ANSWERS

U2.6 Go to a point

Kinematics and frames · University · about 25 min

What this lesson is about

The first useful controller: a vector to the target, straight through the inverse kinematics.

- What is going on
- Where the position came from
- Stopping

Questions

6 marks in all

1. The go-to-point controller sets its speed from the gap to the target. What does this print? [1 mark]

```
for gap in (60.0, 28.0, 10.0, 3.0):
    speed = min(14.0, max(4.0, 0.5 * gap))
    print(gap, speed, "stop" if gap < 4 else "drive")
```

Answer:

```
60.0 14.0 drive
28.0 14.0 drive
10.0 5.0 drive
3.0 4.0 stop
```

Far away 0.5 × gap exceeds the ceiling, so speed is 14. Closer in it eases off in proportion, 5.0 at 10 cm, but never below the floor of 4 cm/s, which keeps the command out of the drive's dead band. Inside 4 cm the loop stops.

2. The target is 30 cm east and 40 cm north of the robot. What does this print? [1 mark]

```
import math
dx, dy = 30.0, 40.0
gap = math.hypot(dx, dy)
speed = min(14.0, max(4.0, 0.5 * gap))
wx, wy = speed * dx / gap, speed * dy / gap
print(gap, speed, round(wx, 1), round(wy, 1))
```

Answer:

```
50.0 14.0 8.4 11.2
```

The gap is 50 cm, so speed is capped at 14. The unit vector is (0.6, 0.8), giving a world velocity of (8.4, 11.2), which is 14 cm/s towards the target.

3. A student writes wx, wy = speed * dx, speed * dy, leaving out the division by gap. What goes wrong? [1 mark]

☐ **A** The effective gain depends on how far away the target is, so the robot behaves differently on long and short runs

☐ **B** The robot heads in the wrong direction

☐ **C** Nothing, because speed is already capped at 14

☐ **D** The robot can never get within 4 cm, because the velocity is always zero

> **Answer:** A. Dividing by gap makes a unit vector, direction without magnitude. Without it the magnitude is speed × gap, so the direction is right but the size grows with distance.

4. How should the stopping threshold, gap < 4, be chosen? [1 mark]

☐ **A** Larger than the distance the robot coasts from the speed it arrives at, or it sails past and hunts

☐ **B** As small as possible, since a smaller threshold is always more accurate

☐ **C** Equal to the robot's radius

☐ **D** Smaller than the noise on position(), so noise cannot trigger a stop

> **Answer:** A. If the robot is still moving fast enough to coast further than the threshold, it overshoots, turns round and comes back. Measure the coast, then pick the number, or ramp the speed down.

5. The robot starts at world (60, 60), and position() counts from where it started. How far is a target at world (100, 45) from the start, in cm to one decimal place? [1 mark]

> **Answer:** 42.7 (accept within 0.05). Relative to the start the target is at (40, -15), and the square root of $40^2 + 15^2$ is 42.7 cm.

6. This controller reads position(), which U1.3 called the lab's overhead camera. Why is that acceptable here? [1 mark]

☐ **A** The subject is kinematics for now, and from U6 an estimate the robot works out for itself replaces it under the same loop

☐ **B** position() is what a real robot would use for this controller

☐ **C** The inverse kinematics only works with a perfect position

☐ **D** odometry() is in the body frame, so it cannot be used with world targets

> **Answer:** A. The loop is written cleanly now so that later estimators plug straight into it.

The task: go to a point

Drive to world `(140, 150)` and stop within 10 cm of it. The robot starts at `(60, 60)` facing the wrong way, and `position()` counts from where it started.

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

V_MAX, V_LAT = 20.0, 15.0
TX, TY = 80.0, 90.0        # the target, in cm from the start
```

The hint students can ask for: The target is at world (140, 150) and the robot starts at (60, 60) facing the wrong way. position() is relative to the start, so the target is 80 across and 90 up from where you began. Each tick: the vector to the target, rotated into the body frame, scaled down as you close in.

A solution

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

V_MAX, V_LAT = 20.0, 15.0
TX, TY = 80.0, 90.0        # the target, in cm from where the robot started

for tick in range(400):
    x, y = position()
    dx, dy = TX - x, TY - y
    gap = math.hypot(dx, dy)
    if gap < 4:
        break
    speed = min(14.0, max(4.0, 0.5 * gap))
    wx, wy = speed * dx / gap, speed * dy / gap
    a = math.radians(heading())
    bx = wx * math.cos(a) - wy * math.sin(a)
    by = wx * math.sin(a) + wy * math.cos(a)
    drive(100 * by / V_MAX, 100 * bx / V_LAT, 0)
    wait(0.1)
stop()
```

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