Learning Aid Device for Learning Concepts Behind ROS usage in Robotics

Tharindu Dharmasena
Computer Engineering and Informatics
Middlesex University
Dubai, UAE
TP549@live.mdx.ac.uk

***Abstract*—Robot Operating System is a crucial framework that every robotics learner should learn as it is widely being used in the industry. Despite its popularity, students face difficulties when it comes to learning ROS for the first time. This report describes a device developed to aid students to grasp the core concepts of ROS usage laying behind the lessons and tutorials which will give students an insight to better understand the usage of ROS in robotics. The list of activities associated with the device will guide the students through the concepts while giving them real-world feedback via the device.**

***Index Terms*—Robots, ROS, Learning**

GitHub link: https://github.com/tharindurm/PDE4432
Demo Video: https://youtu.be/wCSWVreZd98

I. Introduction

Robot Operating System (Robot Operating System ROS) is an open-source robotics that is widely used in many fields. Learning ROS for the first time is a difficult task for many students. It is even more daunting for students with minimal exposure to computer technology related fields. Mainly due to the less exposure to the programming concepts and less familiarity with the Linux environment which is ROS mainly based on. Even the different experience levels of the students from the same discipline may have different learning experiences.

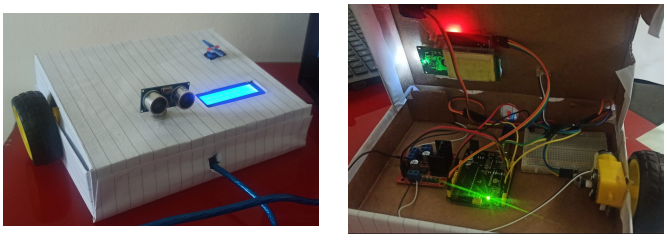
The most important question of a student which needs to be answered clearly when learning something new is; Why?. The answer to this question lays a solid foundation for the student to build the new knowledge they are gaining. This device focuses on making the student understand why we use ROS in the field of robotics rather than the inner workings of ROS. But when the students follow the activities in the list, it gives the students the opportunity to see how the programs are created and the way they behave, and why.

The student needs to have a basic understanding of ROS folder hierarchy and Linux terminal usage. A ROS workspace should be already available in the system so that the provided shell script can be executed which will download and build the downloaded packages. Before beginning the device should be connected to the PC via the USB cable. Then the 'roscore' should be running in the background in order for nodes to run correctly.

II. Hardware Components of the Device

The following components were used to build the device.

1) 1 x Arduino Uno
2) 1 x DC motor
3) 1 x Wheel Encoder
4) 1 x IR sensor
5) 1 x L298N motor controller
6) 1 x 16x2 LCD screen and I2C adapter
7) 1 x Servo motor
8) 1 x Ultrasonic Sensor



(a) The physical appearance (b) Internals

Fig. 1

The microcontroller of the device is an atmega328p on an Arduino Uno board. The 5V and ground of the Arduino board are connected to a breadboard to increase the availability of power connections to the sensors and actuators. The following sensors and actuators are connected to it via the mentioned pins.

Since the device is still in the prototyping stage, all the components have been placed inside a temporary structure made out of cardboard which gives the flexibility to change the physical design at a lower cost and internal connections with ease.

The 16x2 LCD screen with an I2C adapter displays the information the device receives via the 'lcd_msg' topic. Depending on the activity being carried out, the information displayed on the screen will be different.

The externally visible wheel is attached to a DC motor. The rotation of the dc motor is measured using an IR sen-

sor(2C40004) and an encoder wheel. To make the surface more reflective, the encoder is marked with 4 white markers so that it can be detected by the IR sensor when passing in front of it (Fig.3). The detection of the markers is sent to the Arduino as an interrupt which increases a counter variable which is used to calculate the rotating frequency of the wheel. The motor is connected to an L298N motor controller so that the motor can be powered with an external power source. The motor controller is then connected to the Arduino via an enabler pin which is used to control the speed of the motor via a PWM signal.

The servo motor available in the device takes the angle to turn from 'servo_angle' topic. This will jump in to action only when the task2 nodes are running. The node will continuously publish data to the topic so that the servo will do sweep action continuously.

The ultrasonic sensor reads the distance data and publish them back to 'lcd_msg' topic which will display the data in lcd screen.

A breadboard was used to connect VCC and GND pins of the sensors and actuators as there are not enough pins on the board to connect them all. The VCC and GND of the Arduino was extended to the breadboard and then distributed among the devices.

