A Guide to Building a ROS2-based Mobile Robot Platform: SMARTmBot

Jaeeun Kim, Wonse Jo, and Byung-Cheol Min

***Abstract*— This paper is to introduce a new, flexible, scalable, and fully open-source mobile robot platform for a multi-robot research applications. The mobile robot is 15 cm in diameter and 10 cm in height and supports the Robot Operating System 2 (ROS2). The hardware is composed of three layers of printed circuit board (PCB). The first layer of the PCB has two DC motors with a motor driver chip and two line sensors to detect the cliff. The second layer of the PCB has the Raspberry Pi and eight times-of-flight (ToF) sensors which are to accurately measure the distance with obstacles. The third layer has four RGB LEDs and a speaker. Thus, the robot platform is able to change its light either individually or as a whole, play music, and measure the distance of nearby obstacles using ToF sensors. Furthermore, we provide source codes used on the mobile robot platform. In the future, we will use the platform to study on effect of multi-robot system on human perception and cognitive states, and disseminate it for K-12 students to easily learn robotics.**

I. INTRODUCTION

With recent advances in robot technology, Science, technology, engineering, and mathematics (STEM) education using robot platforms has been a rising research field. The major object of STEM education using robots is to provide students with the opportunity to learn various programming languages, such as C /C++/Python computer languages. In addition, communication modules, such as Inter-Integrated Circuit (I^2C), Controller Area Network (CAN), Serial Peripheral Interface (SPI) communications made to easily read and control electronic components, so the single-board computer (SBC) can read sensors values via SPI ADC and control DC motors of the robot. However, students who learn STEM education still encounter large entry barriers due to initial funding for developing robot platform and complex assembly and programming methods.

To address these challenges, various research institutes, universities, and businesses are launching various STEM tools. An example of the mobile robot platforms is Donkey Car, which provides the learning tools for driving an RC car and process data collected with it [1]. Similarly, Miniskybot aims to be a learning tool that is cheap and easy to build [2]. It has 3D printed parts and its own processing board, which is controlled by a PC. Although there are aforementioned mobile platforms that allow the user to build from scratch and modify as they need, there is still a huge barrier for students to learn STEM educations, as well as for the researchers who are not in robotics to conduct human-robot interaction (HRI).

Jaeeun Kim, Wonse Jo, and Byung-Cheol Min are with SMART Lab, Department of Computer and Information Technology, Purdue University, West Lafayette, IN 47907, USA kim2592@purdue.edu, jow@purdue.edu, minb@purdue.edu

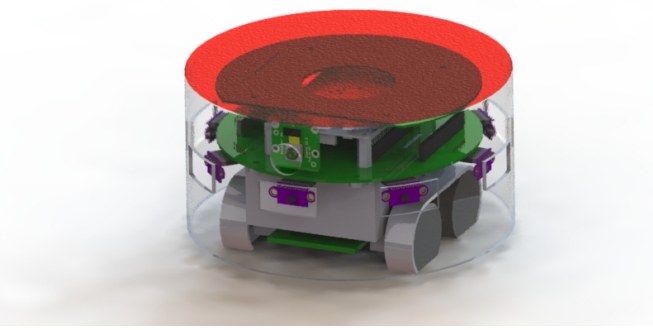


Fig. 1: CAD model of the fully assembled SMARTmBOT

Therefore, we propose a new open-source mobile robot platform, called SMARTmBOT that can be used for education and HRI research purposes. The SARTmBOT is made with a 3D printed casing and using a Raspberry Pi 4 installed Robot Operating System 2 (ROS2) [3] to control the devices and read sensors mounted on three layers of the printed circuit board (PCB). The sensors and visual output components are mounted on the PCBs, such as RGB LEDs, Time of Flight (ToF) sensors, photo reflector sensors, and DC motors. The power source for SMARTmBOT is a portable power bank, and the rest of the sensors and components are powered by the Raspberry Pi 4. As well, we elaborated specific details of software in Section III.

II. HARDWARE

In this section, we will explain how to solder and assemble the robot. The electronic section of the robot will be explained layer by layer. The materials used in the SMARTmBOT are shown in Table I.

