

Part 1. Moving Forward

Requirements - Write a program that makes the robot move forward 1 meter and then stops.

In this lab, our attention will be focused on two components of the robot used in lab: the iRobot Create and the serial interface (UART) on the microcontroller. We will issue commands over the TM4C123G's UART4 connection to the iRobot and use it to control movement. The iRobot Create has an "Open Interface", or a standard way of communicating, that allows us to send a range of commands. The commands we are primarily interested in are those related to movement and sensor values. A simple API has been written for you that handles setting the wheel speed and retrieving sensor data.

You will need to use the **oi_setWheels(...)** function to tell the iRobot Create to move. The **oi_setWheels(...)** function calls the "Direct Drive" command (opcode:145) as given in the Open Interface specification. As you can see in the OI specification, the wheel motors can be set to run independently from each other. The velocity ranges from -500 mm/s (backward) to 500 mm/s (forward).

Beyond making the Create move, you will need to know how far it has moved. The same Open Interface API supports fetching sensor data from the iRobot Create. The **oi_update(oi_t *self)** function should be passed a pointer to a **oi_t** structure. The function will then update every member of the structure with the current sensor value. You should attempt to gain an understanding of the **oi_t** structure described in **open_interface.h** and its relation with the sensor packets listed in the Open Interface specification document.

Notes

- Ask your TA as needed about setting up your project (Lab 1) and workspace directory.
- Ensure the iRobot Create is turned on by checking that the Power Button is lit.
- Make sure you initialize the serial connection with iRobot Create before sending it commands. This is done by calling:

```
oi_t *sensor_data = oi_alloc();  
oi_init(sensor_data); // should turn the iRobot Create's "Dirt  
Detect" LED to blue
```
- Be careful that code doesn't run the iRobot Create off the table... wheels up!
- Turn the iRobot Create off **BEFORE** loading your program so the robot does not run off the table. After programming, disconnect the USB cable, turn the iRobot Create on and reset your program with the reset button.



In addition, you will want the iRobot to tell you how far it has travelled. Every time you call `oi_update(...)`, two members of the struct will be updated: angle and distance. These variables are explained in detail on page 27 of the Open Interface document. Read the following example code to gain further understanding.

```
#include "open_interface.h"

void main() {
    oi_t *sensor_data = oi_alloc();
    oi_init(sensor_data);

    double sum = 0;
    oi_setWheels(500, 500); // move forward; full speed
    while (sum < 1000) {
        oi_update(sensor_data);
        sum += sensor_data->distance;
    }
    oi_setWheels(0, 0); // stop

    oi_free(sensor_data);
}
```

Checkpoint: Verify the Cybot drives forward 1 meter then stops, and demonstrate to your mentor (i.e., TA).

Part 2. Moving in a Square

Requirements - Write an API (create two new files: movement.c, movement.h) that contain new functions that help you (1) move the robot forward a specific distance and (2) allow you to turn the robot left or right by a certain number of degrees. Use these new files and functionality to write a program that makes the robot move in a 50-centimeter square, then stop the robot (i.e., do the following four times: move forward 50 cm, turn 90 degrees).

Reminder: Put comments in your code!

Notes

- Ensure the iRobot Create is turned on by making sure the Power Button is lit.
- Use good method names, such as **void move_forward(oi_t *sensor, int centimeters);** and **void turn_clockwise(oi_t *sensor, int degrees);**
- Follow good coding practices; i.e., put your function signatures in the header file (.h), and put the implementation of the functions in the .c file.
- You'll need to use the angle member of the oi_t struct. Remember, positive angles are counter-clockwise; negative angles are clockwise.
- To save memory, call oi_alloc() and oi_init(...) only once at the beginning of the program. You will need to pass the pointer (oi_t *) to your functions.
- Turn the iRobot Create off **BEFORE** loading your program so the robot does not run off the table. After programming, disconnect the USB cable, turn the iRobot Create on and reset your program with the reset button.

Checkpoint: Verify that each corner of the square is close to 90 degrees, and demonstrate to your mentor (i.e., TA).