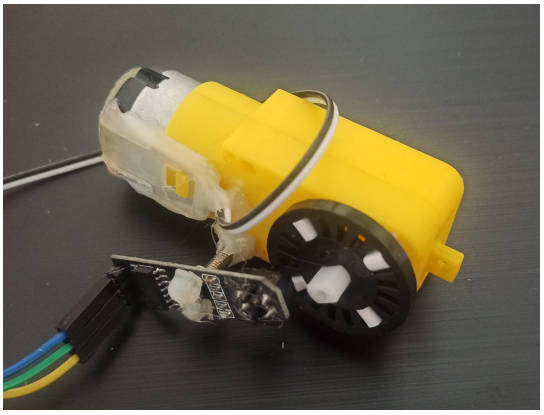


Fig. 2: Assembled wheel and encoder unit

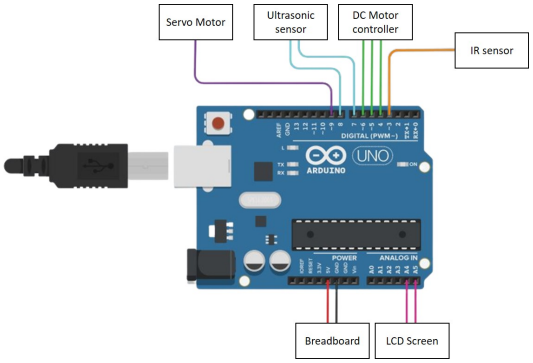


Fig. 3: Wiring Diagram

Pin	Purpose	Device
D3	signal out	IR sensor
D4 D5 D6	Motor polarity and PWM	L298N motor controller
D7 D8	Trigger Echo	2C40004 IR sensor
D9	PWM signal	Servo
A4 A5	SDA (Dataline) SCL (Clock line)	16x2 LCD screen

TABLE I: Pin allocation

III. Software of the Device

A. System Dependencies

1) ROS: The shell script only focuses on setting up the device's packages in a currently existing workspace. The operating system should be already installed with ROS and a workspace should be available.

2) rosserial: The device communicates with the computer via serial communication. To facilitate the connection 'rosserial' packages should be already installed in the system. The bellow command can be used to install 'rosserial' package and its dependencies.

apt install ros-version -rosserial-*

```
ros::Subscriber<std_msgs::Int16> servo_angle("servo_angle", &updateServoAngle);
Servo myservo;
int servoAngle = 0;

void updateServoAngle(std_msgs::Int16& data) {
  servoAngle = data.data;
}

void servoSweep() {
  myservo.write(servoAngle);
  delay(15);
}
void setup() {
  nh.initNode();
  nh.subscribe(servo_angle);
}
```

Fig. 4: ROS topic associated with serial communication

B. Ros Nodes

To set up the training ROS package, the package should be downloaded from the GitHub repository and copied into the 'src' folder of the workspace. Once the workspace is built, the nodes related to activities will be accessible via the terminal. All the nodes are written in Python.

IV. Task Implementation Methodology

The task list consist of 4 individual tasks. For each task a separate node is available which will be focused on different concepts. For second task, two separate nodes need to be executed. All the instructions the student needs to follow and the expected output and learning outcome is mentioned in the lists of tasks document.

A. Task1

Most of the students are used to run a program and see the output of the program on screen computer screen itself. This activity gives student, 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.

This node acts as a publisher and publishes the numbers from 0 to 10 to the topic 'lcd_msg'. The Arduino subscribed to 'lcd_msg' topic will read this information and then sends it to the LCD screen using following code segments. The following code segment shows only the important codes in the Arduino program.

```python
def talker():
    pub = rospy.Publisher('lcd_msg', Int16, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(0.5)
    count = 0

    while not rospy.is_shutdown():
        pub.publish(count)
        print("Published number ",str(count)," to ROS topic '/lcd_msg'")
        if count == 10:
            count = 0
            print("---------------------------------------------- Restarting Loop ----------")
        else:
            count=count+1
        rate.sleep()
```

Fig. 5: Important Python code segments for Task 1

```
ros::Subscriber<std_msgs::Int16> lcd_message_number("lcd_msg", &updateDisplayNumber);
int displayNumber = 0;

void updateDisplayNumber(std_msgs::Int16& data) {
  displayNumber = data.data;
}

void updateDisplayNumber(std_msgs::Int16& data) {
  displayNumber = data.data;
}
void setup() {
  nh.initNode();
  nh.subscribe(lcd_message_number);
}
```

Fig. 6: Important Arduino code segments for Task 1

B. Task2

This task shows that 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 which makes the development of the code easier and increase readability of the code. 2 nodes needs to be executed from the PC which will send messages to the LCD screen and also will publish angles for the servo motor to rotate.

