Part IV. When the world isn't labelled...

Background

In Minecraft (unlike the real world), there is a finite number of object types (various rocks, trees, horses, water, lava, etc.) and every object is already labelled. These labels can be very useful when creating AI agents. We can write conditional behaviours along the lines of "if a nasty creature is north of us, run south!" "If there is gold nearby, pick it up!" etc.

But what if we don't have access to these labels? This is the challenged faced by natural agents. In the real world, sensory input does not arrive with clear identifiers like "this is a predator" or "this is food". How then do we manage to run away from predators and search successfully for food? In the early days of AI, the problem of translating sensory data into labelled input was often thought to be just around the corner. Most researchers thus focused on what they saw to be the more interesting (and more accessible[1]) parts of AI: how to manipulate labelled objects to solve problems. But it has since become quite clear that transforming basic sensory information into labelled entities is much more challenging than it first appeared.

Some computer vision research attempts to addresses this challenge. Primary methods involve identifying features of the environment (e.g. the corners of an object) and using these features to inform the creation of an internal model of the world. To anthropomorphize:

> *"There are six corners in my vision arranged in a particular way that suggests the presence of a box located two meters in front of me and a bit to the left. I will store that inference in my model of the world, and update it if it proves to be wrong later, or if things change."*

For some, the answer to this challenge is to recognise that it in many cases it is not necessary to build those internal representations.

Overview

The purpose of this exercise is for you to explore how artificial agents can accomplish intelligent behaviours without explicitly labelling its sensory data, and without using it to build an internal model of the world.

For this challenge you must use only "raw" sensory information such as the red, blue, green, and/or depth of the agent's vision, and the angle of the agent's head (yaw & pitch) to solve the challenge.

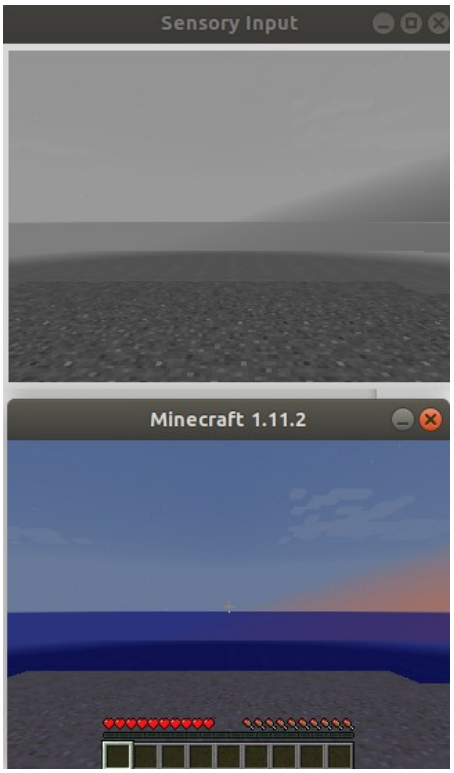
Stranded on a desert island

Backstory: your robot is stranded on a desert island on a distant planet. To send a message requesting assistance it must approximate how much time has passed since it was stranded. Its internal clock is broken, so it must use the angle of the sun (or the moon, whichever is currently in the sky) to determine

1 What do I mean by this?

the time of day. Your challenge is to write the AI that takes raw sensory data and uses that data to estimate what time it is.

Download and unzip the files for worksheet 4. After launching Malmo as in previous worksheets with `launchClient.sh`, run `python3 stranded.py` – this should bring up two windows.



One window is the usual Malmo/minecraft window (bottom) which shows what your AI can see. The other is a debugging tool that I made to help you develop this AI. It shows, in real-time, the value of a 2D array. In the initial code you receive, it shows the blueness of the agent's visual field.

Around line 128 is where this 'visualizer' is updated.

```
  pixels = world_state.video_frames[-1].pixels

  red,green,blue,depth = self.pixels_as_arrays(pixels)

  sv.display_data(blue)
```

Here, the first line gets the data from malmo. The second line sends it to a helper function I made that parses that data into red, green, blue and depth channels...and the third line updates the visualizer to show the blue channel.

You might try to change this visualization to the depth channel to confirm that that channel is what you think it is.

You may need to install extra packages to get this visualization tool to work. In particular, it relies upon PIL (the Python Image Library). Feel encouraged to share your experiences getting the basic code to run and ask for help on piazza. I'll do my best to assist you with this.

Browse through the rest of stranded.py. I have tried to design this exercise so that the only code you need to look at or modify is in this file. Of course you are welcome to use or modify other files, but hopefully you shouldn't need to.

A couple of things to point out:

- I have annotated two places in the file where I expect most (all?) of your solution to go. To find these, you can search for the text `NOTE`.
- Note that the `StartTime` (you can search for this text in the file) is set to be random at the start of the script. You can change this if you like while developing, but at the end, your script should be able to adapt appropriately to find the sun/moon at all times of day.

Start editing the code to cause the agent to look for the sun (or moon) in the sky. I imagine your solution will be inspired by Braitenberg's vehicles.

Once you have an agent that is consistently finding the sun (or moon) in the sky, you can pass the pitch of the robot's head (`self.pitch`) to the method `angle_to_time`, which will make a guess about what the time of day is.

Additional comments

- In this exercise, we assume that the moon is always at the top of the sky at midnight.
- You can type /time set NUMBER_BTWN_0_AND_24000 to instantaneously set the time. This is a useful when developing & debugging.
- You should plot how your estimator functions over the course of a day. Where does it perform best? How does this compare to your own abilities to estimate the time of day by looking at the sun?

Marks breakdown

1. **2 marks** – The robot can track the sun or moon and pass a good estimate of its angle in the sky to the `angle_to_time` method
2. **1 mark** – The robot can distinguish between rising and setting celestial objects. If your solution uses `self.yaw`, your maximum score here is limited to a ½ mark.
3. **½ mark** – The robot can distinguish between sun and moon.
4. **2 ½ marks** – A one-page report (this is a strict maximum, but does not include any graphs). The report should include
    A. A simple human-readable overview of how your solution works (perhaps 1 paragraph). Relate your description to key concepts from the SED portion of the course.
    B. A well made (labelled etc.) plot showing how well your solution estimates the time over the course of a day.
    C. Observations about the conditions in which your solution performs better or worse, and some analysis or explanation for why that might be the case. Relate these comments to the way you might similarly accomplish this task if you were on a desert island.

Please make your submission straight-forward to evaluate.

- You can include a video (recorded with your phone of the screen if need be) showing the behaviour of your AI. Make sure this video is reasonably sized or hosted on youtube.
- Marks 1-3 will be evaluated by looking at the output of your program, which should call `angle_to_time` which prints out AI's estimate. It will also be evaluated by looking the plot that is part of the report (4B).
- Your submission must include your code (and supporting files) so that I can unzip it and run `python3 stranded.py` to see your system working.