Part 3. Collision Detection

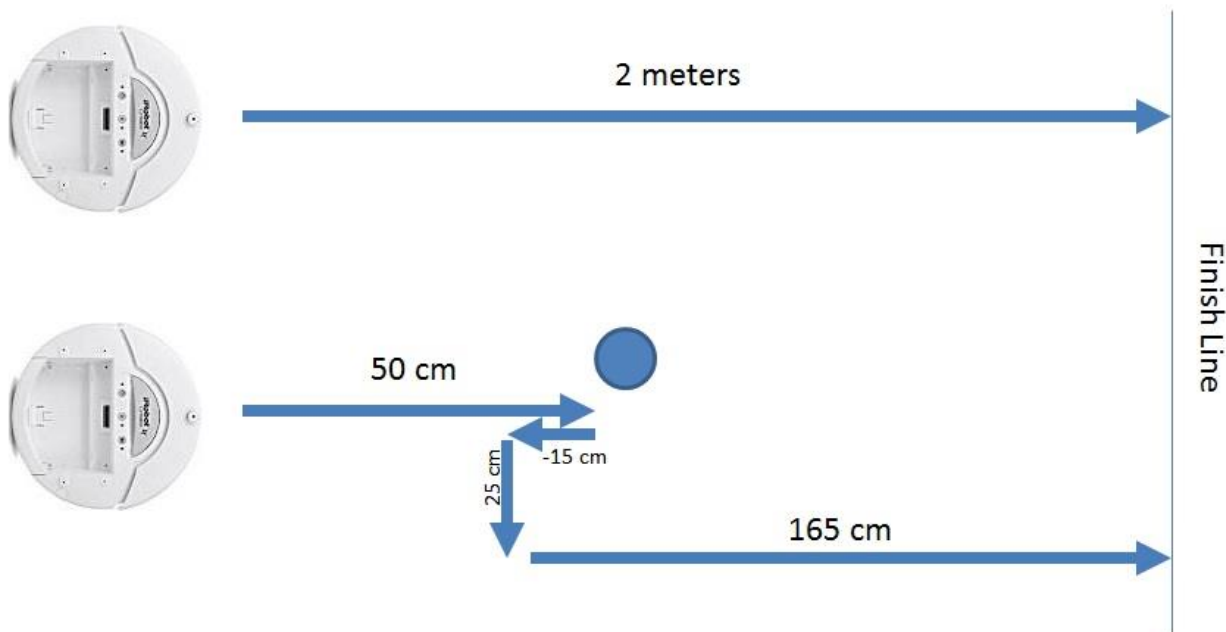
The Create has many sensors. There are two bump sensors in the front of the device. In Part 3, you are to integrate the bump sensors with forward movement. The robot should be able to bump multiple times in its 2 meter run.

Requirements - Write a program that satisfies the following requirements; you should reuse, modify, and expand the API (movement.c, movement.h) from part 2. The robot shall move forward a total of 2 meters (imagine a finish line 2 meters away). If no obstacle is present in the path of travel, the robot should simply travel 2 meters forward. However if the robot comes in contact with an object, the robot should attempt to go around the object by issuing the following commands: back up 15 cm, turn 90 degrees, move laterally 25cm, then turn 90 degrees forward. If the collision occurs with the right bumper, the robot should initially spin 90 degrees to the left. If the collision occurs with the left bumper, the platform should spin 90 degrees to the right. If both sensors report a collision, then pick a direction and perform a 90 degree spin. After attempting to move laterally to move around the object, resume traveling forward the entire 2 meters and stop.

Keep in mind:

- 1) Your program does not need to handle the case of a robot hitting an object while in the process of going around an object.
- 2) The robot will encounter a minimum of 2 objects when demoing.
- 3) The robot must travel a total of 2m independent of how many objects it encounters.

Reminder: Comment your code to help you, your lab partner, and TAs understand you code!



You may want to update your `moveForward(...)` function to stop if it runs into an object, and return the actual distance traveled.

To get the status of the left and right bump sensors, use the `bumpRight` and `bumpLeft` members of the `oi_t` structure. These members are boolean values. A nonzero value means that the bump sensor is colliding with an object. For example:

```
oi_update(sensor_data); // get current state of all sensors
if (sensor_data->bumpLeft) {
    // respond to left bumper being pressed
}
```

Checkpoint: Demonstrate to your mentor (i.e., TA) the Cybot travelling **2m** while navigating around objects it encounters.