C. Task3

This tasks is focused on explaining that distributed nature of the ROS help to increase the overall system stability. Te student is expected to break the power line for the ultrasonic sensor via a mechanical switch. This will stop the distance values being displayed in the screen. Then the student is tasked to publish data manually to the 'lcd_msg' topic. This shows that the failure of ultrasonic sensor does not cause the display to loose its functionality.

```
ros::Subscriber<std_msgs::Int16> servo_angle("servo_angle", &updateServoAngle);
Servo myservo;
int servoAngle = 0;

void updateServoAngle(std_msgs::Int16& data) {
  servoAngle = data.data;
}

void servoSweep() {
  myservo.write(servoAngle);
  delay(15);
}
void setup() {
  nh.initNode();
  nh.subscribe(servo_angle);
}
```

Fig. 7: Important Arduino code segments for Task 2

D. Task4

The modular architecture of ROS 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. This task focuses on spinning a wheel at a given speed. The student will publish a desired speed in the given range and the program will adjust its PWM signal to match the given speed.

V. Results and Challenges

The ROS header files which need to be included in the Arduino program take a lot of space from the dynamic memory available in the Arduino Uno. This limits the number of variables the developers can use and number of header files they can import into the program which will limit the amount of work that the Arduino can do.

Memory Type	Size	Percentage
Dynamic memory	1799/2048	87%
Program storage	14340/32256 bytes	44%

TABLE II: Memory usage

It was noted that once the Arduino program is uploaded and the rosserial node is running on the PC, the new programs cannot be uploaded back to the Arduino as the serial channel is being used by ROS. In an event where the Arduino program needs to be updated, rosserial communication have to be stopped first.

As the DC motor used in this project was a hobby DC motor, it showed poor performance while spinning. It failed to maintain a constant speed despite a constant PWM signal being received. This caused PID controller to behave in unexpected ways and finally decided to use only a P controller despite its steady state error.

A major difference in the way ROS topics behave in PC and in Arduino is that, due to serial communication protocols, the data from Arduino topics will not publish and subscribe data at the same time. This caused noticeable delays in the actuators connected to the Arduino board. For example the servo sweep will pause for a brief moment when the lcd_msg topic receives a message to be displayed.

When all the sensors are connected to the Arduino, the board cannot provide power for all the sensors at the same

time. This is mainly due to lack of VCC pins and the maximum current limit Arduino can handle which is 400mA via its VCC[1] pin. But since only a limited number of sensors or actuators are running at a time depending on the task maximum current limit is not an issue. To overcome the lack of VCC pins, the VCC pin of the Arduino was extended to a bread board. All the VCC pins and GND pins of the sensors and actuators were then connected to the breadboard.

VI. Conclusion

This project focused on creating a device which will help students to understand the concepts behind the usage of ROS framework. The amount of tasks are limited due to the technical limitations of the Arduino uno such as maximum current limit, limited I/O pins and the quality of the sensors used in the prototype such as DC motor which can not maintain a constant speed with a constant PWM signal. The rosserial package is a great way to connect Arduino to ROS instead of writing manual serial communication codes.

As future improvements, the Arduino board can be replaced with a board with a higher specs such as Arduino Mega which will provide more I/O pins to connect sensors and actuators while increasing the amount of dynamic memory available for the programmer. The actuators used in the project can be replaced with high quality devices so that the readings will be accurate and the performance will be consistent.

References

[1] ”How much current can I draw from the Arduino’s pins?.” [Online]. Available: https://electronics.stackexchange.com/questions/67092/how-much-current-can-i-draw-from-the-arduinos-pins[Accessed: 06-Jan-2023]

[2] ”Interface an I2C LCD with Arduino” [Online]. Available: https://lastminuteengineers.com/i2c-lcd-arduino-tutorial/ [Accesses: 14-Dec-2022]

[3] ”Interface L298N DC Motor Driver Module with Arduino” [Online]. Available: https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/ [Accessed: 05-Jan-2023]

[4] ”Arduino - Motor PID speed control” [Online]. Available: https://engineer2you.blogspot.com/2016/12/arduino-motor-pid-speed-control.html [Accessed: 20-Dec-2022]