ROMR: A ROS-based Open-source Mobile Robot

Nwankwo Linus*, Fritze Clemens, Konrad Bartsch, Elmar Rueckert

Abstract

Currently, commercially available intelligent transport robots that are capable of carrying up to 90kg of load can cost \$5,000 or even more. This makes real-world experimentation prohibitively expensive, and limiting the applicability of such systems to everyday home or industrial tasks. Aside from their high cost, the majority of commercially available platforms are either closed-source, platform-specific, or use difficult-to-customize hardware and firmware. In this work, we present a low-cost, open-source and modular alternative, referred to herein as "ros-based open-source mobile robot (*ROMR*)". *ROMR* utilizes off-the-shelf (OTS) components, additive manufacturing technologies, aluminium profiles, and a consumer hoverboard with high-torque brushless direct current (BLDC) motors. *ROMR* is fully compatible with the robot operating system (ROS), has a maximum payload of 90kg, and costs less than \$1500. Furthermore, *ROMR* offers a simple yet robust framework for contextualizing simultaneous localization and mapping (SLAM) algorithms, an essential prerequisite for autonomous robot navigation. The robustness and performance of the *ROMR* were validated through real-world and simulation experiments. All the design, construction and software files are freely available online under the GNU GPL v3 license at https://doi.org/10.17605/OSF.IO/K83X7. A descriptive video of *ROMR* can be found at https://osf.io/v8tq2.

Keywords

Mobile robot, ROS, open-source robot, differential-drive robot, autonomous vehicle, open shuttle

Table 1: Specification table

Hardware name	ROS-based Open-source Mobile Robot (*ROMR*)
Subject area	• Robotics • Sensor fusion • Simultaneous localisation and mapping (SLAM) • Navigation • Teleoperation • Research and development in robotics • General
Hardware type	• Mechatronic • Robotic
Open source license	GNU GPL v3
Cost of hardware	< \$1500
Source file repository	https://doi.org/10.17605/OSF.IO/K83X7

1. Hardware in context

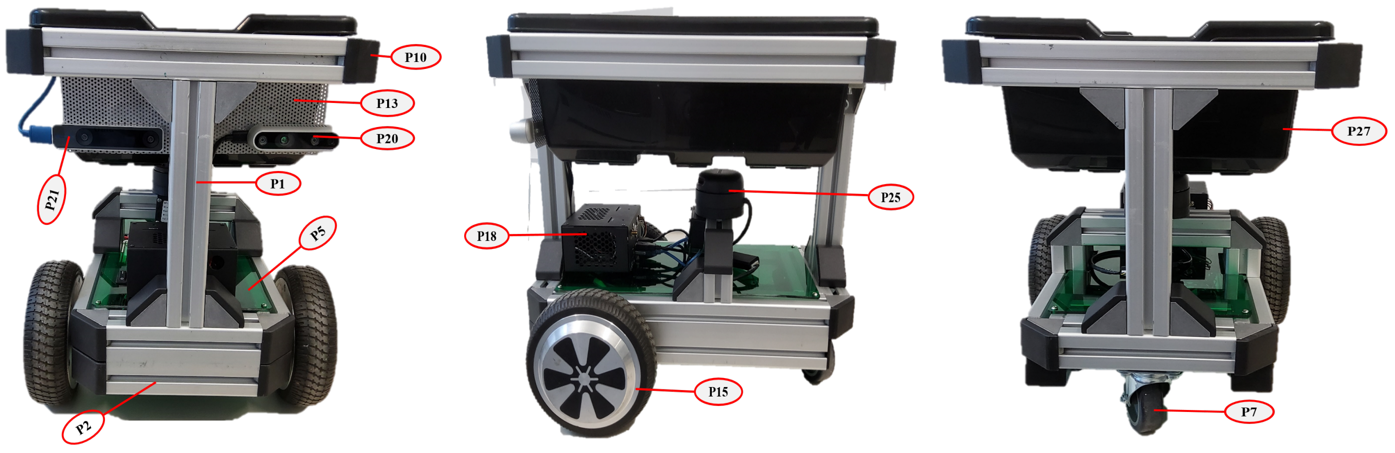
Intelligent transport robots (ITRs) are becoming an integral part of our daily activities in recent years [34], [13], [4]. Their application for day-to-day activities especially in industrial logistics [23], warehousing [7], household tasks [40], [26], etc., provides not only a cleaner and safer work environment but also helps to reduce the high costs of production. These robots offer the potential for a significant improvement in industrial safety [37], productivity [27] and general operational efficiency [1]. However, several challenges such as the reduced capacity [25], [1], affordability [8], and the difficulties in modifying the inbuilt hardware and firmware still remain [11].

Although, significant effort has been undertaken by the scientific community in recent years to develop a standardised low-cost mobile platform, there is no open-source system that fulfills the high industrial requirements. Most available open-source, low-cost platforms are still limited in their functions and features, i.e., they are commercially not available, do not match the required payloads for logistic tasks, or cannot be adapted. For example, while [15], [2], and [16],are inexpensive, they cannot be used in day-to-day tasks that require transporting materials of high loads of 5kg or more.

On the other hand, many commercially available industrial platforms exist that feature high payloads, see Table 2 for an overview. Unfortunately, they are expensive, closed-source, and platform-specific with inbuilt hardware, and firmware that may be difficult to modify [16]. This restrains many users' ability to explore multiple customisation or reconfiguration options to accelerate the development of intelligent systems.

Consequently, there is a crucial need to have low-cost robots with comparable features that can easily be scaled to adapt to any useful purpose. To this end, we propose *ROMR*, a modular, open-source and low-cost alternative for general purpose applications including research, navigation [28], and logistics.

ROMR is fully compatible with ROS, has a maximum payload of 90kg and costs less than \$1500. It features several lidar sensor technologies for potential application for perception [19], simultaneous localisation and mapping [5], [36], deep learning tasks [35], and many more. Figures 1 and 8 present the pictorial view and the cyber-physical anatomy of the *ROMR* respectively.



(a) Front view. (b) Side view . (c) Back view.

Figure 1: The *ROMR* is built from a consumer hoverboard wheels with a high torque brushless direct current (BLDC) motors. It utilises Arduino Mega Rev3, a Nvidia jetson Nano, and the components described in Tables 5 and 6.

1.1 Related hardware platforms

In Table 2, we present a comparative evaluation with *ROMR*. Similar hardware that was developed in recent years for research, navigation and logistics applications. Our comparison focused on some key features that define the open-sources, robustness and versatility of the robot design, e.g., ease of reconfiguration or modification, cost performance, load carrying capacity, and full compatibility with the ROS [29].

In Table 2, the ROS feature indicates whether the robot is fully compatible with ROS or not. Custom determines whether the platform satisfies easy modification of its design and integration of additional components. OpenS determines whether the hardware (electronic circuits, design files, etc.) and software (source-codes, ROS packages, etc.) are fully open-source and maintained by the open-source community such that external hobbyists can replicate the same design without the need to contact the developer. Finally, the cost feature determines the affordability of the system. As shown in Table 2, the majority of the robots are not open-source and are expensive. This limits wide application, e.g., in research, navigation and logistics. Our *ROMR* has been developed as a low-cost open-source alternative. The approximate cost to redevelop it currently stands at less than $1,500.00. Recently, *ROMR* is been used for both B.Sc. and M.Sc. projects in our laboratory.

Table 2: Comparison of existing mobile platforms to our *ROMR* development.

Robot Name	**ROS**	**Payload** (kg)	**Custom**	**Cost** ($k\$$)	**OpenS**
RMP Lite 220	✓	50	✓	2.99	x
Ackerman Pro Smart	✓	22	✓	3.99	x
Nvidia Carter	✓	50	✓	10.00	✓
Panther UGV	✓	80	✓	15.90	x
Tiago base	✓	100	✓	11.50	x
Clearpath TurtleBot 4	✓	9	✓	1.90	✓
Summit XL	✓	65	✓	11.50	x
MIR100	✓	100	✓	24.00	x
AgileX Scout 2.0	✓	50	✓	12.96	x
Jackal J100	✓	20	✓	18.21	x
4WD eXplorer	✓	90	✓	15.70	x
ROSbot 2.0	✓	10	✓	2.34	x
ROMR	✓	90	✓	1.50	✓

2. Hardware description

ROMR is compactly and robustly designed (dimension - 0.46m $\times$ 0.34m $\times$ 0.43m - length x width x height) to ensure stability, ease of integration of additional components, and low cost of reproducing the system. For this purpose, we leveraged off-the-shelf (OTS) electronics that are commercially available online, additive manufacturing technologies (3D printing), and aluminium profiles for the system structural design. The main objective of using aluminium profiles is to use a lightweight structures that can carry the hardware and associated electronics of the robot, without increasing the overall weight of the platform. At the same time, the profiles resist load stress and damage during everyday use. The profile bars with slots were used also to enclose all the electronics and power subsystems at the base of the robot for proper weight distribution. This increases the flexibility to connect any additional hardware component to it.

The whole system is operated by an Arduino Mega Rev 3, a Nvidia Jetson Nano board and an Odrive 56V V3.6 brushless direct current (DC) motor controller. *ROMR* uses two inbuilt 350W hoverboard brushless motors with five hall sensors in a differential drive configuration. *ROMR* is powered by rechargeable lithium-ion batteries (36V 4400mAh), which are affordable, inexpensive to maintain, and eco-friendly (reduced carbon footprint).

