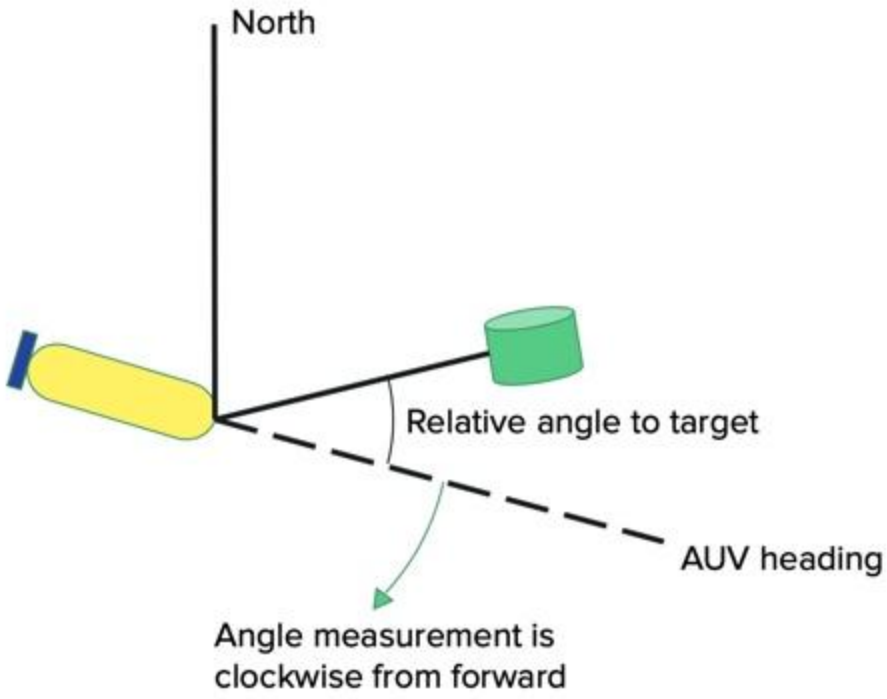
BWSI AUV Challenge 2021

Lab 10
Navigating with a Camera Sensor



Introduction

In the first simulation challenge your AUV had perfect knowledge of the positions of the next red and green buoys. In the final AUV challenge this will not be the case! You will be relying on detections from a camera that at best will provide relative angles to the buoys., which are measured clockwise from the heading of the AUV. For now we will assume that when you read the camera it will report correct relative angles to any green and red buoys within its field of view. Even this “perfect” camera will present several challenges to your developing controller code, which we will address in this lab. In later labs, and in the final challenge, you will have to compute those angles from the imagery that the camera provides.

Python Code

In the Lab 10 directory you should have all of the script files you need to run the simulation examples in this lab. The AUV_Controller.py is not fully developed so it makes some less-than-robust angle calculations and crude decisions, but it is instructive for code organization --- and it *almost* works on the first course. Please start from this version of the code -- you can pull over the decision logic that you developed in Lab 8 and the Simulation Challenge 1 as you progress through the lab - but please try to fit it within the structure of the provided AUV_Controller.py.

The main test function, lab10_navigating_with_a_camera.py, has two global variables at the top to control the different test configurations. These are COURSE_NUMBER, which is a selector for different buoy fields. The valid options for COURSE_NUMBER are 0,1,2,3, or 4. The other global variable is VISIBILITY, which is the distance the camera can see buoys (in m). The camera also has a fixed view angle of +/-60 degrees (relative to forward on the AUV), so it will only see buoys within those angles. In this simulation, if a buoy is inside the fixed view angle range and is less than VISIBILITY meters away, it will always be detected and reported to the controller.

You will be changing the values of those two global variables throughout this lab to build up the complexity of your controller logic.

The First Test Case

At the outset it is set up as COURSE_NUMBER=0, which means the AUV will follow the same course as in the Sim Challenge 1, and VISIBILITY=99, which means that the camera will see any buoys within 99 m of the vehicle. This visibility range is convenient for the first course because it means the camera will *almost* always see one red and one green buoy (assuming your AUV is heading the correct general direction), so the simulation will run the same as the Sim Challenge 1, with some minor adjustments for the rare case that a buoy pair is not visible, as long as you calculate the desired heading correctly.

Update your decide() function so it calculates the desired heading correctly when the sensor_type is ‘ANGLE’. Note that this calculation has been delegated to a private member function *__heading_to_angle* so the calculation goes there. The provided code will correctly calculate the desired heading most of the time - but there are two considerations you will need to deal with:

1. The function has assumed that there is one red buoy and one green buoy. What happens if that is not the case? In this test case, the number of green and red buoys visible will either be 0 or 1. If it is zero, the best course of action is generally to maintain the current heading until you do.
2. Is the target heading actually correct? Assuming you maintain a steady track, it will always return the correct heading in this case. Consider the case where the AUV heading is at 359 degrees and the buoys are off to its right about 10 degrees. What would the returned tgt_heading be?

Trying the Other Courses

Once your controller is working well on the first test case, change the value of COURSE_NUMBER at the top of lab10_navigating_with_a_camera.py to 1, and then 2, 3, and 4 as you progress through the courses. Once your controller is navigating each of these courses well, you can move on to the next step. Note that by “navigating the courses well” we mean that it is making its way through the gates: if you make it through most of the course but just run out of battery that is good enough to move on. We have given you an extra large battery for these missions so that you will be less likely to run out of battery.

On a Nice Clear Day You Can See Forever

After a few calm days with minimal wind and shipping traffic, the ocean clears up significantly, and your camera’s detection range improves a bit. Set the VISIBILITY to 150 at the top of lab10_navigating_with_a_camera.py, and start over from COURSE_NUMBER=0. Now you will need to contend with the possibility that two red and/or green buoys are visible to the camera at a given time.

When Your Ship Comes In

After showing off your AUV navigating with a camera, a large corporation gives you big $$$ to build a system to autonomously navigate their underwater cargo ships. You decide to upgrade your AUV to the new ultra high power Digitronix Lightwave camera so you can navigate even better! Now set VISIBILITY to 500 m and repeat the COURSE_NUMBERs from 0 to 4, this time contending with the potential that multiple red and green buoys are visible to the camera at any time.

Summary

Which situations were more challenging? Is having a long-range camera helpful or is it better to have a relatively short range camera? What if your camera range was 5000 m?

Extra Section (*optional*)

You might have noticed that, in addition to the BWSI_Camera, your simulated AUV has a BWSI_Laser. The Laser sensor is a 'RANGE_ANGLE' type sensor, and it returns a (range, angle) tuple to each buoy. Its detection range is also limited to the value of the VISIBILITY variable, and it has a default scan angle of +/-60 degrees. We've provided a copy of lab10_navigating_with_a_camera.py that instead uses the laser, and called it lab10_navigating_with_a_laser.py.

As a first cut, you could just modify the decide() function to handle the 'RANGE_ANGLE' type, processing the angle the same as if it came from a camera. Once you've done this, verify that the controller works the same as before by running some of the same courses with the same visibility values as before.

This approach works! However, notice that you've thrown away a piece of information - the distance to the buoys! How might you make use of that information? Consider updating your command selection function to make use of the range to target. Would you make a different decision on a heading command if you are 3m from the gate than you would if you were 10000m from the gate?