The Shell script user has to execute in the beginning will create a separate workspace in the home folder and download the required files from GitHub to start the activity list.

1. Running first node (Let’s get this started)

**Task**: Open a new terminal window and run the first ROS node in ‘Task1’ package.

(commands: rosrun task1 first\_node.py)

**Device response**: The LCD screen will print the numbers from 1 to 10 on it.

*[Students are used to run a program and see the output on screen computer screen itself. This activity gives the insight that the node running on the terminal is not a process doing something that is local to the PC but can be used to operate actuators separated from the working machine via ROS framework.]*

1. Running multiple nodes (Two can handle this)

**Task**: Open 2 terminal windows and execute 2 nodes in ‘Task2’ package in parallel.

**Device response**: The servo motor will do sweep action continuously. LCD will print the numbers from 1 to 10.

*[Instead of using a single large program as students used to, multiple programs (nodes) can be used to control different actions of a single robot]*

1. System stability of a robot when using ROS (To infinity and beyond)

**Task**: Run 2 ROS nodes in ‘Task3’ package. Stop the execution of ultrasonic sensor node via the physical switch to simulate a sensor failure. Then publish a message manually to the same topic via terminal.

**Device response**: Print the ultrasonic sensor’s distance measurement on the LCD screen. When sensor is terminated, last received message will keep displayed on the screen. Student will publish manual messages to the same topic which will be again displayed on the LCD.

*[This shows that distributed nature of the ROS help to increase the overall system stability. Failure of ultrasonic sensor doesn’t break the display functionality]*

1. Hardware and software abstraction (Abracadabra)

**Task**: Run a ROS node in ‘Task4’ package which will take a number between 0 to 5 which will be the rotation speed of the wheel.

**Device response**: The number will correspond to the revolutions per second of the wheel. A PID controller will try to adjust the speed of the wheel according to the user input.

*[The modular architecture helps to implement abstraction in hardware and software basis making it easier for the ROS user to create the program without worrying about the hardware and software implementations]*