Table 3: *ROMR* technical specifications with a total weight of 17.1kg, and a maximum payload of 90kg.

Parameters	**Technical specifications**
Robot dimensions	L * W * H = 0.46 * 0.34 * 0.43 (m)
Wheel dimensions	Drive wheels = 0.165m; Caster wheel = 0.075m
Robot Weight	17.1 kg
Max. Payload	90 kg
Max. speed	Up to 3.33m/s
Battery Capacity	36V 4400mAh
Motor Type	BLDC with 15 pole pairs (350W x 2)
Ground clearance	0.065 m
Operation environment	indoor/outdoor
Run time (full charge)	Approximately 8 hours

Table 4: Overview of the *ROMR* hardware and software libraries used.

Features	**Description**
Actuation	ODrive 56V V3.6 brushless DC motor controller, Nvidia Jetson Nano, and Arduino Mega Rev 3
Sensors & feedback	RPlidar A2, IMU, Depth cameras (Intel Realsense D435i & T265)
Operating system	Ubuntu 20.04 (ROS Neotic) or Ubuntu 18.04 (ROS Melodic)
Communication	ROS architecture (ROS C/C++ & ROS Python libraries), WiFi 802.11n, USB & Ethernet (for debugging)
Navigation & drive interfaces	ROS navigation stack, position & joint trajectory controller, joystick, rqt-plugin, ROS-Mobile (Android devices), hand gesture, web-based GUI
SLAM	Hector-SLAM, Cartographer, Gmapping, RTAB-Map
Simulation & visualisation	Gazebo, Rviz, MATLAB

The system is endowed with an Intel Realsense D435i RGB-D camera for deep learning tasks and Intel Realsense T265 for localization and tracking. *ROMR* is also equipped with a 9-axis MPU 9250 inertial measurement unit (IMU) sensor for tracking and localisation. The IMU sensor is used also for gesture-based teleoperation which allows a non-robotics expert to intuitively control the robot using hand gestures. For environment mapping, we used a RPlidar A2 M8 with a 360-degree field of view (FOV). This lidar has a maximum range distance of 16m, and operates at a frequency of 10Hz. The lidar sensor allowed us to simulate the 2D line-of-sight of the sensors specifically for building a 2D occupancy map of the environment, for visualization and for algorithm prototyping purposes.

The software interface is built on both Ubuntu 18.04 (ROS Melodic version) and Ubuntu 20.04 (ROS Neotic version). The robot weighs approximately 17.1 kg with a possible achievable speed of 3.33 m/s. The technical specifications are summarized in Table 3.

Furthermore in Table 4, we present the key features of the *ROMR* in line with other robots with comparable specifications. Initially, when envisioning the design, one of our core goals was to develop a low-cost scalable platform that robotic developers and the open-source community could easily adapt, to foster research in mobile navigation. For this reason, we tailored our design considerations based on this goal such that the *ROMR* should be:

- Modular - To offer the users the opportunity to easily integrate additional parts or units and reconfigure them to suit their needs.
- Portable and simple - To ensure that minimal and off-the-shelf hardware components could be used for its construction and replication. This would allow users not to worry about purchasing costly components and instead focus on the system's functional design.
- Low-cost and open-source - To ensure affordability and commercialisation of the system so that users can leverage the $ROMR$ framework in any form to develop novel and trivial robotic applications.
- Versatile and suitable - To ensure that it takes minimal time to be re-programmed for any useful purpose, whether navigation, logistics etc as well as adapt to new processes and changes in the environment.
- Unique - To enable hobbyists and users to learn new tools, techniques and methods useful to accelerate the development of intelligent systems.

3. Design files summary

The $ROMR$ design files are categorised into three different units, (a) the mechanical unit, (b) the software unit, and (c) the power, sensors, vision, communication, control and electronics units. Each of these units is briefly described in Tables 5 and 6. Table 5 describes the additive manufactured part (3D printed) and the 3D CAD models of the aluminium profiles and their accessories. The CAD files were used to generate the universal robot description format (URDF) [39] description of the $ROMR$ in order to simulate it using the ROS framework [29]. These parts were designed using the Solid Edge CAD tool. Table 6 lists all off-the-shelf (OTS) components. All design and construction files can be downloaded on our repository: https://doi.org/10.17605/OSF.IO/K83X7.

3.1 Mechanical unit

The mechanical unit includes the additive manufactured parts, the 3D CAD models of the aluminium profiles, and their accessories as summarised in Table 5. The labels P_n with $n = 1, 2, ...$, refers to the individual parts.

- $P1 - P4$, $P7 - P9$ and $P12$ are aluminium profiles used for the robot's chassis construction.
- $P5$ and $P6$ are used to cover the base of the robot, where the electronics components are placed.
- $P10$ is the full CAD assemble of the $ROMR$.
- $P11$ and $P13$ are used for material carriage and for mounting the RGB-D cameras respectively.
- $P14$ is 3D-printed to attach the RPlidar sensor.
- $P15$ is the $ROMR$ drive wheel from a consumer hoverboard scooter.

In subsection 3.2, the electronic components were discussed.

Table 5: Summary of the *ROMR* mechanical structure unit.

Des.	Description	File type	Open S. license	File location
P1	Alum. 40x40mm slot 8	.par (Solid Edge)	GNU GPL v3	https://osf.io/zhbd8
P2	Alum. 40x80mm slot 8	.par (Solid Edge)	GNU GPL v3	https://osf.io/z7qu5
P3	Corner bracket I-type	.par (Solid Edge)	GNU GPL v3	https://osf.io/dr4qe
P4	Mounting bracket I-type	.par (Solid Edge)	GNU GPL v3	https://osf.io/yqkea
P5	Bottom cover	.par (Solid Edge)	GNU GPL v3	https://osf.io/279zh
P6	Top cover	.par (Solid Edge)	GNU GPL v3	https://osf.io/yuf7c
P7	Swivel caster	.par (Solid Edge)	GNU GPL v3	https://osf.io/397cw
P8	Screw/bolt	.par (Solid Edge)	GNU GPL v3	https://osf.io/fhspk
P9	Corner bracket cover cap	.par (Solid Edge)	GNU GPL v3	https://osf.io/3xpj6
P10	*ROMR* full assemble	.asm (Solid Edge)	GNU GPL v3	https://osf.io/jybzf
P11	0.42x0.32x0.15m box	.par (Solid Edge)	GNU GPL v3	https://osf.io/k72zb
P12	Lock nut	.par (Solid Edge)	GNU GPL v3	https://osf.io/qdpwa
P13	Front hole plate	.par (Solid Edge)	GNU GPL v3	https://osf.io/38fbd
P14	RPLidar base holder	.stl (3D print)	GNU GPL v3	https://osf.io/envdh
P15	Drive wheel	.par (Solid Edge)	GNU GPL v3	https://osf.io/pbqcy

3.2 Power, sensors, vision, communication, control and electronics units

Table 6 shows a summary of the used electronics, sensors and control devices, as well as the power sources. $P16 - P27$ are off-the-shelf (OTS) components from different vendors. All vendors are listed in Table 8.

Table 6: OTS electronics, sensors and control devices.

Des.	Description	File type	Qty	File location
P16	Nvidia Jetson Nano B01 64GB	png	1	https://osf.io/72xtm
P17	Arduino Mega Rev 3	png	1	https://osf.io/d3qmj
P18	ODrive V3.6 56V	png	1	https://osf.io/jghnv
P19	IMU (MPU-9250 9DOF)	png	1	https://osf.io/k2h35
P20	Intel Realsense D435i camera	png	1	https://osf.io/xu368
P21	Intel Realsense T265 camera	png	1	https://osf.io/6cj5r
P22	nRF24L01+PA+LNA module	png	1	https://osf.io/43kd6
P23	Turnigy 2.4GHz 9X 8-Channel V2 transmitter & receiver	png	1	https://osf.io/3wu2x
P24	125mm & 225mm M2M, M2F, F2F GPIO wires	png	24	https://osf.io/kv982
P25	RPLidar A2 M8	png	1	https://osf.io/94sk2
P26	36V Lithium Ion battery 4400mAh	png	1	https://osf.io/umskh
P27	Power bank 2400mA	png	1	https://osf.io/z98gv

- $P16$ is the main brain of the robot. It handles the high-level control task required to run all the sensing, perception, planning and control modules.
- $P17$ is one of the most successful open-source platforms with relatively easy-to-use free libraries compared

to other open-source micro-computers. It is used for the low-level control, to send the control command to $P18$ which in turn drives and steers the $P15$. It also handles the communication between $P16$, $P18$ and any other compatible node.

- $P18$ is a high-performance, open-source brushless direct current (BLDC) motor controller from ODrive robotics [24]. It regulates all computations required to drive the two inbuilt hoverboards brushless DC motors with five hall-effect sensors. The sensors are used for the motor position feedback.
- $P19$ is a 9-axis inertial measurement unit (IMU) sensor specifically useful for tracking and localization tasks.
- $P20$ and $P21$ are 3D vision cameras for deep learning tasks like localization and tracking. Both cameras generate 3D image data of the task environment, process the data and then publish it to the appropriate topic in the ROS network.
- $P22$ and $P23$ are communication devices for wireless control of the $ROMR$.
- $P24$ are general purpose input-out (GPIO) connection wires.
- $P25$ is a 2D lidar with a 360-degree field of view (FOV), a maximum range distance of 16m, operating at a frequency of 10Hz. It is used for 2D occupancy grid mapping of the environment, as well as for collision detection and avoidance.
- $P26$ and $P27$ are the system's power sources for the motors, the sensors, and the control boards.