A. 3D designs

The SMARTmBOT is 15 cm in diameter and 10 cm in height, and composed of 3D printed parts as shown in Figure 2. All parts should be completed before assembling parts.

Figure 2a is the transparent case to cover the whole robot platform and illuminate color lights using RGB LEDs mounted on the third layer of PCB. The required quantity is one.

Figure 2b is the power bank holder that is a fundamental part of the SMRTmBOT. The part is to mount the power bank and connect other PCB layers. The required quantity is one.

Figure 2c is The motor mount bracket to fix two DC motors on the power bank holder. The perforated hole is

TABLE I: BOM list for the SMARTmBOT

#	Item	Quantity
1	Portable Charger	1
2	DC Gear Motor	2
3	32GB SD Card	1
4	Raspberry Pi4 4GB	1
5	22T Track Wheel Set	1
6	RPR220 Photo Reflector Sensor	2
7	VL53L0X Time-of-Flight Laser Ranging Sensor	8
8	5050 SMD RGB LED	4
9	MCP3008	1
10	L239D	1
11	2.5mm JST Female Connector	3
12	2.5mm JST Male Connector	3
13	USB-A to USB-A Connector	1
14	USB-A to USB-B Connector	1
15	FPC Connector	2
16	FPC Cable	1
17	100 Ohm Resistor	3
18	200 Ohm Resistor	2
19	68k Ohm Resistor	2
20	90 Degree Angle Pin Header	8
21	20 Pin Female Header	1
22	10 Pin Female Header	4
23	10 Pin Male Header	2
24	Spacer	8
25	Bolts	20
26	Nuts	12

to pass and arrange a power cable and cables of the DC motors. The required quantity is one.

Figure 2d is the shaft support bracket to fasten the the shaft of belt chain on the power bank holder. The required quantity is two.

All design and 3D printing files can be downloaded from: *https://github.itap.purdue.edu/ByungcheolMinGroup/SMARTmBOT/tree/master/Hardwar_design*

B. PCB design

1) First Layer: The first layer of the PCB shown in Figure 3a is located between the wheels, facing downwards. It includes a motor driver, L293D, ADC-SPI chip, MCP3008, and two photo reflector sensors, RPR220.

Turn the layer so the half-circle is at the top. Place the parts as indicated in Figure 3a and solder them. Make sure you are at the right side of the board by verifying with the silkscreen. When done soldering with this side, flip the board to solder the FPC connector on the surface.

Also, solder the male JST connector to the DC motors as well as to a USB-A header. This cable will we connected to the Raspberry Pi 4 and power the motors.

2) Second Layer : is placed on top of the battery housing, holding mounts for Raspberry Pi and ToF sensors. Place the board so the side with the silkscreen is on top. Place the parts as indicated in Figure 3b and solder them. Refer to Figure X to make sure that the pin headers for the ToF sensors are in the right orientation.

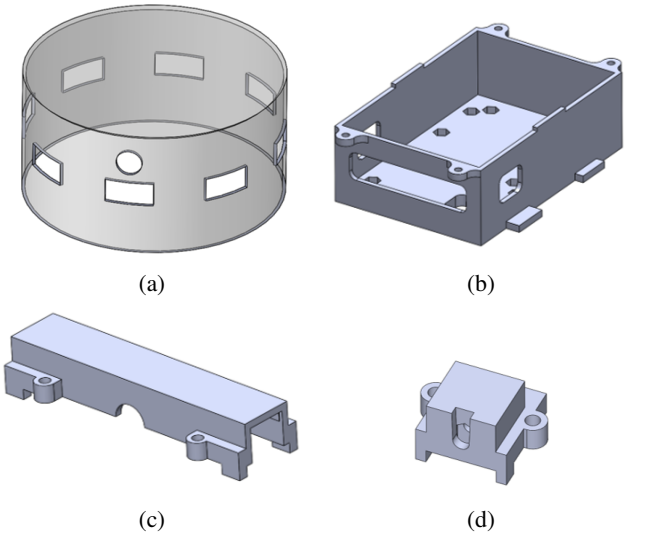


Fig. 2: 3D printing parts for SMARTmBOT; (a) Transparent case (1EA), (b)Power bank holder (1EA), (c) Motor mount bracket (1EA), and (d) Shaft support bracket (2EA).

