Comp2403 Dead Reckoning

C.B.Price September 2021

Purpose	To learn how to drive the robot forward for a given distance in millimetres, and to realize the serious limitations of this approach.
Files Required	Arduino Sketchbook and Octave scripts on the web-pages.
ILO Contribution	LOs 1 & 3
Send to Me	nix
Homework	Read chapters 1 & 2

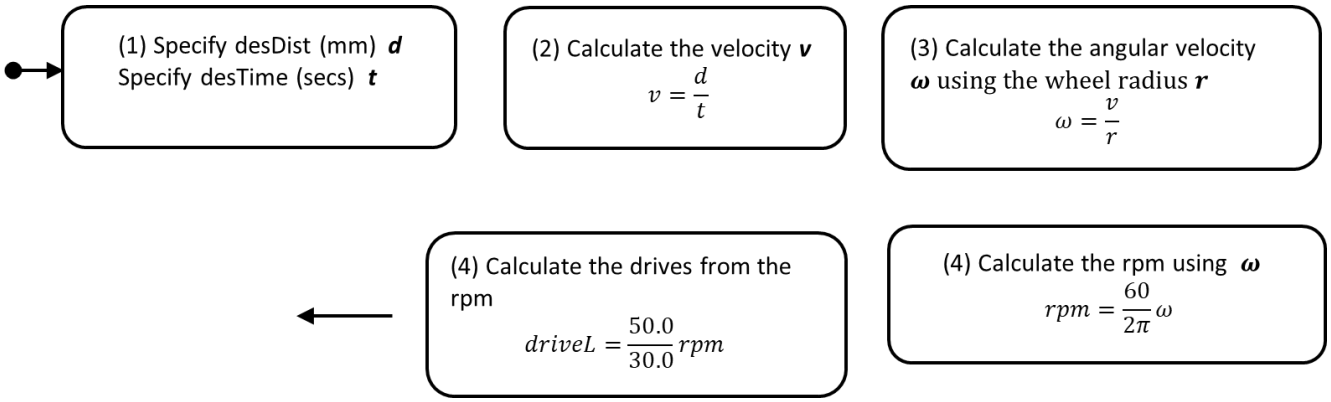
Activities

---

1 The problem and coding the solution

We want to get our robot moving forward for a distance (in mm) we specify. We also need to specify the time (in secs) we want our robot to cover this distance. That's the input to the problem. The output from the problem (the solution) is to get values for **driveL** and **driveR** which we send to the servos.

So, we need to do some computations. The diagram below shows these, in order, a sort of flow-diagram.



The 50.0/30.0 comes from the servo graph which you saw in the first worksheet. Here's a table that maps some maths symbols to variable names, used in the supplied code template

d	**desDist**
t	**desTime**
v	**desSpeed**
r	**wheelRad**
ω	**omega**
rpm	**rpm**
driveL	**driveL**

(a) Open up the sketch **SBP_2403_R_Forward_Calibrate.ino** and build up your code using the above information. The place where you should start is clearly indicated. Don't forget to drive both wheels. Choose a sensible desired distance, say 80 – 100 mm, and a reasonable time say not less than 1 second.

(b) Upload and run your sketch. Keep the robot attached to the Serial monitor using its USB umbilical so you can see the messages. ***Hold the robot above the bench top, and observe the wheels turning.*** Look at the Serial monitor printout. If you want to check your calculations are correct, an Octave script is provided. Grab your tutor.

(c) Now disconnect the umbilical and take the robot to the arena. Grab a piece of paper, pencil and rules. Draw a 'start line'. Switch the robot on, press reset, wait for the wheels to glitch then put the robot down. Measure how far it goes. *You will not be impressed.*

(d) You may like to increase the drive time and see if things improve. I haven't tried this, so I will be interested in your findings.

2 Calibrating the system

So, we saw the calculation almost certainly gave you the wrong result. Correcting this is straightforward, we use the desired distance and the observed distance.

(a) Add the following line of code to correct the angular velocity **omega**

omega = omega*desDist/actDist**;**

where actDist is the distance you measured and **desDist** is what it is. See if there is any improvement.

(c) Rerun and make some measurements. A real engineer would make several runs, note down the distances covered and make a scatter-plot of these and calculate the mean, and possibly standard deviation.