3.3 Software subsystem

The software subsystems include the Arduino sketches, ODrive calibration files, the ROS workspace containing the $ROMR$ universal robot description format (URDF) file for a Gazebo simulation, the Gazebo plugins files, the $ROMR$ meshes, the launch files, the joint and rviz configuration files, and RPlidar packages. The files are listed in Table 7, and are published using the open-source license GNU GPL V3, which allows the community to reproduce, modify, redistribute, and republish it.

Table 7: Software files.

Des. Name	**File Name**	**Description**	**Type**	**Open source license**	**File location**
S1	sketches.zip	Folder containing the Arduino sketches	Arduino sketches	GNU GPL v3	https://osf.io/279m4
S2	romr_robot.zip	Folder containing the $ROMR$ ROS files	ROS files	GNU GPL v3	https://osf.io/4qcn5

4. Bill of materials summary

Table 8 provides a summary of the bill of materials, which includes the lidar, the depth cameras, the Turnigy 2.4GHz 9X 8-Channel V2 transmitter & receiver, the 22nF capacitors, and the MPU-9250 sensor, that were sourced in the laboratory.

Table 8: Bill of materials used to build our *ROMR*.

Designator	**Qty**	**Unit cost (€)**	**Total cost (€)**	**Source of material**	**Material type**
Profile 40x40L I-type slot 8 (1.98m long)	1	26.49	26.49	www.motedis.at	Other
Profile 40x80L I-type slot 8(1.1m long)	1	25.64	25.64	www.motedis.at	Other
Swivel caster	1	4.51	4.51	www.motedis.at	Other
Hoverboard brushless DC motor wheels	1	49.00	49.00	www.voltes.nl	Other
Nvidia Jetson Nano B01 64GB	1	260.22	260.22	www.amazon.de	Other
Arduino Mega Rev 3	2	32.78	65.56	www.amazon.de	Other
ODrive V3.6 56V	1	249.00	249.00	www.odriverobotics.com	Other
IMU (MPU-9250)	1	15.92	15.92	www.distrelec.at	Other
nRF24L01+PA+LNA module	1	9.99	9.99	www.amazon.de	Other
Turnigy 2.4GHz 9X 8-Channel V2 transmitter & receiver	1	69.99	69.99	www.hobbyking.com	Other
125mm & 225mm M2M, M2F, F2F GPIO wires	1	3.99	3.99	www.amazon.de	Other
36V Lithium-Ion battery 4400mAh	1	33.63	33.63	https://de.aliexpress.com	Other
22nF capacitors	6	0.05	0.30	https://www.conrad.at	Other

5. Build instructions

Once the materials described in Table 5 and Table 6 are available, the subsequent task is to assemble them accordingly. To do that, the sequence of steps presented in this section should be followed. Some basic tools such as pliers, screwdrivers, a wire stripper, a 3D-printer, a saw, a soldering iron and others are required for building the hardware.

5.1 Hardware build instruction

The building of the *ROMR* chassis is done using the parts described in Table 5. To assemble the chassis, we considered the system compactness to maintain an acceptable mode of operation for a light and reliable system not compromising stability. The mechanical construction is done in several steps as follows:

1. Top base assembly: For this step, the parts required are $P1$, $P3$, and $P8$ as referenced in Table 5. First, cut out two each 0.38m, 0.26m and 0.24m lengths of $P1$. For each end of the piece, use a tap to cut thread and a clearance hole of a radius of 0.08m (M8). A description of how to use a tap can be found here https://www.wikihow.com/Use-a-Tap. Assemble the parts as illustrated in Figure 2, by following the direction of the arrow. The first block shows the components required, the middle block represents the exploded view with the interconnection of the components, and finally, the last block shows the outcome

of the assembly.

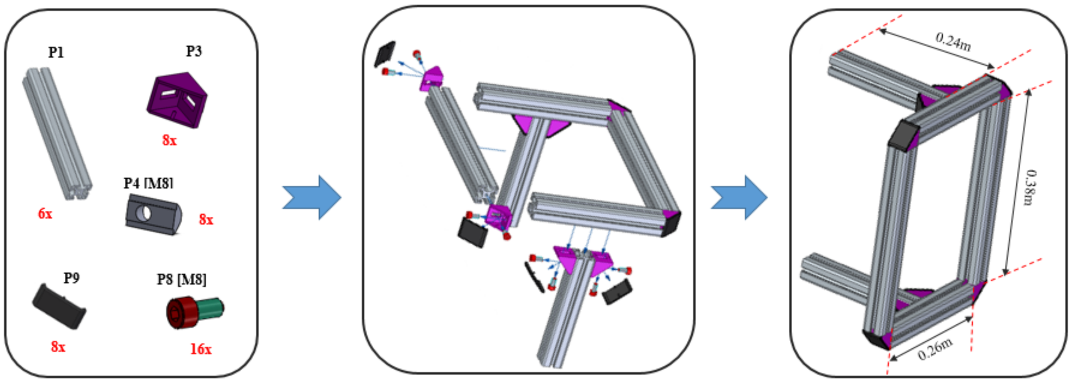


Figure 2: Top base chassis assembly.

2. Bottom and top chassis assembly: Parts required in this step are $P2$, $P3$, $P4$, $P8$, $P9$ and the result of step 1. Assemble the parts as illustrated in Figure 3.

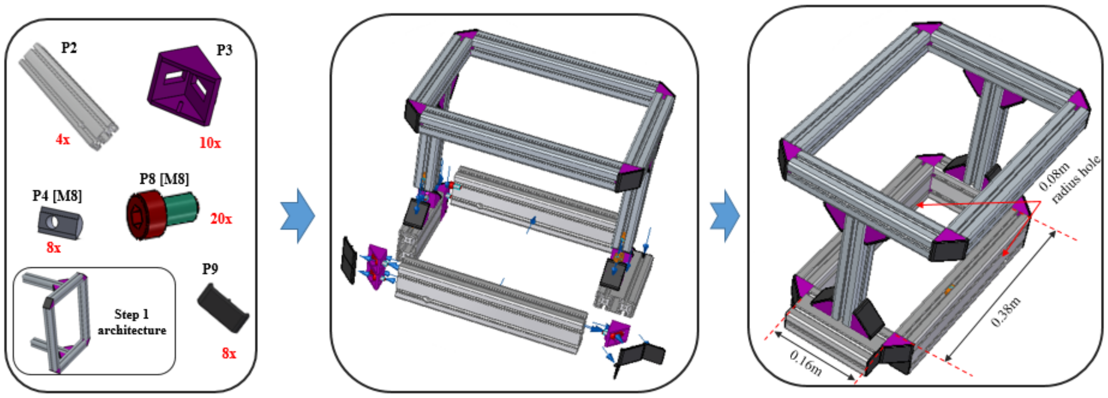


Figure 3: Assembling the top and bottom chassis.

3. Mounting of the caster and the drive wheels: Parts required are $P4$, $P7$, $P8$, $P15$ and the base from step 2. Note that $P8$ ($M8$) and $P8$ ($M6$) are required to mount $P15$ and $P7$ firmly to the base respectively. Assemble the parts as illustrated in Figure 4.

4. Mounting the bottom base plate: The base plate is a rectangular green plastic meant to hold and protect all the electronics subsystems. Parts required are $P5$, $P8$ ($M3$) and the resulting base from step 3. These parts are assembled as shown in Figure 5.

5. Mounting the internal electronics, the power subsystems to the bottom base plate and the RGB-D cameras: Parts required are $P13$, $P17$, $P18$, $P19$, $P20$, $P21$, $P22$, $P23$, $P26$, and the result from step 4. A couple of M3 screws (P8) are required for mounting the electronic components at the respective position. Figure 6 illustrates the procedure.

6. Mounting the top cover, external electronics, and final coupling: Parts required are $P1$, $P3$, $P4$, $P6$, $P8$ ($M3$), $P8$ ($M8$), $P9$, $P11$, $P16$, $P25$, and the resulting hardware from step 5. Figure 7 illustrates the procedure.

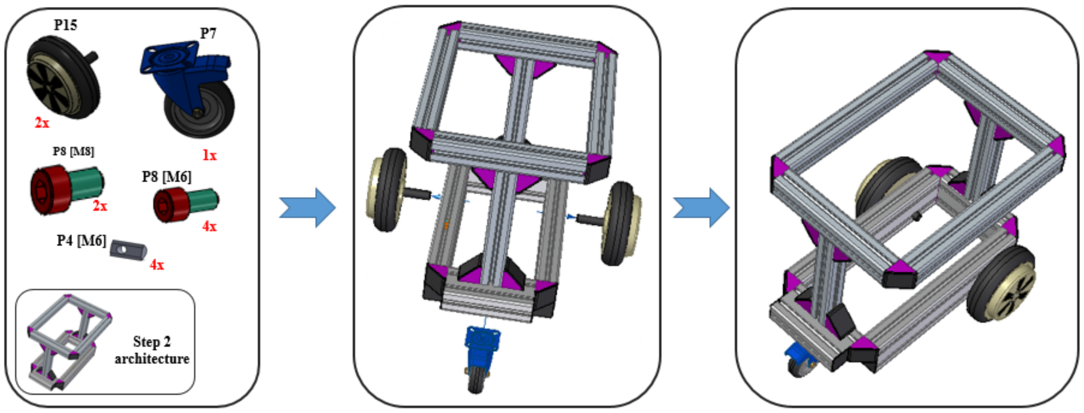


Figure 4: Mounting of the caster and the drive wheels to the base frame.