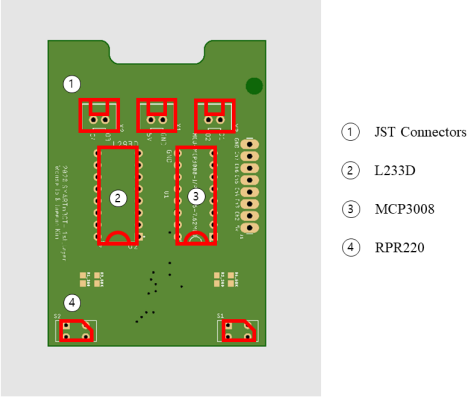
3) Third Layer : is placed on top of the second layer, connected with pin headers. It has RGB LEDs and male pin headers, then place the board so the side with the silkscreen is on top as indicated in Figure 3c and then solder them. Refer to Figure X to make sure that the pin headers are in the right orientation.

4) Custom Layer : provides extra pins on the third layer PCB to connect an additional layers.

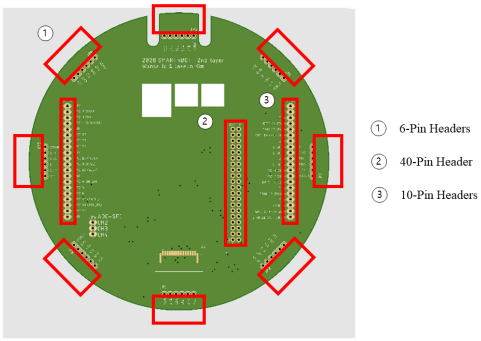
C. Assembly Instructions

With every layer soldered, prepare the 3D printed parts.

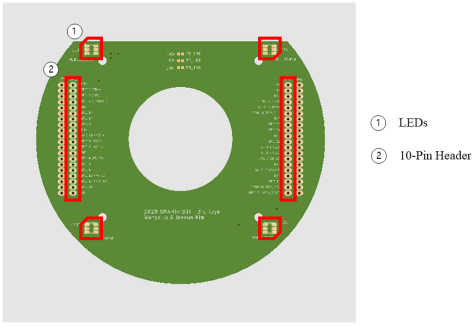
1) Screw four spacers to power bank holder as in Figure 5a.
2) Place two wheels to shaft support brackets as shown in Figure 5b.
3) Place the two DC motors at each end of motor mount bracket as in Figure 5c. The JST cables attached to the motors should be left out of the housing through the half-circle hole.
4) Secure then brackets with nuts and bolts so the finished product looks like Figure 5d.
5) Add the assembled first layer on top of the spacers and screw it on them (Figure 5e).
6) Place the portable charger inside the battery holder (Figure 5f).
7) Fit the assembled second layer's square holes to the battery holder. Secure it with bolts and nuts (Figure 5g).
8) Fit the Raspberry Pi's pins to the second layer's 20-pin header (Figure 5h).
9) With USB-A to USB-B cable, connect one of the ports of the powerbank to the power input of Raspberry Pi.



