WORKSHEET

U11.6 Visual servoing

BugBotLab

Vision · University · about 35 min

Name _____ Class _____ Date _____

What this lesson is about

Closing the loop on pixels instead of converting to metres first, and why that is more robust.

- Two families
- The two loops on this robot
- The interaction matrix
- Gains, dead bands and losing sight

Questions

7 marks in all

1. What distinguishes image based visual servoing (IBVS) from position based (PBVS)? [1 mark]

- ☐ **A** IBVS defines the error in pixels and never converts to metres; PBVS estimates the target's pose in the world and controls to it
- ☐ **B** IBVS needs an accurate calibration; PBVS does not
- ☐ **C** IBVS uses a depth sensor; PBVS uses a camera
- ☐ **D** IBVS controls rotation only; PBVS controls translation only

2. The forward loop should stop when a 6 cm ball is 25 cm away. What apparent width in pixels is the setpoint, to 1 decimal place, with f = 92.4? [1 mark]

3. The loop from the last question, with its 22.2 pixel setpoint, is pointed at a ball that is really 8 cm across. At what range does the robot stop, in cm to 1 decimal place? [1 mark]

4. This is the lesson's rotation law for several column errors. What does it print? [1 mark]

```
for err in (40, 10, 4, -30, 200):
    rot = max(-55, min(55, 0.35 * err))
    if abs(err) < 6:
        rot = 0
    elif abs(rot) < 17:
        rot = 17 * (1 if rot > 0 else -1)
    print(err, round(rot))
```

5. Why is rotating to centre a target the most reliable visual control there is? [1 mark]

- ☐ **A** The rotational terms of the interaction matrix have no depth in them, so no range estimate is needed
- ☐ **B** Rotation is faster than translation on the BugBot
- ☐ **C** Turning has no dead band
- ☐ **D** The principal point is always exactly at column 160

6. A real camera pipeline has 100 ms of latency and the robot drives at 20 cm/s. How far does it move before it acts on a frame, in cm? [1 mark]

---

7. The blob list comes back empty in the middle of an approach. What is the best behaviour? [1 mark]

- ☐ **A** Keep turning the way the target was last seen to go
- ☐ **B** Keep driving with the last command
- ☐ **C** Stop and wait for the ball to reappear
- ☐ **D** Spin in a fixed direction until something is seen

The task: servo on the pixels

A red ball, 6 cm across, is off to the robot's right and the robot is not facing it. Drive up and stop about 25 cm away, controlling rotation from the blob's column and forward speed from its apparent width. Plot `cx` and `width`, and print `width:`, the apparent width you stopped at. Neither `distance()` nor `position()` is allowed.

```
from bugbot import *
connect()

F, R = 92.4, 3.0
STANDOFF = 25.0
set_cv("blob", "red")
wait(0.3)
```

Plan your program here, then type it in and press Run.

Do it on the robot
www.bugbotlab.com/learn/u11-6-visual-servoing/
The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Double the rotation gain and look at the `cx` plot. Where does it start to ring?
2. Delay your measurement by one tick on purpose, using the previous frame's `cx`. How much gain can the loop take now?
3. Change the standoff to 40 cm without touching the control law. Which line did you have to edit, and what does that say about where the units live?