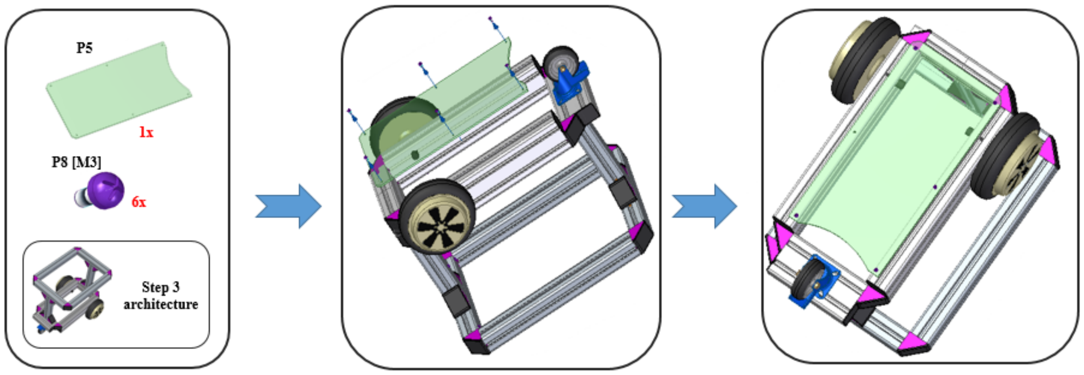


Figure 5: Mounting of the bottom plate to the base frame for attaching the electronics subsystems.

5.2 General connection and wiring instruction

The electronics, vision and sensor subsystem of *ROMR* are composed of several components interconnected by energy links and bidirectional or unidirectional information links as shown in Figure 8. Each link either sends or receives information from the interconnected components. The instruction below shows how the system wiring was done.

1. ODrive brushless DC motor controller (P18): The ODrive controller has to be wired to the motors as illustrated in Figure 9. Each of the three phases of the motors have to be connected to the motor outputs M0 and M1. The order in which the motor phases are connected is not important. The ODrive controller will figure it out during a calibration phase. However, after calibration, the order cannot be changed. If changed, consider re-calibrating the motors. Furthermore, a hoverboard motor is equipped with five hall sensors coloured red, black, blue, green and yellow for position feedback. Unfortunately, the ODrive controller has no noise filtering capacitors and consequently, the hall sensors are susceptible to noise. To get consistent and clean readings from the hall-effect sensors, noise filtering capacitors are required to be connected to the corresponding ODrive's J4 pinouts as described in Table 9.

   The required value of the filtering capacitor is approximately $22nF$. However, if you do not have exactly the required $22nF$, connecting two $47nF$ (c1 = c2 = 47nF) in series to obtain approximately 23.5nF will also work. Also, a 50W power resistor is required if the robot runs on battery. This prevents the ODrive from unexpected shutdown as a result of regenerated energy into the battery. Usually, the resistors come along with the ODrive controller during supply.

2. Arduino to ODrive interconnection: ODrive communicates with Arduino through a serial port or a universal asynchronous receiver/transmitter (UART). The UART pinouts are GPIO 1 (TX) and GPIO 2

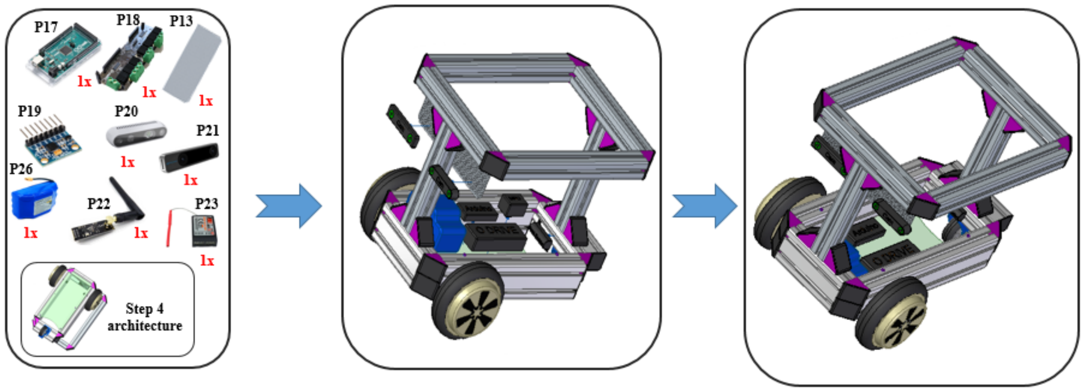


Figure 6: Mounting of the internal electronics and power subsystems to the bottom base plate.

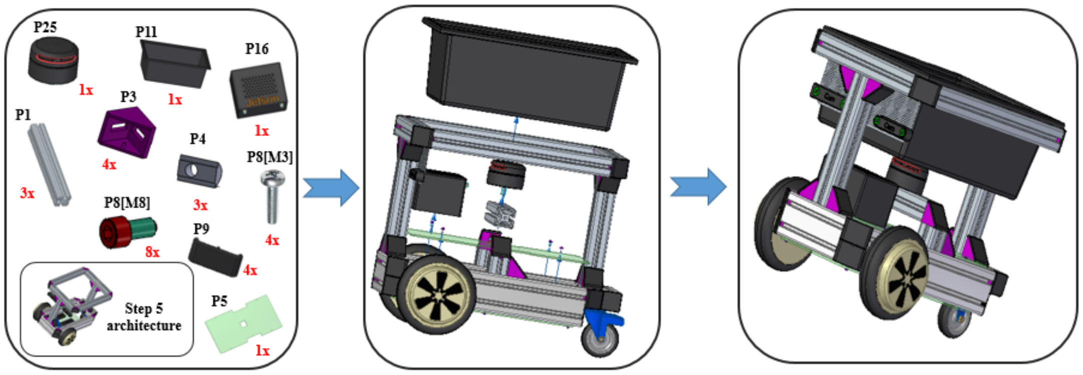


Figure 7: Mounting of the external electronics and the final coupling.

(RX) which should be connected to the Arduino Arduino's RX (pin 18) and TX (pin 17) respectively. Also, there is a need to connect the grounds (denoted by GND) of the Arduino and the ODrive controller boards.

3. Power distribution: It is recommended to use two separate power sources for the ODrive controller and Jetson Nano to avoid a ground loop. The 36V lithium-ion 4400mAh battery is wired directly to the ODrive motor controller, and an additional power-bank battery ($P27$) was used to power the Jetson which requires only 5V 2500mA.

4. The nRF24L01+ module, MPU-9250 and Arduino connections: From Figure 14, the control unit consists of $P17$, $P19$, and $P22$. These parts are required for the wireless transmission of data. The MPU-9250 and the Arduino are connected with four cables, the ground (GND), the power supply (VCC) and two cables for the I2C communication (SDA, SCL). The SDA pin of the MPU-9250 is connected to the SDA (pin 20) of the Arduino, and the SCL pin to the SCL (pin 21) of the Arduino. A power supply between 2.4V and 3.6V is needed. As a consequence, the 3.3V power supply pin of the Arduino is recommended to be used [3]. The communication between the nRF24L01+ module and the Arduino is established via an SPI interface. In Figure 10, a detailed description of the nRF24L01+ pinout can be seen.

   The SPI pins of the Arduino (Mega 2560 Rev3) are MISO $\rightarrow$ pin 50; MOSI $\rightarrow$ pin 51, and SCK $\rightarrow$ pin 52. In this work, the pins are connected as follows: nRF2401 GND $\rightarrow$ Arduino GND; nRF2401 VCC $\rightarrow$ Arduino 3.3V; nRF2401 CE $\rightarrow$ Arduino digital 7; nRF2401 CSN $\rightarrow$ Arduino digital 8; nRF2401 MOSI $\rightarrow$ Arduino digital 51; nRF2401 SCK $\rightarrow$ Arduino pin 52; nRF2401 MISO $\rightarrow$ digital 50. Furthermore, at the robot unit (Figure 14), the nRF24L01+ module and the Arduino are wired in the same way as the control unit.

