catkin build
roslaunch . . box.launch
rosrun 4_3 ex_4_3_1
rosrun 4_3 ex_4_3_2

robot_move.md

Exploring the Robot's Motion

In this tutorial, you will create a program that uses the buttons to control the speed of the left and right motors separately. You will then use this program to learn how the robot can perform simple turns, speed up, and apply its brakes.

For these exercises, you'll be using the `box.launch` launch file.

The Move Function

bot.move(50, 50);

[-100, 100]

```
void move(int speedL,int speedR);
```

The `move` function for our robot takes in two parameters, `speedL` and `speedR`, and moves the left and right motors at the speeds specified.

The values of `speedL` and `speedR` range from -100 to 100, where -100 is the maximum speed in reverse, and 100 corresponds to the maximum speed in the forward direction.

Passing in `bot.move(0,0)` will stop the robot.

If you ever lose the robot or want to just move it back to the beginning, push the reset button!

You can turn the robot by moving the motors at different speeds.

- -100, 100 would turn the robot as hard to the left as possible
- 100, 50, would turn the robot less sharply to the right, navigating the robot in a circle..

Exercise 4.3.1

- Fill in the "empty" program in ex_4_3_1.cpp so the robot moves when it's started, using `move`, and stops when it sees an obstacle.

We recommend setting both speeds to 25 for this exercise so you can easily observe the stopping behavior as well as master its usage before crashing your robot into the wall at top speed.

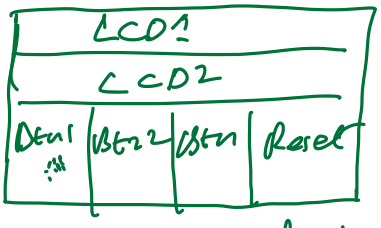
Building a Simple Interface Using the LCD and Buttons

```
int readButton();
```

bot.readButton() == 0, 1, 2

The `readButton` function indicates which of the buttons Button 1, Button 2, or Button 3 is being pressed. The function returns an int, with the possible values:

Value	Meaning
0	no button is being pressed
1	Button 1 pressed
2	Button 2 pressed



switch (bot.readButton())

Value	Meaning
3	Button 3 pressed

This function, just like the lcd, led, and move functions, must also be called on the TexBot object.

The following exercises will build off of each other. Please keep your code for each exercise in their separate cpp files, but as you work through these exercises feel free to copy your code from the previous exercise into the file for the next exercise and make your modifications in the new program.

Exercise 4.3.2

- In this program, print which button is being pressed on the top line of the LCD, "No button," "Button 1," "Button 2," "Button 3."

Using a switch-statement to identify which button is being pressed will simplify your code.

Exercise 4.3.3

We're going to begin to create a simple interface that will allow us to control the speeds of the motors.

- Add a boolean variable to the top of your program (before the `ros::ok` loop). Call it `leftRight`. Set it to `false`.
- In the loop block, make it so that if `leftRight` is `false`, "Left" prints on line 2 of the LCD. If it is `true`, print "Right" on line 2 of the LCD, instead.
- Make it so that pressing Button 3 toggles `leftRight` so that if it is `true` it becomes `false` and vice-versa.

At this point, your program should now say which button is being pressed on the top line if a button is currently pressed, and should switch between "Left" and "Right" on the second line whenever you press Button 3.

Button Toggles

Right now, when you press Button 3, you've probably noticed that the value for `leftRight` changes rapidly. This is due to the fact that pressing and holding the button causes the button to be pressed multiple times.

To make the button toggle, we want to change the value of `leftRight` only when we have pressed Button 3 for the first time.

Exercise 4.3.4

- Add another boolean variable to the top of your program. Call it `threePressed` and set it to false.
- Instead of only checking to see if Button 3 has been pressed when changing the value of `leftRight`, make sure that Button 3 has not already been pressed.
    - We will now also set `threePressed` to `true` when we change the value of `leftRight` to show that the button has been pressed.
- If Button 3 was not the button that got pressed, set `threePressed` back to `false` so it can be pressed again later.

Exercise 4.3.5

- Add two more variables before the `ros::ok()` loop, `int leftVal` and `int rightVal`.
- Set both to zero at the start of the program.

- Make it so that pressing Button 1 increments `leftVal` by 1 if `leftRight` is `false`, and `rightVal` by 1 if `leftRight` is `true`.
- Make it so that pressing Button 2 decrements `leftVal` by 1 if `leftRight` is `false`, and `rightVal` by 1 if `leftRight` is `true`.
- Display the values of `leftVal` and `rightVal` on LCD line 1. Remove the code that displays which button is being pressed.

Now your program should allow you to increase and decrease `leftVal` and `rightVal` and select which is changed using the pushbuttons on the robot. You may notice that your values change in large increments. This will be fixed in the next exercise!

Exercise 4.3.6

We'll want to fix Button 1 and 2 so that they don't increment at too high of a rate.

- Add one more variable before the `ros::ok()` loop: `int count`. Set it to zero.
- Instead of always checking to see if Button 1 or 2 have been pressed, only check at a large interval.
  - We can do this by seeing if `count` is a multiple of a large number.
  - For example, only check the buttons if `count` is a multiple of 7000. You can do this with the `%` operator!
- Make sure that you still set the value for `threePressed` properly, even if you don't check to see if Button 1 or Button 2 are pressed.
- Don't forget to increment `count` once every time the `ros::ok()` loop executes!

Exercise 4.3.7

- Use `if` statements, the modulo (`%`) operator, or other logic to limit the range of `leftVal` and `rightVal` to be between -100 and 100.

Controlling the Robot's Motors

What we're going to have you do now is combine the user interface that you just wrote with the motor code from Exercise 4.3.1.

This will let you try different things with the robot's motors to see how the robot can move and turn, as well as stop before it hits an obstacle.

Exercise 4.3.8

- Create a new program combining your UI from Exercise 4.3.7 with the motor code from 4.3.1.
- loop block should take the user input from the buttons at the top of the function.
- At the bottom of the block, you should detect whether or not an obstacle is in front of the robot.
  - If there is an obstacle, the robot should stop, using the `move` function.
  - If there is not an obstacle, it should put leftVal and rightVal into the left and right motors, respectively, using `move` as well.

Exercise 4.3.9

- Go back to the `box.launch` launch file, and try a few different things with the robot's motors.
  - Can you make the robot make a big circle to the right?
  - Can you make it make a big circle to the left?
  - Do you understand why this happens?

    - Can you make it spin in place in both directions?
    - Can you make it go backwards?

- You can use the UI you've made to adjust the speeds of the wheels to see these behaviors!

Reference:

- https://github.com/Texas-Robotics-Academy/markdown.git