quintic time laws and what smoothness costs, the speed limit that depends on direction (this robot is fastest on the diagonal), and the time-optimal speed along a path.

- A time law as a function
- What the robot makes of them
- The speed limit depends on direction
- Time-optimal along a path
- Driving the plan

The task: dash to the corner

The far corner of the mat is about two metres away, on the diagonal. Be in it by 8.8 seconds and stay there until the clock passes 11, then let the program end; the run is cut off at 12. Facing the corner and driving flat out gets there after about 11 seconds on this robot, which is too slow. Use the corner of the velocity box instead: work out which way to face so that the robot's fastest direction points at the corner, turn to face it, and go. Even a perfect aim will not stay perfect, because the vibration drive turns the robot slowly as it goes, and leaks a little of its forward push sideways. So steer as you drive: compare the direction the robot is actually moving with the direction to the target, and turn a little to close the gap. Once it steers, the steering also soaks up a few degrees of error in the aim, so the data sheet's speeds will do nearly as well as your own. They matter if you drive open loop, without steering: they put the corner of the box at 37 degrees rather than this robot's 41, and a few degrees of aim is 10 to 15 cm at two metres. The figure shows the two runs against the clock.

```
from bugbot import *
import math
connect()

V_FWD, V_SIDE = 20.0, 15.0                  # replace with the speeds you measured
START, TARGET = (30.0, 30.0), (170.0, 170.0)

def here():
    px, py = position()
    return START[0] + px, START[1] + py
```

The hint students can ask for: Work out the angle of the corner of this robot's velocity box from the two speeds you measured, and turn so that angle points at the far corner of the mat. The robot will not hold that line by itself, so as it drives, keep comparing the direction it is actually moving with the direction to the target and turn a little to close the gap. Stop in the corner and stay there until the clock passes 11 seconds, then let the program end: the run is cut off at 12.

A solution

```
from bugbot import *
import math
connect()

V_FWD, V_SIDE = 18.3, 16.0                  # this robot's speeds at 100 percent, measured
in the second cell
START, TARGET = (30.0, 30.0), (170.0, 170.0)

def here():
    px, py = position()
    return START[0] + px, START[1] + py

def bearing(a, b):                          # degrees clockwise from up the mat, from
point a to point b
    return math.degrees(math.atan2(b[0] - a[0], b[1] - a[1]))

best = math.degrees(math.atan2(V_SIDE, V_FWD))
print("fastest direction:", round(best, 1), "degrees right of forward, at",
round(math.hypot(V_FWD, V_SIDE), 1), "cm/s")
turn_right(60, angle=bearing(START, TARGET) - best)   # point the corner of the box at the
target

drive(100, 100, 0)
```

```
wait(0.5)                                   # past the lag
prev = here()
while math.hypot(TARGET[0] - here()[0], TARGET[1] - here()[1]) > 12:
    wait(0.1)
    now = here()
    err = (bearing(now, TARGET) - bearing(prev, now) + 180) % 360 - 180   # the way to go,
less the way it is going
    prev = now
    rot = 0.0
    if abs(err) > 1:                        # turn to close the gap, with at least 16
percent to get past the dead band
        rot = math.copysign(max(16, min(30, 8 * abs(err))), err)
    drive(100, 100, rot)
stop()
print("in the corner after", round(clock(), 1), "s at", tuple(round(c) for c in here()))
while clock() < 11:                         # stay until the check is done, and finish
before 12 s
    wait(0.1)
```

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