5. RC receiver (P23) wiring to Arduino: RC receivers are needed to drive the motor using pulse-width modulated (PWM) signals [22]. A typical RC receiver such as the one used in this work (P23) has

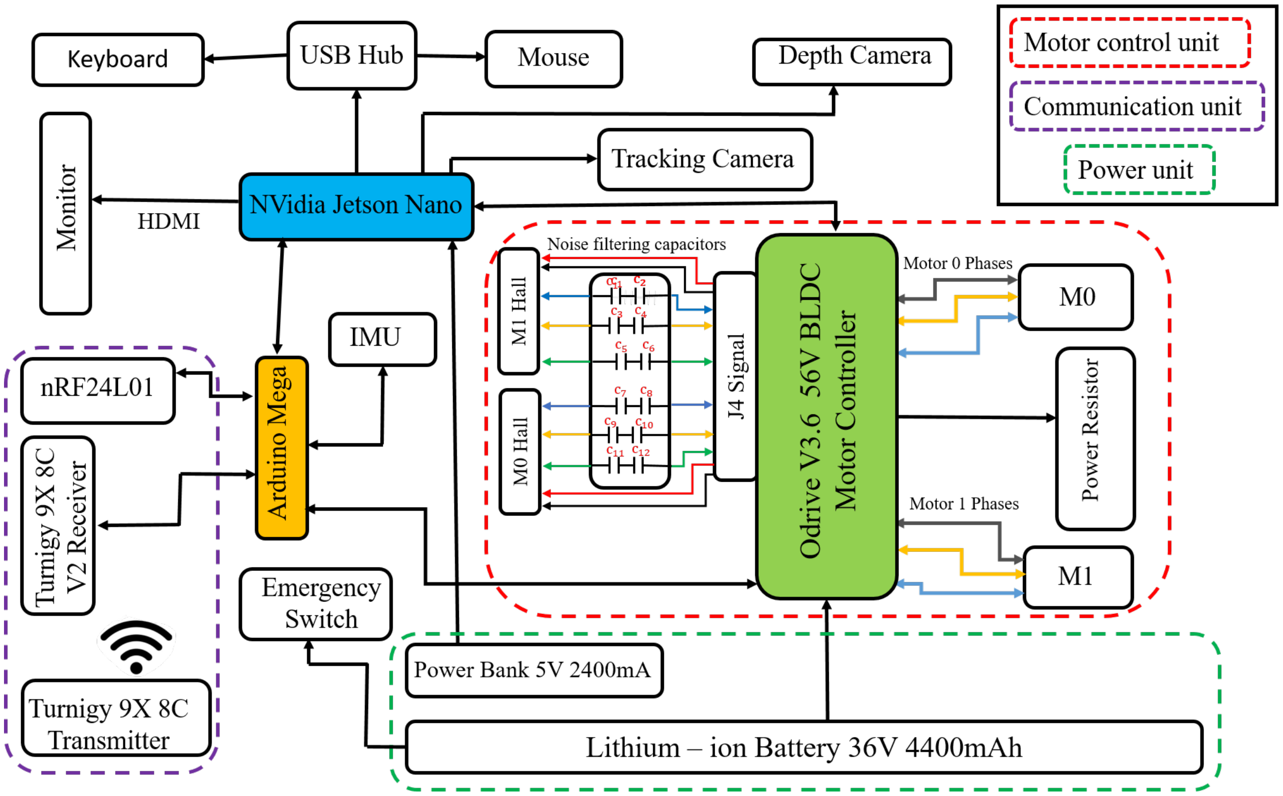


Figure 8: The *ROMR* hardware architecture is based on an ODrive board (light green) to actuate the motors, a Nvidia Jetson Nano (light blue) as computing interface for high level control, an Arduino Mega Rev3 (orange) as low level computing interface.

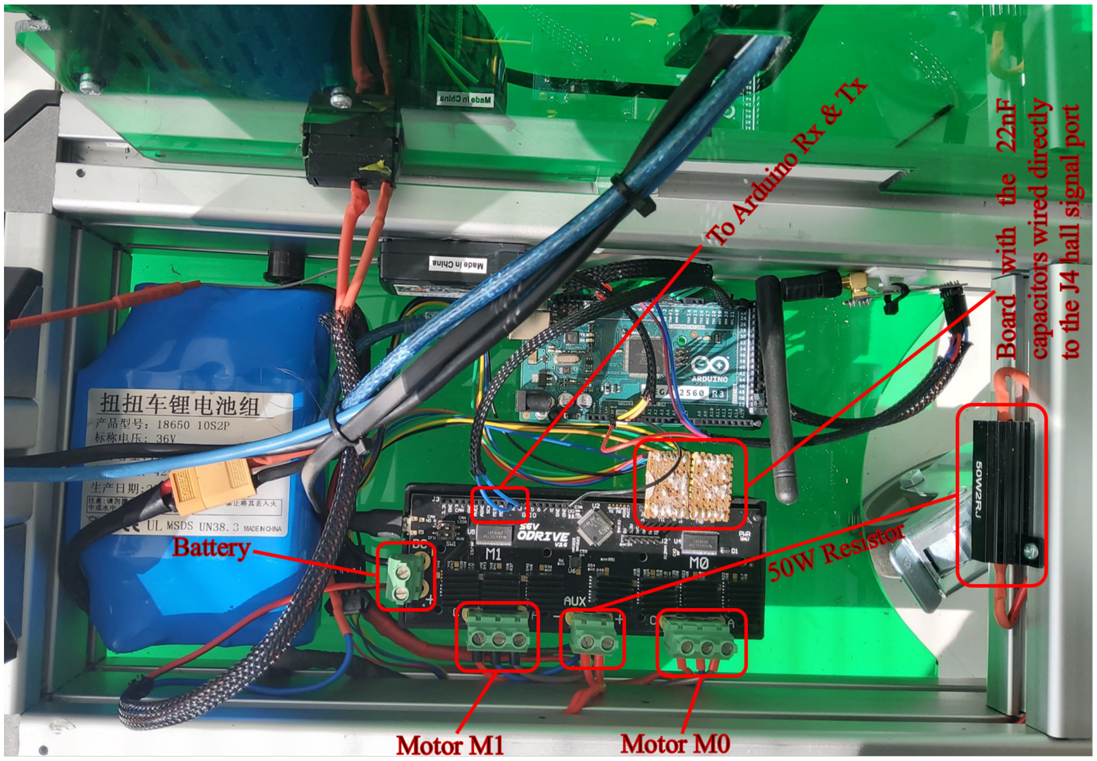

To Arduino Rx & Tx
Board with the 22nF capacitors wired directly to the J4 hall signal port
Battery
50W Resistor
Motor M1
Motor M0

Figure 9: Internal wiring of the ODrive controller. On the bottom, the three phases of the motors (M0 and M1) are connected. On the top, 22nF capacitors are used as noise filter for the hall sensors per motor

Table 9: Hall sensor wiring at the J4 signal port of the Odrive.

Hall wire	**J4 signal port**
Red	5V
Yellow	A
Blue	B
Green	Z
Black	GND

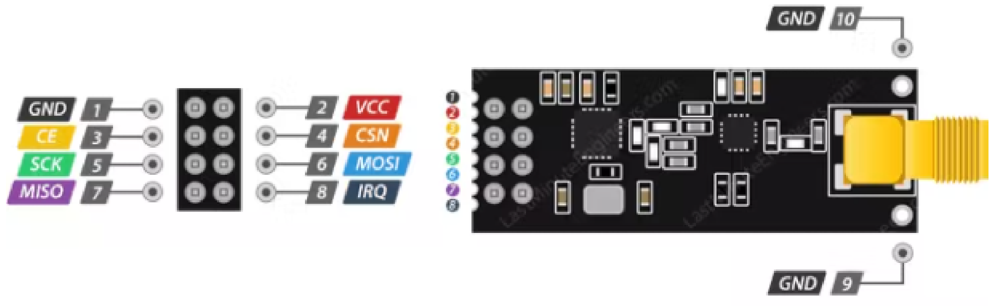


Figure 10: Illustrated is nRF24L01 module with its pinout [20]. The module is used for the wireless communication between the Arduino at the remote control unit and the Arduino at the robot unit (see Figure 14).

three kinds of pins, two of which are GND and 5V respectively, and the remaining are for PWM signals. Hoverboard motors require three signal pins, i.e., throttle, steering and enable [17]. Therefore, the receiver to the Arduino wiring is described as follows: RC_GND → Arduino_GND, RC_5V → Arduino_5V, RC_CH1 → Arduino_port2, RC_CH2 → Arduino_port3, and RC_CH3 → Arduino_port18.

6. Operation instructions

To get started operating the *ROMR* in real-time for the first time, the first step is to power ON the robot by pressing the ON-OFF switch beside the Jetson nano board ($P16$) and start controlling it with the transmitter. Note that after powering ON the robot, a red LED blinks. You have to wait until the blinking stops, then it is ready to be used. However, if you have changed the default configuration, or probably wish to rebuild and reconfigure the robot from scratch to operate it in other modes, then this section provides step-by-step instruction to get started. Before these steps, ensure that all the robot parts are properly assembled and wired according to the instructions in Section 5.

6.1 Initial configuration and setup instruction

This section provides details about the initial configuration of the *ROMR*. It is recommended to follow the instruction in this section very careful as it determines how well the system would perform.

6.1.1 Nvidia Jetson Nano set up and ROS installation

Nvidia Jetson Nano acts as the host mini-computer for the *ROMR* enabling one to run intelligent applications as well as the provided ROS packages. To set it up, some basic tools such as a microSD card (32GB minimum recommended), a USB keyboard and a mouse, a computer display (HDMI or DP) and a micro-USB power supply are required. The microSD card and micro-USB power supply usually come with the Jetson Nano during supply, if you purchased the full development kit. The setup instructions are as follows:

1. Download the Jetson Nano developer SD card image (JetPack), and write the image to the microSD card that usually come along with the Jetson Nano board. The reference instruction to configure the JetPack for the first time can be found at https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit.
2. Install ROS and its packages on the JetPack. The instruction for the installation is provided at the official ROS wiki page at http://wiki.ros.org/melodic/Installation/Ubuntu for ROS Melodic, which is supported per default by the JetPack. Or follow the instructions here: https://github.com/Qengineering/Jetson-Nano-Ubuntu-20-image to configure Ubuntu 20.04 OS image for ROS Noetic installation. Thereafter, install ROS Noetic from http://wiki.ros.org/noetic/Installation/Ubuntu.
3. Create a workspace. A workspace is a set of directories with which you can store the ROS code that you may have written. Instructions can be found at http://wiki.ros.org/catkin/Tutorials/create_a_workspace.

