ANSWERS

U2.2 The rotation matrix

Kinematics and frames · University · about 30 min

What this lesson is about

Turning a vector from one frame into the other, in two dimensions, by hand and in code.

- From body to world
- Checking it without trusting it
- In code
- The other way

Questions

6 marks in all

1. A robot at heading 60 measures (0, 10) in its own frame. What does this print? [1 mark]

```
import math

def body_to_world(vx, vy, h_deg):
    a = math.radians(h_deg)
    return (vx * math.cos(a) + vy * math.sin(a),
            -vx * math.sin(a) + vy * math.cos(a))

wx, wy = body_to_world(0, 10, 60)
print("wx:", round(wx, 2))
print("wy:", round(wy, 2))
```

Answer:

```
wx: 8.66
wy: 5.0
```

wx = 10 sin 60 = 8.66 and wy = 10 cos 60 = 5.0. Ten centimetres ahead of a robot turned 60 degrees clockwise is mostly to the right and partly away.

2. A robot at heading 90 slides 10 cm to its own right, the body vector (10, 0). What does this print? [1 mark]

```
import math
a = math.radians(90)
vx, vy = 10.0, 0.0
wx = vx * math.cos(a) + vy * math.sin(a)
wy = -vx * math.sin(a) + vy * math.cos(a)
print(round(wx, 2), round(wy, 2))
```

Answer:

```
0.0 -10.0
```

wx = 10 cos 90 = 0 and wy = -10 sin 90 = -10. A robot facing +x has its right hand pointing along -y, towards you.

3. A student computes 10 * math.sin(90) for a robot facing +x and gets 8.94 instead of 10. What is wrong? [1 mark]

- ☐ **A** math.sin takes radians, so 90 was treated as 90 radians; it needs math.radians(90)
- ☐ **B** The x component should use cos, not sin
- ☐ **C** The minus sign belongs on this term in a clockwise convention
- ☐ **D** The vector is in the world frame and needs rotating back first

> **Answer:** A. Every angle in the API is in degrees and Python's trigonometry is in radians. Forgetting math.radians() is the commonest way to get a rotation wrong.

4. Which matrix undoes R(h), turning a world vector back into the body frame? [1 mark]

- ☐ **A** R(-h), which is also the transpose of R(h)
- ☐ **B** R(h) itself, applied a second time
- ☐ **C** R(h + 180)
- ☐ **D** The matrix with each entry of R(h) replaced by its reciprocal

> **Answer:** A. The inverse of a rotation is the same rotation the other way, and for a rotation matrix that is just the transpose. No division is ever needed.

5. In R(h) as the lesson writes it, the minus sign sits on -sin h below the diagonal. Why there? [1 mark]

- ☐ **A** Because heading is clockwise positive; in an anticlockwise convention it sits above the diagonal
- ☐ **B** Because Python's trigonometric functions work in radians
- ☐ **C** Because the body's y axis points forward rather than right
- ☐ **D** Because a rotation matrix always has its minus sign below the diagonal

> **Answer:** A. Reversing the sense of positive angle replaces h with -h, which moves the minus sign across the diagonal. Checking the h = 90 case against the robot catches the mistake.

6. This rotates the body vector (3, 4) into the world at heading 37 and back again. What does it print? [1 mark]

```
import math

def body_to_world(vx, vy, h):
    a = math.radians(h)
    return vx * math.cos(a) + vy * math.sin(a), -vx * math.sin(a) + vy * math.cos(a)

def world_to_body(wx, wy, h):
    a = math.radians(h)
    return wx * math.cos(a) - wy * math.sin(a), wx * math.sin(a) + wy * math.cos(a)

wx, wy = body_to_world(3, 4, 37)
bx, by = world_to_body(wx, wy, 37)
print(round(math.hypot(wx, wy), 6), round(bx, 6), round(by, 6))
```

> **Answer:**
>
> ```
> 5.0 3.0 4.0
> ```
>
> A rotation never changes length, so the world vector is still 5 long, and rotating back recovers (3, 4). Both are cheap checks that the maths is right.

The task: rotate a vector

A robot at heading 60 measures `(0, 10)` in its own frame. Print the same vector in the world frame, as `wx:` and `wy:`, worked out with the rotation matrix.

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

h = 60
vx, vy = 0.0, 10.0
```

The hint students can ask for: A robot at heading 60 measures (0, 10) in its own frame: ten centimetres straight ahead. World x is vx*cos(h) + vy*sin(h); world y is -vx*sin(h) + vy*cos(h). The angle is in degrees.

A solution

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

h = 60
vx, vy = 0.0, 10.0
a = math.radians(h)
wx = vx * math.cos(a) + vy * math.sin(a)
wy = -vx * math.sin(a) + vy * math.cos(a)
print("wx:", round(wx, 3))
print("wy:", round(wy, 3))
```

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