(a) First PCB layer

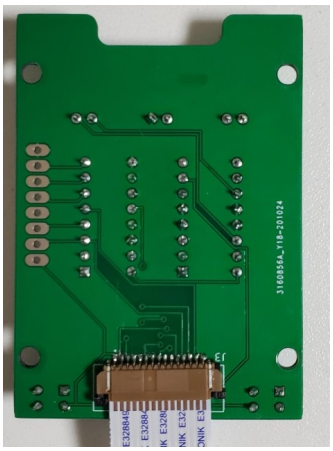


(b) Second PCB layer

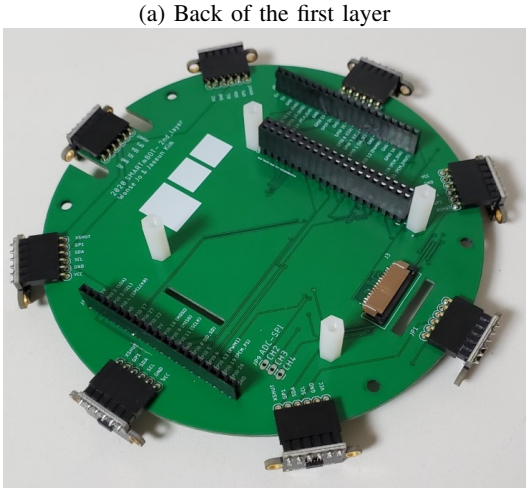


(c) Third PCB layer

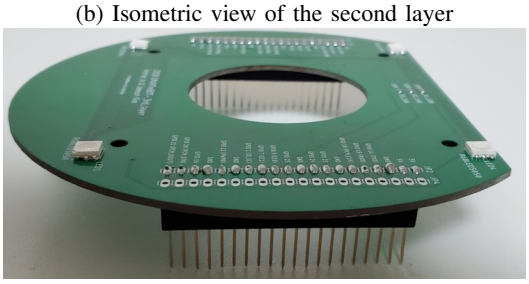
Fig. 3: Gerber design files for the PCB layers;



(a) Back of the first layer



(b) Isometric view of the second layer



(c) Isometric view of the third view

Fig. 4: Three simple graphs

10) With USB-A to 2.5mm JST cable, connect one of the USB output ports of Rasberry Pi to the middle of the connectors on the first layer.
11) Connect the two FPC connectors, one in the first layer and another in the second layer, with a FPC cable.
12) Fit 10-pin female headers to 10-pin male headers on the third layer. The pins should be extended as shown in Figure 4c.
13) Fit the extended pins on the 10-pin female headers on the second layer (Figure 5i).

The fully assembled robot should resemble Figure 5j.

III. Software

In this section, we will explain how to configure the Raspberry Pi to run the program and manipulate the inputs and outputs.

A. *Configuring Raspberry Pi*

To begin, prepare a SD card and install Ubuntu 18.04 Server and run the image on the SD card. Then insert the SD card to the Raspberry Pi and log in with the default credentials (Username: ubuntu, password: ubuntu). A monitor and keyboard may be necessary. Activate the WiFi by typing those lines in the terminal in order from Code 1 to Code 3.

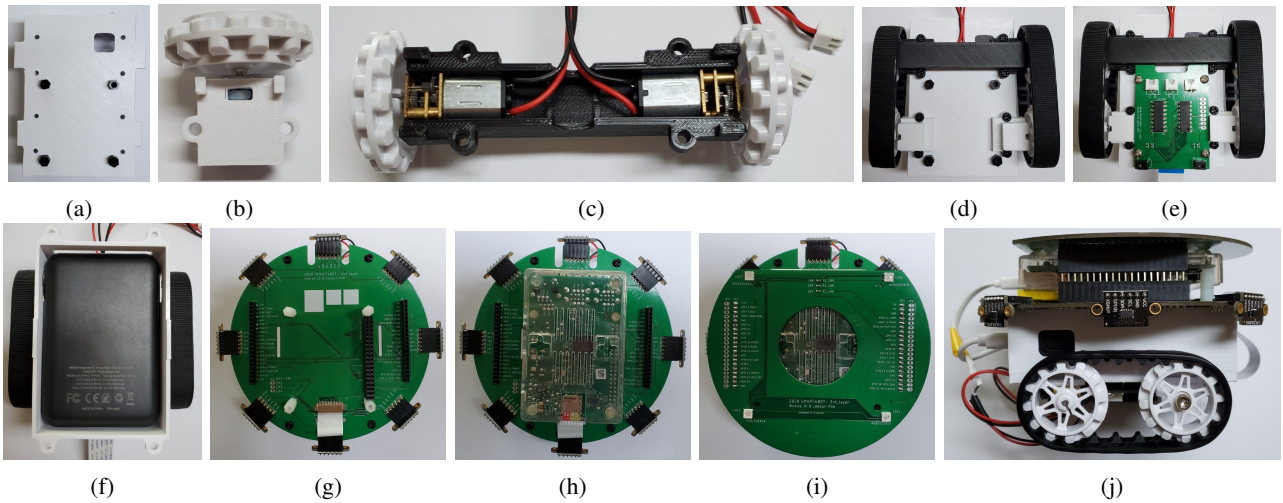


Fig. 5: Assembly instructions; (a) Battery holder with spacers, (b) Shaft support bracket with a wheel, (c) Motor mount with the motors, (d) Battery holder with spacers, (e) Battery holder with first layer, (f) Battery holder with power bank, (g) Second layer added, (h) Raspberry Pi added, (i) Third layer added, and (j) Side view of completed robot.