6.1.2 Setup the ODrive tool and calibrate the BLDC motors

To begin the initial configuration and calibration of the hoverboard BLDC motors, ensure that all the necessary wiring has been completed as described in subsection 5.2. Make sure that all relevant switches are turned ON, and the motors are positioned in such a way that they can freely move. The ODrive need to be connected to the host computer (the Nvidia Jetson Nano). ODrive has a python3 programming interface, called "odrivetool" for configuring the BLDC motors and commanding them to move at a specific number of revolutions per minute or rotations.

Therefore, python3 is required to be installed first on the host computer before setting up the "odrivetool". The instruction for the setup can be found at https://docs.odriverobotics.com. Alternatively, you could simply download and run the calibration script which has been prepared to avoid the tedious task of following the tutorial to calibrate the motors at https://osf.io/awf9t.

The script will configure the axes of the motors and their respective encoders as well as set motor parameters such as the velocity gain, the position gain, the bandwidth, and more. After the successful calibration, you can test the motors from the "odrivetool" command-line to ensure that it is properly configured and ready to receive velocity commands. First, start the "odrivetool" and from the command-line send the following commands to the motors:

```
# Place the motor connected to axis0 (M0) of the ODrive board at velocity control mode.
    odrv0.axis0.controller.config.control_mode =  CONTROL_MODE_VELOCITY_CONTROL
# Place the the axis0 (M0) motor at closed loop control mode.
    odrv0.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
# At this point, the axis0 motor should spin at 1 turn/s.
    odrv0.axis0.controller.input_vel = 1
# Stop the motor on axis0 from spinning (set the velocity to 0 turn/s).
    odrv0.axis0.controller.input_vel = 0
# Disable the motor on axis0 and return it to an idle state.
    odrv0.axis0.requested_state = AXIS_STATE_IDLE
```

Repeat the same test with the second motor that is connected to axis 1 of the ODrive board. If the configuration and calibration of the motors are correct, then both motors should spin until they receives 0 as commanded velocities. Or receives idle state commands. At any point during the calibration and testing with the interactive "odrivetool", always use the code below to list calibration errors and to clear them.

```
dump_errorrs(odrv0,True) # dumps ODrive calibration errors and clear them
```

6.1.3 Install Arduino integrated development environment (IDE) and connect to ROS

The Arduino IDE allows one to write software programs (sketches) and upload them to the Arduino board for robot control. The steps to set up the IDE, and connect it to ROS are described below:

1. First, download the Arduino IDE at https://www.arduino.cc/en/Guide and follow the onscreen instructions to set it up.
2. Integrate the Arduino to communicate with the ROS via rosserial node. The rosserial_arduino package enables the Arduino to communicate with the Jetson Nano via a USB-A male to USB-B male cable. The setup instruction can be found at the http://wiki.ros.org/rosserial_arduino.
3. Launch the ROS serial server by running the code below at the command line to ensure that the setup was successful.

   ```
       rosrun rosserial_python serial_node.py  /dev/ttyACM0  # rosserial python node
   ```

   For a detailed explanation of the above rosserial-python node, visit the ROS wiki address at http://wiki.ros.org/rosserial_python#serial_node.py. Ensure that you check the port to which your Arduino is connected and the baud rate of your device. In our case, it is ttyACM0 and 115200 respectively. In your case, it may be different. Take note of it always.
4. Connect the Arduino to the ODrive. First, install the ODriveArduino library. Clone or download the repository https://github.com/odriverobotics/ODrive/tree/master/Arduino. From the Arduino IDE, select Sketch → Include Library → Add .ZIP Library and select the enclosed zip folder. Run the *"ODriveArduinoTest.ino"* sketch with the motors connected to ensure that it is properly configured and ready to accept commands. If everything went successfully, then the motors should move accordingly.

6.1.4 The MPU-9250 and nRF24L01+ modules setup

In Figure 14, the function of the control unit is based on the Arduino sketch that reads the MPU-9250 data and forwards it to the robot via the nRF24L01+ module. To run the sketch, the "FaBo 202 9Axis MPU9250" library by Akira Sasaki released under the Apache license, version 2.0 must be installed, as well as the "rf24" library by TMRh20 Avamander released under the GNU general public license. The FaBo 202 9Axis MPU9250 library is used for reading the data measured by the MPU-9250 sensor, while the rf24 library is needed for the usage of the nRF24L01+ module. Both libraries can be found in the library manager of the Arduino IDE.

In the robot unit, there are multiple components which have to be set up beforehand. These components are the Jetson Nano, ROS, rosserial_arduino package, the Arduino, the imuOrientToCmdVelTranslater sketch and the ODrive 56V V3.6 board. These setups are explained in the previous section.

The Arduino at the robot unit primarily forwards the raw IMU data that it receives from the nRF24L01+ module to the Jetson Nano. This is done by publishing the data to the "imu/data_raw" topic of the ROS system, which is running on the Jetson Nano. The Jetson Nano converts the raw IMU data into velocity commands that include a target linear velocity and a target rotation speed of the robot. The Arduino receives these velocity commands by subscribing to the "cmd_vel" topic. After the Arduino receives the velocity commands, it converts them into target speed values of the motors that are set on the ODrive board.

6.1.5 Setup the RPlidar

The RPlidar sensor provides the scan data required for mapping, localization and navigation purposes. It is connected to the Jetson Nano or the host computer through a USB serial port. The procedure for its setup is summarized in the following steps.

1. Clone the RPlidar ROS packages https://github.com/Slamtec/rplidar_ros to your ROS catkin workspace source directory and run the code below to build the rplidarNode and rplidarNodeClient.

   ```
   catkin_make
   ```

2. Check the authority of the rplidar serial port by typing at the command window:

   ```
   ls -l /dev |grep ttyUSB
   ```

   Take note of the port in which the USB is connected eg .../ttyUSB0. Add authority to write the USB:

   ```
   sudo chmod 666 /dev/ttyUSB0
   ```

3. Launch the RPlidar node to view and test if the setup was successful.

   ```
   roslaunch rplidar_ros view_rplidar.launch
   ```

   If correctly set up, you will obtain an output similar to the one displayed in Figure 16b with the lidar scan represented as red dots.

After the above setups, the *ROMR* is ready for experimentation.

6.2 Experimentation & remote operation instruction

To allow a human operator to intuitively control the *ROMR*, we developed three remote control techniques. In this section, we provide step-by-step instructions on how these techniques can be implemented.

6.2.1 *ROMR* teleoperation from remote-control (RC) devices

As per default, *ROMR* is configured to operate in RC mode. However, if the default configuration has been altered or changed, then, the following steps must be taken to reconfigure it. Before these steps, make ensure that the ODrive controller has already been calibrated and configured to accept commands (see sub-subsection 6.1.2 for instructions). Also, it is important to ensure that the RC receiver is properly wired according to the instructions in subsection 5.2.

1. Upload the *"romr_remote_control.ino"* sketch to Arduino. Before that, the "Metro" library has to be included in the Arduino IDE library. The "Metro" library can be downloaded from https://github.com/thomasfredericks/Metro-Arduino-Wiring.
2. With the motors switched off, move the RC transmitter sticks and monitor it from Arduino serial plotter. If there is communication between the receiver and the transmitter, you would obtain a similar response as the one shown in Figure 11 from the Arduino serial plotter.

