ALBiR unit 4: motor control and navigation

Learning objectives:

Motor control is key to successful locomotion. Providing appropriate motor drive relies heavily on prior knowledge of the actuators and plant dynamics. Even without explicit sensory feedback, animals and robots achieve a variety of motor tasks with adequate accuracy. A great example is dead reckoning navigation which employs odometric information. In particular, we would like to explore the internal estimation of motor output by integrating the motor commands.

To model a mobile robot, one must characterize the motor system well at different speeds, understand the effect of starting and stopping, and tune the drive function appropriately. This will lay down the basis for sensorimotor integration and eventually autonomous racing in later units.

In-class activities:

1. Lecture in motor control and navigation

This is the first session for section 2 on sensorimotor control. Apart from object manipulation, navigation is perhaps the most important topic for sensorimotor control in animals and robots. This lecture will introduce the challenges of navigation and motor control.

2. Assembling the differential drive robot kit [study-group]

Check your own assembly against the following criteria:

- Are both motors properly aligned and secured on the body plate?
- Are both wheels and front skid installed properly?
- Are electronic components properly mounted and secured?
- Are the signal cables neatly folded and secured down?
- Is the camera module securely installed and the USB cable does not hinder its pan action?

The software setup instructions for the Raspberry Pi can be found in a separate document on BlackBoard. You will follow the instructions to set up your system. Ping any questions on Team study group channel. For code to run the robot refer to the github link.

https://github.com/dragonflyneuro/ICL-BioEng-ALBiR-PixyBot/tree/main/Setting%20up

Homework 1: robot motor tuning (individual assignment)

In order to understand the behaviours of the motors, we must examine the input-output of the system. Here, the input will be our motor command signals and the output will be measured from the vehicle kinematics. There are a variety of black-box approaches to modelling motor input-output relationship. Here we introduce the most basic approach to allow transparency of what's happening. Please use the guidelines below to complete the system motor calibration:

- Plot the left and right motor velocities as a function of drive levels. Please mark the deadband and include standard deviation of the measurements. Justify your model fit.
- From single wheel turns, plot adjusted drive values as a function of desired wheel velocities.
- From straight line test, plot the percentage correction as a function of drive levels.
- From symmetric turn, plot the robot mean turn rate as a function of expected turn rate.
- Discuss your results from these key plots and answers the questions below.

Single wheel turns

Place the robot on a flat surface on which the wheel slippage is minimal and record its heading. To initiate a single wheel turn, download open the exercise 1.py script on your pixyBot. In the main function, uncomment the function call to tp.singleWheelTurn in the main function. The first argument is the motor drive (-1 ~ 1), the second argument is the side on which you want to turn the wheel ('left' or 'right'), and the third argument is the amount of time for which you want to run the motor in seconds. Call exercise 1.py from the terminal to start the robot test. The general idea is to spin the robot with one wheel for a fixed time (e.g. enough time to spin 5~10 rotations). This allows the estimation of the average turn rate as a function of the drive level. Following is an example table to capture data (note: each wheel might need some time to warm up to consistent speed):

Drive level	Left wheel driven (average rate °/s)	Right wheel driven (average rate °/s)	Notes
+1			
+0.8			
+0.6			
+0.5			
+0.4			
+0.3			
+0.2			
0			
-0.2			
-0.3			
-0.4			
-0.5			
-0.6			
-0.8			
-1			

If the pixyBot seems to move when the turn rate is set to 0, you can re-set the zero point of the continuous servos by editing zeroPosL and zeroPosR in lines 220-221 in pixyBot.py. The servos use PWM as control signal and these variables control the ON-period of the PWM control signal for when the servos should not be moving. Try adjusting this by in +-50 inc/decrements

Enter your data into the exercise 1_modelFitter.m MATLAB script to generate a model fit of the motor behaviour. We chose to fit a sine function to capture the motor saturation effects. The code will

normalise out the real unit (°/s) so you will end up with a unitless motor model. Open the pixyBot.py and search for where lCoeff and rCoeff are defined to insert your results from the MATLAB output.

Find the drive levels for both motors that produce the same turn rate. Find the dead band. What's the variations? What does the variation depend on in your opinion? What's theoretical maximum speed for your robot considering both motors?

Straight line

Based on the single wheel turning test data, we can now attempt to drive the robot straight to fine-tune the drive levels. Set the robot to drive in a straight line for a fixed time at different speed levels.

To make the robot drive straight, go back to the exercise 1.py script, comment out tp.singleWheelTurn and uncomment tp.straightLine in the main function. The first argument is the speed as a fraction of the maximum (-1 ~ 1). The second argument is the corrections to apply to the left and right motors respectively in ration (e.g. 0.1 means adding 10%). The third argument is amount of time in seconds for which the motors will turn. The 2nd and 3rd arguments can be left blank for the default values. If you have edited the lCoeff and rCoeff already, the calibration would have been applied here. You will be testing if your calibration from the last section requires any further correction.

Speed	**Actual speed (cm/s)**	**Drifting? Which way?**	**Notes**
+1			
+0.8			
+0.6			
+0.4			
+0.2			
0			
-0.2			

Discuss your straight line runs, and any further corrections. What's the impact of the transient effects for stop and go? How would you account for them?

Symmetric wheel turns (optional)

Based on the above two tests, we should be able to control spot turning with good precision. Perform similar tests as single wheel turns by attempting to drive the two wheels in the opposite directions with the same velocity. Measure the actual turn rate under such condition. It should be higher than the single wheel turning. To initiate a symmetric wheel turn, go back to the exercise 1.py script, comment out tp.straightLine and uncomment tp.symmetricTurn in the main function. The first argument is the turning speed as a fraction of the maximum (-1 ~ 1), and the second argument is the amount of time to run in seconds. Create a table like in single wheel turns exercise to help with data collection.

To perform open-loop control well, one can improve the motor model using the extra data collected from the straight line and symmetric turn exercises and/or transient effects. What are some ways to improve the simple motor model fit we generated?

Dead reckoning game (optional)

Using the information you have collected through the previous exercises, hard code a sequence of commands that makes the PixyRacer complete a simple dead reckoning course you design. Check how accurate is your open-loop response given your motor tuning.

To help you get started, we provide the tp.deadReckoning function in the exercise 1.py script. Just add a list of time durations and motor commands to the respective variables, and the robot will execute them in sequence when you call the function (comment out all other function calls and uncomment the call to tp.deadReckoning). For example, setting

deadReckoningRoute = [[1,1,5],[1,-1,0.5],[-1,-1,2],[-1,1,0.8],[1,1,4]]

will make the robot move forward for 5 seconds, spot turn right for 0.5 seconds, go backwards for 2 seconds, spot turn left for 0.8 seconds, and finally move straight for 4 seconds.