Code 1: Opening the netplan file

```
$sudo touch /etc/netplan/01-netcfg.yaml
$sudo nano /etc/netplan/01-netcfg.yaml
```

Code 2: Editing the netplan file

```
network:
  version: 2
  renderer: networkd
  wifis:
    wlan0:
      dhcp4: yes
      dhcp6: yes
      access-points:
        "your-wifi-name":
          password: "your-wifi-password"
```

Code 3: Applying the revised netplan file

```
$ sudo netplan apply
$ reboot
```

B. Installing ROS 2-Dashing Diademata on Raspberry Pi

Access the Ubuntu terminal installed in the Raspberry Pi. Follow the official instruction in ROS2 website [4].

1) Setup a locale to support UTF-8.

```
$locale  # check for UTF-8
$sudo apt update && sudo apt install locales
$sudo locale-gen en_US en_US.UTF-8
$sudo update-locale LC_ALL=en_US.UTF-8 LANG=
    en_US.UTF-8
$export LANG=en_US.UTF-8
locale  # verify settings
```

2) Get a GPG key of the ROS 2 to register the ROS2 apt repositories on the system.

```
$sudo apt update && sudo apt install curl
    gnupg2 lsb-release
$curl -s https://raw.githubusercontent.com/ros
     /rosdistro/master/ros.asc | sudo apt-key
     add -
```

3) Install ROS 2 packages

```
$sudo apt update
$sudo apt install ros-dashing-base
```

4) Set up the ROS 2 environment on the local system

```
$source /opt/ros/dashing/setup.bash
```

In order to validate the installation of the ROS 2, we provides a subscriber and publisher examples. This example is to create a new subscriber and publisher node as follows:

Code 4: An example of ROS2 subscriber and publisher

```

import rclpy
from rclpy.node import Node
from std_msgs.msg import String, Int32

class Minimal_Subscriber_Publisher(Node):
    def __init__(self):
        super().__init__( '
    Minimal_Subscriber_Publisher')
        # Initialize and define Publishers
        self.pub_name = self.create_publisher(Int32
    , 'wonsu0513/pub_test', 10)

        # Setting timer for pulbisher
        pub_name_timer_period = 0.5  # seconds
        self.pub_name_timer = self.create_timer(
    pub_name_timer_period, self.
    pub_name_timer_callback)
        self.pub_name_i = 0

        # Initialize and define subscription (
    subscribe)
        self.sub_name = self.create_subscription(
            Int32,'wonsu0513/pub_test', self.
    sub_name_callback, 10)
        self.sub_name  # prevent unused variable
    warning
    def sub_name_callback(self, msg):
        self.get_logger().info('(sub) I heard: "%d"
    ' % msg.data)
    def pub_name_timer_callback(self):
        msg = Int32()
```

```
        msg.data = self.pub_name_i
        self.pub_name.publish(msg)
        # display data on the terminal (it is good to test the code)
        self.get_logger().info('(pub): "%d"' % msg.data)
        self.pub_name_i=self.pub_name_i+1

def main(args=None):
    rclpy.init(args=args)
    minimal_subscriber_publisher = Minimal_Subscriber_Publisher()
    rclpy.spin(minimal_subscriber_publisher)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_subscriber_publisher.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Then, the codes should be built on the ROS 2 environment as follows:

Code 5: Building the ROS2 environment on the ROS2 working space

```
$ cd ~/ros2_ws
$ colcon build --symlink-install
$ source install/setup.bash
```

C. Installing the Package with a Launch File

Complete the following steps to install the necessary dependencies and codes. Do so by copy and pasting the following codes to the Ubuntu terminal, unless specified otherwise.

1) Install Dependencies