Part 4. Cybot communication using Putty

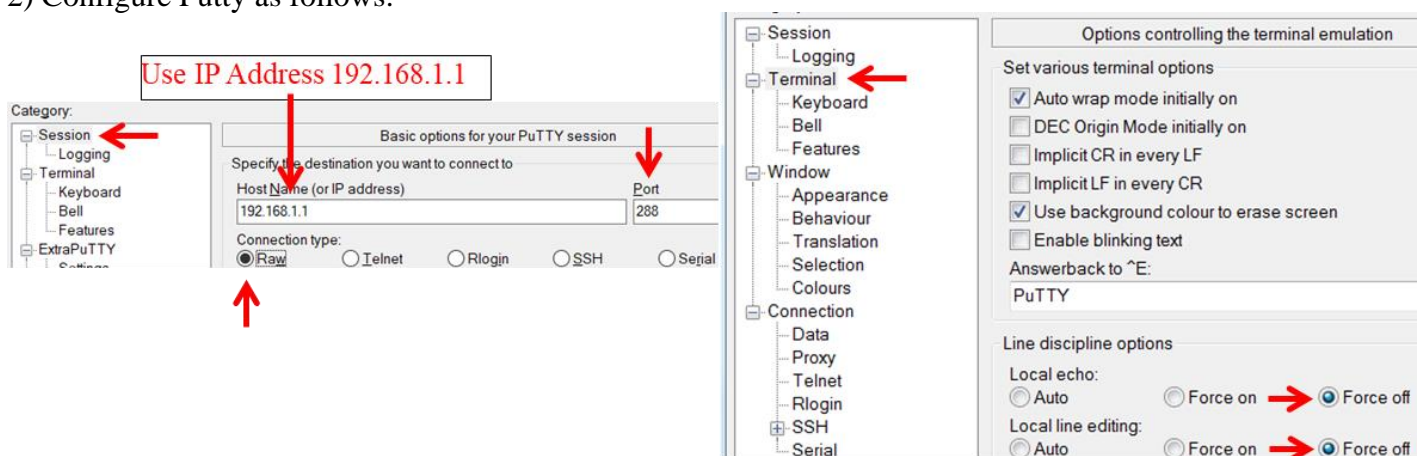
In this part of the lab, you will develop an embedded application that allows you to send messages to the Cybot to manually drive it from your Desktop computer:

- 1) Review `cyBot_uart.h`. It describes 3 functions you will use to communicate with your Cybot, using **WiFi** (or a **UART cable**) via an application called PuTTY. (**Note: pay special attention to the description of these three functions, and the implications of their behavior**)
- 2) Display to the LCD screen the ASCII character sent when you hit a key within the PuTTY application.
- 3) Send a message back to PuTTY when you press the 'm' key (e.g., "Got an m").
- 4) **Drive the Cybot**, based on what you type into PuTTY.
 - For example: w to go forward, s to go backward, a to go left, d to go right

Using Putty to interact with your CyBot over WiFi:

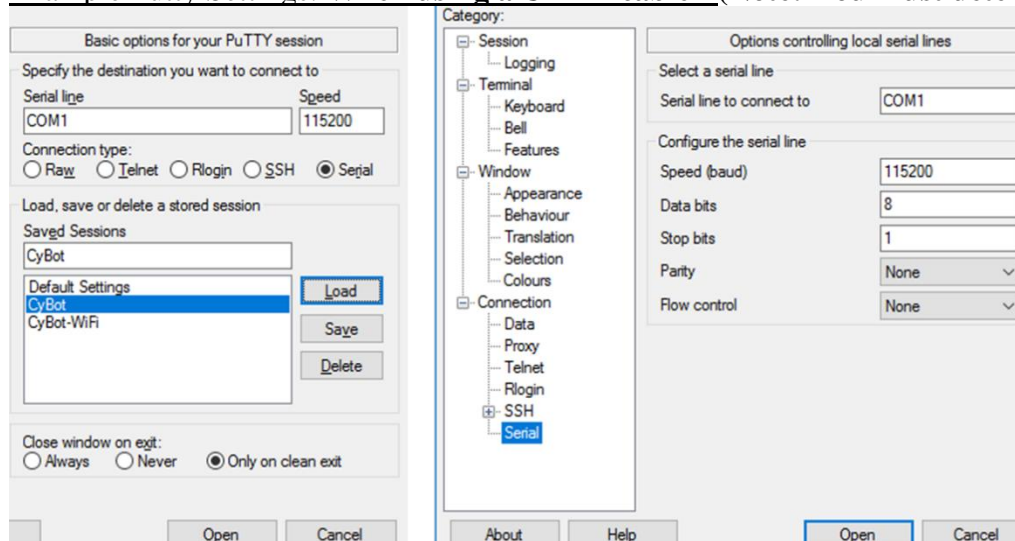
1) Make sure your PC is connected to the WiFi network for your CyBot. The network will be called: **cyBOT #** (where # is your CyBot's number), and the WiFi password is: **cp288psk**

2) Configure Putty as follows:



****Note:** When using the Virtual CyBot: 1) Skip step 1 of the WiFi setup, and 2) Configure Putty using: a) the IP address given by 288SimGUI, and port 50000

Example Putty Settings: When using a UART cable - (Note: You must determine which COM port to use)



The screenshot shows the PuTTY configuration window with the following settings:

- Basic options for your PuTTY session:**
 - Serial line: COM1
 - Speed: 115200
 - Connection type: ☒ Serial
- Options controlling local serial lines:**
 - Select a serial line: COM1
 - Speed (baud): 115200
 - Data bits: 8
 - Stop bits: 1
 - Parity: None
 - Flow control: None

The 'Category' list on the left shows 'Serial' selected under the 'Connection' category.



UART cable

Checkpoint: Demonstrate communicating between your Desktop and Cybot platform to your mentor. You should be able to send and receive messages, and drive your Cybot from your Desktop.