ANSWERS

U2.8 Driving like a car

Kinematics and frames · University · about 35 min

What this lesson is about

The unicycle and the car, the velocity constraint that makes them non-holonomic, the bracket that gets a car sideways anyway, and parallel parking.

- Three robots on one mat
- The constraint is on velocity, not on position
- Getting sideways without going sideways
- Parallel parking
- Why the BugBot does not bother

The task: parallel park

The bay is to the robot's right, between two parked cars. All the distances are from the robot's starting point, facing along `+y` (heading 0). - **The bay** is 18 cm wide and 47 cm long. Across, it runs from 13 cm to 31 cm to your right; along, from 10 cm behind you to 37 cm ahead. Its middle is 22 cm to your right and 13.5 cm ahead. - **The parked cars** are 14 cm wide and 22 cm long, from 16 cm to 30 cm to your right. The one behind runs from 35 cm to 13 cm behind you; the one ahead from 42 cm to 64 cm ahead. - **The robot** is 7 cm across, and touching a car counts from its edge, not its centre. To pass, the robot's centre must finish inside the bay, facing within 15 degrees of the way it started (a final heading between 345 and 15), without touching either car, inside 40 seconds. Use `car()`. The checker ignores how you call the motors and checks that the robot never slides, never turns on the spot and never turns tighter than `R_MIN`. Any speed up to 100 percent is allowed, forwards or in reverse.

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

V_MAX, W_MAX, R_MIN = 20.0, 120.0, 25.0

def car(speed, steer, seconds):
    """speed -100..100 (negative reverses), steer -1..1 (full lock left..right). Turn rate
is tied to speed."""
    speed = max(-100, min(100, speed))
    steer = max(-1, min(1, steer))
    v = speed / 100 * V_MAX
    w = math.degrees(v / R_MIN) * steer
    drive(speed, 0, 100 * w / W_MAX)
    wait(seconds)
    stop()
    wait(0.4)

# your moves, all with car()
```

The hint students can ask for: Work out, from the numbers in the task, how far sideways and how far along the bay's middle is from where you start. The two-arc formulas turn the sideways distance into the angle each arc must turn, and that angle into the length of road the arcs use; so you know how far forward to go before reversing. Then turn each distance into seconds from the car's speed. The forward leg passes well clear of the parked cars; the tight moment is where the lock changes, when the first arc has swung the robot in close to the back corner of the car ahead, so check how much room the arcs leave there.

A solution

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

V_MAX, W_MAX, R_MIN = 20.0, 120.0, 25.0

def car(speed, steer, seconds):
    """speed -100..100 (negative reverses), steer -1..1 (full lock left..right). Turn rate
is tied to speed."""
    speed = max(-100, min(100, speed))
    steer = max(-1, min(1, steer))
    v = speed / 100 * V_MAX
    w = math.degrees(v / R_MIN) * steer
```

```
    drive(speed, 0, 100 * w / W_MAX)
    wait(seconds)
    stop()
    wait(0.4)

# aim 22 cm to the right and 11 cm ahead: the middle of the bay across, a little behind its
middle along (away from the car ahead)
SIDE, AHEAD = 22, 11
v = 60 / 100 * V_MAX                                # cm/s at 60 percent
phi = math.acos(1 - SIDE / (2 * R_MIN))             # each arc turns this far: 2 R (1 - cos
phi) = SIDE
back = 2 * R_MIN * math.sin(phi)                    # and the two arcs together take this
much road
arc = R_MIN * phi / v                               # seconds for one arc

car(60, 0, (AHEAD + back) / v)                      # past the bay, far enough to reverse
into it
car(-60, 1, arc)                                    # swing the back in
car(-60, -1, arc)                                   # straighten up
print("phi", round(math.degrees(phi)), "arc", round(arc, 2), "s, then", round((AHEAD +
back) / v, 2), "s")
```

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