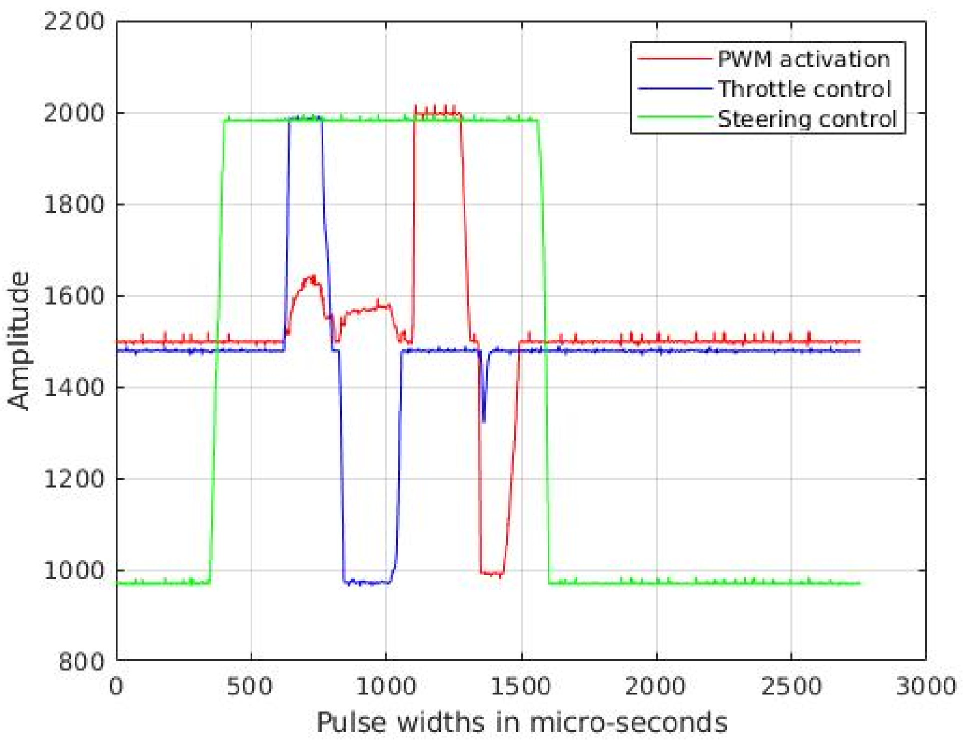


Figure 11: Setting up the RC control on the turnigy 9x

6.2.2 *ROMR* control from Android-based device

One of the main features of the *ROMR* is the ability to be teleoperated from any Android-based device. The idea is to alleviate the need for complex robot teleoperation devices such as a joystick, a RC transmitter, etc., and to provide an intuitive way of controlling the robot by simply touching the Android device screen. To achieve this, we leveraged the framework developed by Rottmann Nils et al. [31]. The setup is straightforward. First, you have to ensure that Arduino has been set up to communicate with ROS. If you have not done that yet, it is advisable to follow the instructions in Section 6. "rosserial" is very important for this section. Therefore, make sure that the "rosserial" python node (rosserial_python serial_node.py) is running properly. The whole communication structure is described in Figure 12a.

As illustrated in Figure 12, the "rosserial" python node allows all the compatible connected electronics to communicate directly with the robot using the ROS topics and messages. All the information between the interconnected systems is communicated with the help of the rosserial package. Ensure that the robot is switched ON, the battery is connected, and the wheels are free to spin. If you have changed the default ODrive calibration, ensure that the calibration is completed before continuing. Open the downloaded ROS-Mobile App, which enables ROS to control the robot's joint velocities. The App supports linear (forward and backward movement) and angular (rotation around the z-axis) movements. See Figure 12b for the setup. A SSH connection has to be established between the devices, and all the devices have to be on the same wireless network. The steps are summarised as follows:

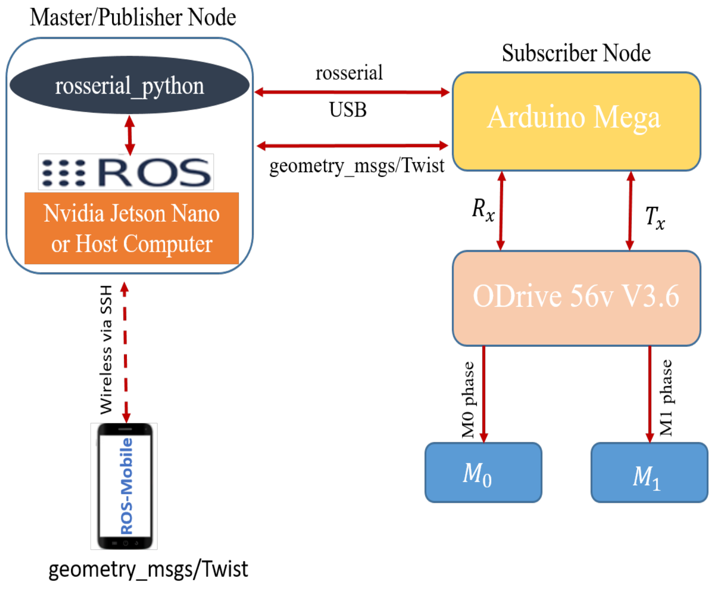
1. Download the ROS-Mobile App from the Google Playstore.
2. Configure the IP address. First, connect the robot and the Android device to the same wireless network. From the command window terminal, type "ifconfig", this will display the IP address, e.g., 192. 168.1.15.
3. At the "master" node URI of the ROS-Mobile App, enter the IP address and 11311 for the "master" port. Ensure that *roscore* is running, and click on the connect button.

4. Once the above steps are completed, upload the *"ros_mobile_control.ino"* sketch to Arduino. While the *roscore* is still running, run the rosserial python node:

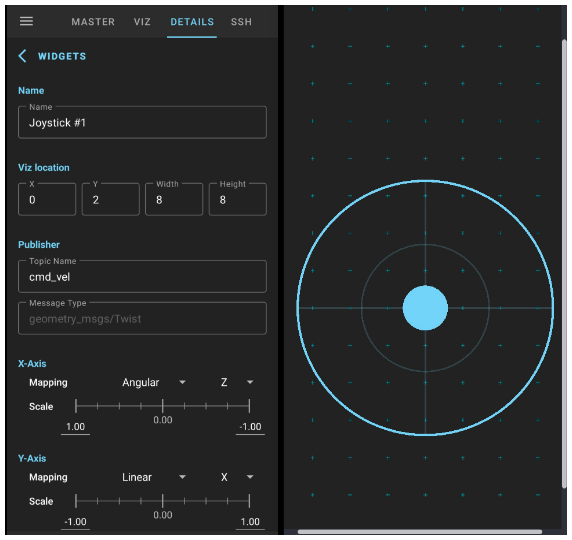
```
    rosrun rosserial_python serial_node.py /dev/ttyACM0
```

   in a separate terminal. Take note of the .../dev/ttyACM0 Arduino port. It may be different in your case.

5. At the "details" tab of the ROS-Mobile App, select "Add widget", select "joystick" and set the xyz-coordinates accordingly. Click on the "viz" tab to visualise and control the robot. Ensure that the rosserial python node is running and the cmd_vel topic is been subscribed to. Once done, the robot can be controlled by simply touching on the respective coordinates on the screen.



(a) System communication structure.



Name
Name: Joystick #1

Viz location
X: 0 | Y: 2 | Width: 8 | Height: 8

Publisher
Topic Name: cmd_vel
Message Type: geometry_msgs/Twist

X-Axis
Mapping: Angular | Z
Scale: 1.00 | 0.00 | -1.00

Y-Axis
Mapping: Linear | X
Scale: -1.00 | 0.00 | 1.00

(b) Control and monitoring from ROS-Mobile [31] or Android-Based devices.

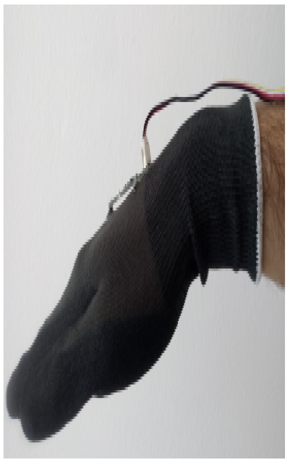
Figure 12: $ROMR$ real-time control and monitoring with ROS-Mobile device.

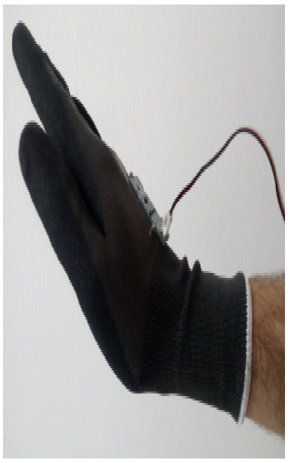
6.2.3 Gesture-based control of the $ROMR$

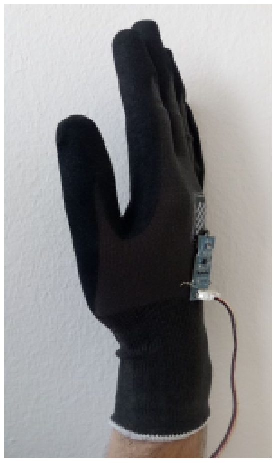
Unlike the traditional or "ready-to-use" robot control approaches such as a joystick or the ROS rqt plugin, a gesture-based approach has the potential to control the robot in a very intuitive way. This strategy allows the operator to focus on the robot instead of the controller. The goal is for a non-robotics expert to be able to remotely navigate the robot depending on the direction in which the operator's hand is tilted. For example, if the hand is tilted forward (pitch angle), the robot should move forward. If the hand is tilted to the side (roll angle), the robot should rotate. Since the MPU-9250 sensor does not measure the orientation of the operator's hand, but only the acceleration and the rotations speed of the sensor, the IMU sensor ($P19$) data must be fused to determine the orientation of the operator's hand and the tilt angle of the sensor.

We implemented four gestures with different hand motions (see Figure 13). We used the IMU sensor ($P19$) to measure the hand gestures needed and map them onto the robot's linear and angular velocities. The data collected by the IMU sensor is sent via a wireless connection to the robot, which uses it to calculate the movement commands for the motors. We leveraged the framework proposed in [9] and [21] to achieve the gesture-based control strategy.

A detailed overview of the hardware and software architecture and the data flow is shown in Figure 14. The components are divided into a control unit and a robot unit, as the components of these two units are physically


(a) Hand tilt forward → the robot moves forward.


(b) Hand tilt backwards → the robot moves backward.


(c) Hand tilt to the right → the robot moves to the right.

