**LAB 3: Write-up**

(Team 30: Manan Patel & Sukriti Singh)

1. **What is the sampling time of your system (hint: how fast can your get sensor data)? How does this matter with a subscriber based controller?**

Answer:

Technically speaking, the sampling time of our system is limited by the framerate of the image capture routine from the raspberry pi camera module which is 10 Hz. However, the sampling time for find\_object node is around 6 Hz. Hence, truly, the sampling time of our robot is limited by this routine.

This means that for a subscriber based controller, it will be acting on past value for around 0.15 sec (1/6 Hz), that is, there is a time delay in the system. And when modelling a system, even if the gains are perfect, the system can become unstable if there is a large enough delay in the system. So, it is important to incorporate this time delay when working with subscriber based controllers.

1. **What variant of PID control did you use? Why?**

Answer:

We used PID control in both cases, i.e. for controlling angular as well as linear velocity.

To begin with, we had only PI controllers in both cases. The robot was definitely reacting faster than that in Lab 2, which was obviously expected due to the feedback loop. There were only very slight oscillations when the robot rotated to face the object (in our case, the diamond shape). This was also at a moderately low angular and linear speed, and when roscore was being run on our system (and not the Turtlebot).

However, as we started running the code on the robot itself and tried to increase the speeds to reduce response time, we saw slight increase in oscillations and jerky movements. Hence, we then introduced a Kd term to stabilize these aspects. This way, we could use higher proportional and integral gains too.

1. **If you use an integral term, how do you deal with windup? If you use a derivative term how do you deal with noise/fast changes in the object's location? If you just used a purely proportional control, how do you deal with disturbances?**

Answer:

We have added the following conditions in our logic to deal with integral windup:

1. The sum of errors, i.e. the past error values that the integration is done over, is reset to zero once the robot has reached the set position and location. This is done separately for each controller.
2. We have also set a maximum value of velocity (a threshold) as a safety feature, so that even if the integral builds up over time, the controller does not blow up.

We did use a derivative term, however we have used a small Kd in comparison to the effects of Ki and Kp. Hence, we did not actually face a scenario of undesired robot behavior during fast changes of the object’s location, the values of proportional and integral terms make up for it.

1. **What does it mean for this system to be unstable? A helicopter/plane will fall out of the sky if it uses an unstable controller, what does your robot do when your controller is unstable?**

Answer:

We noticed the following at different instances that can be classified as unstable controller behaviour:  
(i) Oscillating (rotating left and right) continuously instead of facing the object as desired

(ii) Having too high velocity when it begins to move (high overshoot) such that it moves unexpected distance and diverges from set position/ location

1. The speed only increases over time, causing the control logic to fail.
2. **Describe your algorithm to determine where the object is relative to the robot. Specifically, how do you use the camera and LIDAR data to produce a desired velocity vector? Include mathematical expressions used and supportive figures where appropriate.**

Angular distance: We converted pixels to degrees taking into consideration, the field of view of the Turtlebot camera (which is mentioned as 62.2 degrees in the specs. sheet). Our code to find the object (diamond\_finder) returns the x-coordinate of the centre of the diamond. Since there are 154 pixels in total, the centre of an image captured would be 77. Hence, we take this difference between the centre of the image and the coordinate returned by the diamond\_finder, convert it to degrees and use it to set the desired angular position of the robot. The angular velocity vector is set based on the difference in the desired angular position and current angular position of the robot.

Chart, line chart

Description automatically generated

Equation used: y=mx+c,

where

m = (62.2/77)=0.8078 ,

c= -62.2

x is the x-coordinate returned by the diamond finder

(Note: When the robot is facing the diamond exactly, the angular position is zero. When the diamond is towards the right or left sides, the difference in desired and current angular position is positive or negative accordingly)

Linear distance: We use the LIDAR data to measure the distance of an object at 0 degrees (which is the same as 360 degrees) of its orientation. The code is such that a distance of 0.3m is expected to be maintained between the robot and the object. When the flag for the object being found is True and the robot’s angular position is such that it is facing the object, then if the distance between the object and robot is greater than or lesser than ~0.3m, a linear velocity is set to bring this difference in desired and current distance from object to zero (The code obviously has has a limit, and not accurately zero). Summarizing the above logic in a simplified flowchart:

Diagram

Description automatically generated