```
$ pip install RPi.GPIO
```

2) Create Workspace and Download the Package

```
$ mkdir -p ~/smart_mbot && cd ~/smart_mbot
$ git clone https://github.itap.purdue.edu/ByungcheolMinGroup/smart_mbot_ws.git
```

3) Build the Package

```
$ echo 'source /opt/ros/dashing/setup.bash' >> ~/.bashrc
$ colcon build --symlink-install
$ source install/setup.bash
$ ros2 pkg create --build-type ament_python smart_mbot
```

D. Control the LEDs with GPIO Pins

The LEDs can emit red, green, or blue light. Each color is connected to a separate GPIO pin, so the user can turn individual or multiple colors at the same time. The pins used for the LEDs are 7, 8, and 25. Code 6 is the part of LED code included in the package. To control the LED colors, set the output of the wanted color's pin to HIGH by changing the GPIO.LOW to GPIO.HIGH.

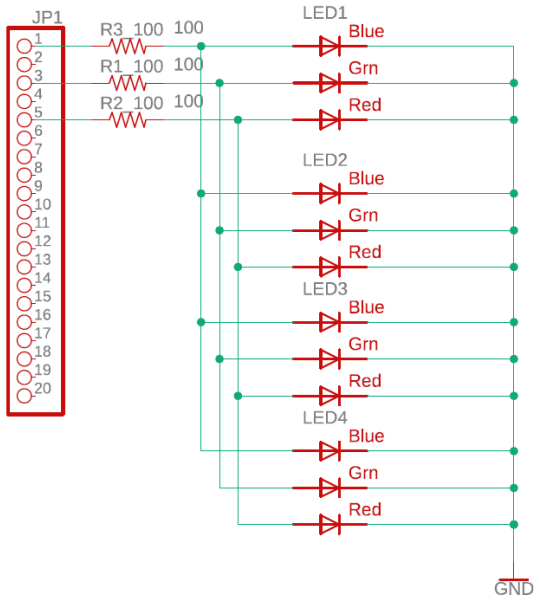


Fig. 6: Schematic of four LEDs (not an exhaustive diagram)

Code 6: GPIO control of the Raspberry Pi for LEDs

```
GPIO.output(pin, GPIO.LOW)
```

E. Read ADC via SPI Node

The photo reflector sensors used in this robot, RPR220, are analog sensors which signals are converted into digital signals and read on SPI bus via MCP3008 chip. The pin used for the sensor is 5.

The code in the package will convert the analog data to voltage values, stored in an array. Below is the part of Code 7 used for reading data from the sensors. The data is automatically published as ROS2 message *Float32MultiArray* as 'smartmobil/ADC'.

Code 7: Example code to make a publisher of ROS2

```
# Initialize and define Publishers
self.pub_adc_spi_reader = self.create_publisher(
    Float32MultiArray, 'smartmobil/ADC', 10)

# Setting timer for publisher
pub_adc_spi_reader_timer_period = 0.001 #sec
self.pub_adc_spi_reader_timer =
self.create_timer(pub_adc_spi_reader_timer_period,
    self.pub_adc_spi_reader_timer_callback)
```

F. Control DC Motors

We used L293D motor driver chip to control two DC motors with pulse-width modulation (PWM). The pins used for first motor is 37, 33, 35, and second motor is 36, 38, 40. The last pins listed in each motors are used for PWM control.

The code 8 is displaying the equation used to calculate each wheel's velocity with given linear and angular velocity. WHEEL_BASE is the distance between the two wheels and WHEEL_RADIUS is the radius of the wheel.

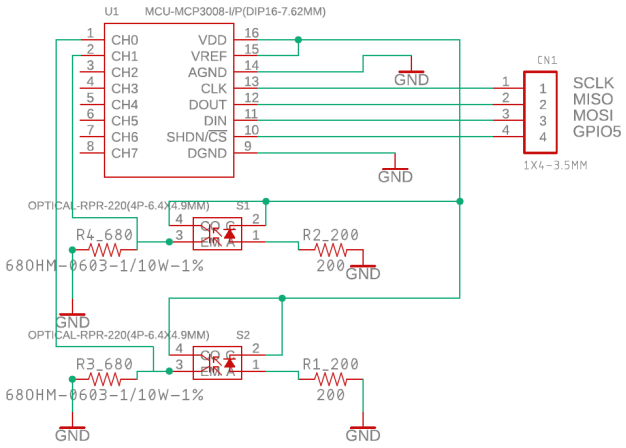


Fig. 7: Schematic of SPI-ADC wiring with photo reflector sensors (not an exhaustive diagram)

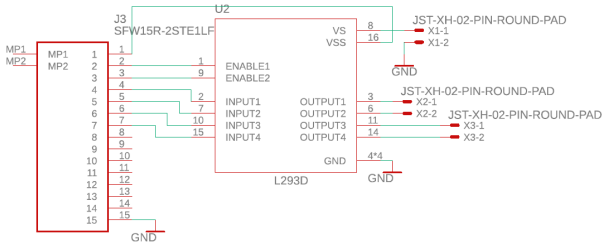


Fig. 8: Schematic of motor driver with two DC motors and an external power source (not an exhaustive diagram)

Code 8: Calculating of each wheel's velocity using linear and angular velocity

```
self.right_velocity = abs((self.lin_vel - self.
    ang_vel * WHEEL_BASE / 2.0) / WHEEL_RADIUS)
self.left_velocity = abs((self.lin_vel + self.
    ang_vel * WHEEL_BASE / 2.0) / WHEEL_RADIUS)
```

G. Read ToF Sensors

The eight ToF sensors or on a I^{2}C bus, each with their unique I^{2}C addresses. The pins used for the sensors are 13, 18, 16, 15, 12, 11, 8, and 10.

Code 9 is the part of the code used for reading data from the sensors. The data is automatically published as ROS2 message *Int32MultiArray* as 'smartmobil/I2C'.

Code 9: Example of I^{2}C code to read ToF sensors

```
# Initialize the i2c module
self.i2c = board.I2C()
self.pub_i2c_tof_reader = self.create_publisher(
    Int32MultiArray, 'smartmobil/I2C', 10)
```

IV. ROS 2 Launch

Code 10 is a ROS2 launch file to run a group of nodes at a same time. For this application, the launch file included in the package runs motor, SPI, and I^{2}C nodes together.

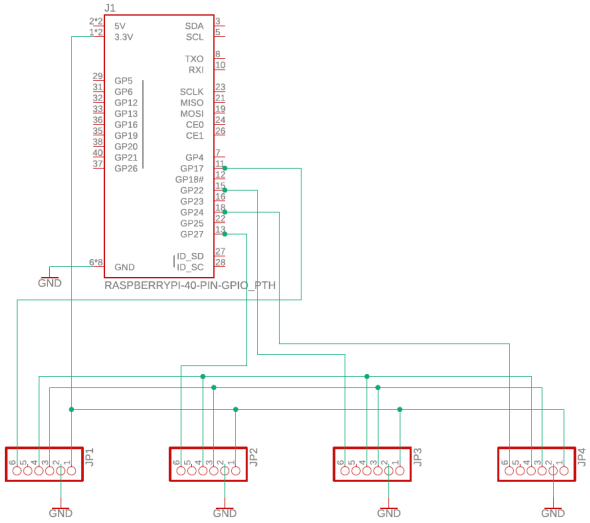


Fig. 9: Schematic of I^{2}C wiring with four ToF sensors (not an exhaustive diagram)

TABLE II: ROS2 Topic Subscriber List

Name	Type	Description
cmd_vel	*Twist*	Linear and Angular Velocity for the Motors

Code 10: Launch file to control SMARTmBOT

```
from launch import LaunchDescription
from launch_ros.actions import Node
import launch_ros.actions

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='smart_mbot_pkg',
            node_executable='MOTOR',
            node_name='motor_sub'
        ),
        Node(
            package='smart_mbot_pkg',
            node_executable='SPI',
            node_name='ADC_SPI_Reader',
        ),
        Node(
            package='smart_mbot_pkg',
            node_executable='I2C',
            node_name='I2C_ToF_Reader',
        )
    ])
```

The code below is to run the launch file when typed in the terminal. Run the command in the workspace directory.

```
ros2 launch smart_mbot_ws mbot.launch.py
```

There are one subscriber and two publishers in the package. Since they are in ROS2 message formats, the user can write to the publisher and read the publishers by accessing the topics with appropriate messages. The topics are listed and explained further in Table II and Table III.

As an example, it is possible to directly publish a message to cmd_vel to control the motors. The message type Twist accepts linear and angular velocity as a form of linear: x: 0.0,

TABLE III: ROS2 Topic Publisher List

Name	Type	Description
smartmobil/I2C	*Int32MultiArray*	Array of ToF Sensor Values
smartmobil/ADC	*Float32MultiArray*	Array of Photo Reflector Sensor Values

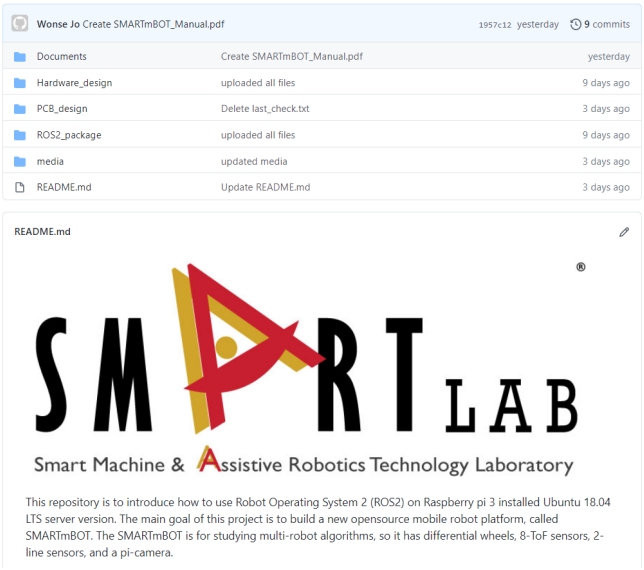


Fig. 10: SMARTmBOT GitHub page

y: 0.0, z: 0.0, angular: x: 0.0, y: 0.0, z: 0.0. When running in the terminal, the command should look like Code 11.

Code 11: Controlling motors of SMARTmBOT

```
ros2 topic pub --once /cmd_vel geometry_msgs/msg/
    Twist "{linear: {x: 5.0, y: 0.0, z: 0.0},
    angular: {x: 0.0, y: 0.0, z: 2.0}}"
```

The code for controlling motors will dissect the message and calculate appropriate speed for each motor and output them as PWM values.

V. Online Repositories

The GitHub repository can be found in the following link: *https://github.itap.purdue.edu/ByungcheolMinGroup/SMARTmBOT*. In the repository, there are 3D design CAD files, PCB design files, and ROS2 package utilized on the SMARTmBOT. The 3D design files are created by SolidWorks 2020, the PCB designs are made by AutoDesk Eagle CAD, and ROS2 nodes are based on Python3.

VI. Conclusion and Future Works

In this paper, we presented an open-source mobile robot platform to serve as a learning tool for K-12 student. The components are relatively cheap and mainstream so that they are easy to obtain, and this manual will help the students to assemble and modify the codes.

In the future, we will conduct a new user study to investigate the effect of the multi-robot system on human perception and cognition. And also, we will share the mobile platform with K-12 students to give opportunities to learn human-robot interaction via virtual meeting and/or in-person meeting.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1846221. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Autorope, "donkeycar," https://github.com/autorope/donkeycar, 2020.
[2] J. Gonzalez-Gomez, A. Valero-Gomez, A. Prieto-Moreno, and M. Abderrahim, "A new open source 3d-printable mobile robotic platform for education," in *Advances in autonomous mini robots*. Springer, 2012, pp. 49–62.
[3] "Ros 2 documentation¶." [Online]. Available: https://index.ros.org/doc/ros2/
[4] "Installing ros 2 dashing diademata¶." [Online]. Available: https://index.ros.org/doc/ros2/Installation/Dashing/