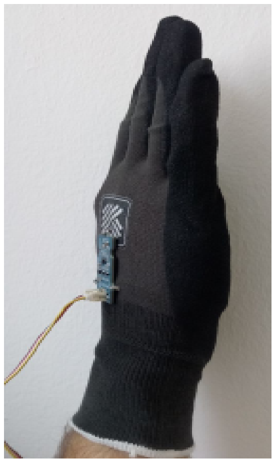

(d) Hand tilt to the left → the robot moves to the left.

Figure 13: *ROMR* real-time control and monitoring based on hand movement.

located in different places. While the control unit is been attached to the user's arm/hand, the robot unit is within the *ROMR* control board.

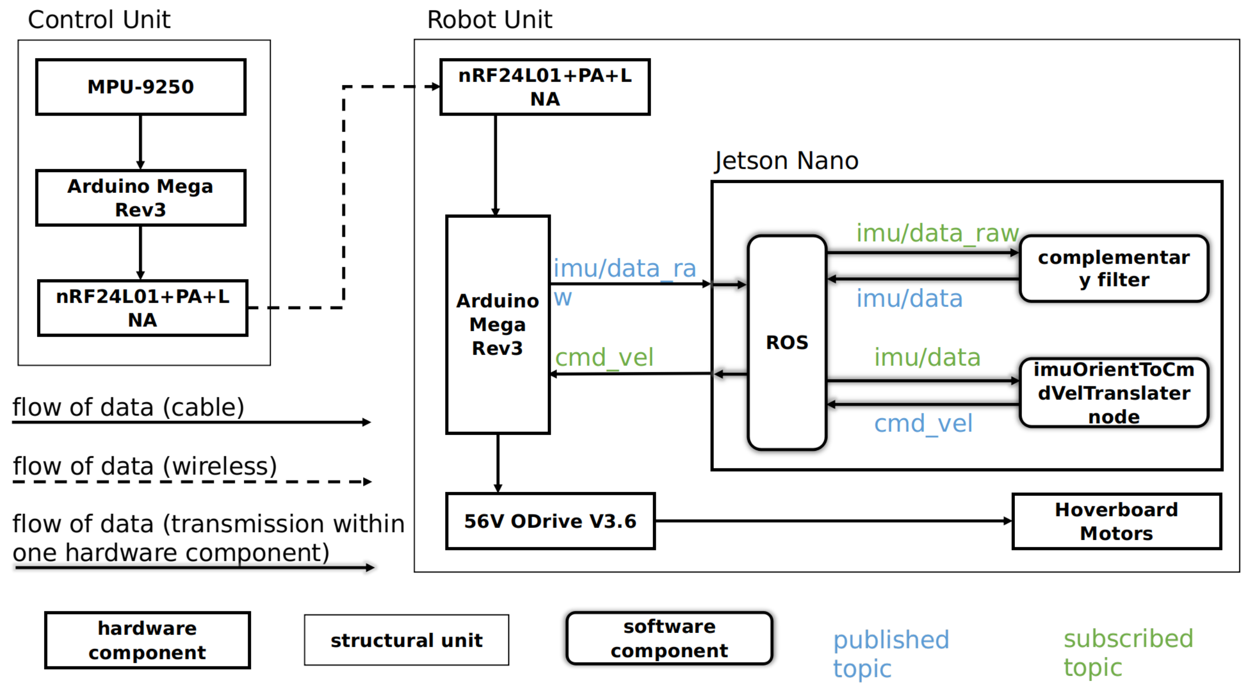


Figure 14: Block diagram illustrating the architecture and flow of information during the gesture control approach.

The step-by-step instruction to operate the robot based on gesture demonstration is described as follows:

1. At the control unit, upload the *"IMUDataNRF24L01_Transmitter.ino"* sketch to Arduino, and at the robot unit, upload *"NRF24L01Receiver_PC_WheelController_WheelMonitoring.ino"* sketch to the Arduino.
2. From the host computer, run *roscore* to start the ROS master, and then:

```
roslaunch romr_robot romr_bringup.launch # Or
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

to start rosserial python node and establish a connection between the Arduino and the Jetson Nano. Depending on which port the Arduino is connected to the Jetson Nano, the port ttyACM0 must be adjusted accordingly.

3. Finally, execute the command:

```
rosrun romr_robot imuOrientToCmdVelTranslater
```

to start the complementary filter node. By switching the robot ON, the motors are automatically powered and ready to be controlled with the IMU sensor. Depending on the direction the IMU sensor is tilted, a corresponding movement of the robot is obtained as shown in Figure 13a-d. During the operation, attention must be paid so that the hand is not tilted about 90°.

7. Validation and characterization

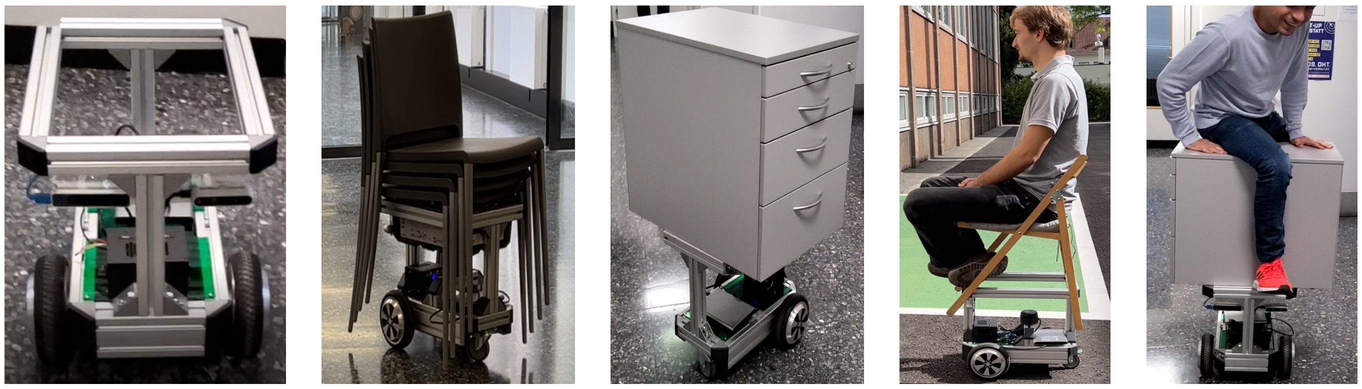
ROMR has been successfully developed, tested and validated both in simulation and in real-world scenarios. Experiments were performed to characterise its performance, robustness and suitability for research, navigation and logistics applications. The evaluation results are presented in this section. Furthermore, the validation video can be viewed at https://osf.io/v8tq2.

7.1 General subsystem validation test

This section provides information about the robustness of the system by completing different tests with the robot and its sub-components. The outcome of each test is presented in Table 10.

Also, in Figure 15, the result of the payload test carried out as described in Table 10 is presented.

The robot is teleoperated to move along linear and angular trajectories with different loads ranging from the robot weight only (17.1kg) to 90kg. The goal is to verify how much load the robot can carry without affecting the controller. Although the *ROMR* can carry a load up to a maximum of 90kg, it is, however, not recommended to operate it at maximum load continuously to extend its life span. All the various load tests were carried out on the robot while moving on a flat surface. Thus, we did not evaluate the performance on irregular or unstructured surfaces.



(a) *ROMR* weight only. (b) 16kg load. (c) 25kg load. (d) 85kg load. (e) 90kg load.

Figure 15: Validation of the maximum load capacity of the *ROMR*. Shown from a-e are some of the load test result

Table 10: General subsystems validation test.

Test name	**Test purpose**	**Test Process**	**Expected Result**	**Outcome**
Chassis test	To Verify the solidity and stability of the robot's chassis	Complete five different movement tasks at high speed with all the components mounted	All the subsystems must be stable and rigid throughout the test	Passed
Payload test	To Verify the maximum load capacity the robot can carry without affecting the controller	Place different loads on the robot and check the response	The robot should be able to convey the load up to the maximum capacity	Carried up to 90kg (see Figure 15)
Reconfiguration test	To Verify how long and easy to dismantle and re-assemble the system	Dismantle all the subsystems including the chassis and wiring and re-assemble them. Record the time taken to complete the process	Should not take more than 3 hours	Took about 1 hour 15 minutes
Battery live test	To evaluate how long the battery can power the shuttle for a long mission task	Operate the robot continuously with all the electronics parts active for a long period of time	The battery should last up to the maximum capacity	The battery lasted for about 8 hours

7.2 Simulation scenarios

Although, the development of *ROMR* focused on real-world applications, it is also important to have a 3D simulation model of the robot, to enable users to work in virtual environments to explore tools, techniques and methods. Furthermore, the simulation model enables users to become familiar with 3D simulation and visualisation tools such as Gazebo [10] and Rviz [38].

The simulation platform includes three parts: the environment model, the robot model, and the sensors model. For the environmental model, we created the floor plan of our laboratory environment using the Gazebo model editor (see Figure 16a).

Taking advantage of the ROS framework [29], the *ROMR* model was implemented in accordance with the unified robot description format (URDF) [39]. The URDF is an extended mark-up language (XML) format that describes all kinematic and dynamic properties of the robot, the physical elements, such as the links, the joints, the actuators, and the sensors [33]. We generated the URDF of the robot including the sensors using the 3D CAD models described in Table 5. Finally, all the models were verified with several simulation tests as depicted in Figure 16. The step-by-step procedure for this simulation are as follows:

1. Download the *ROMR* ROS files at https://osf.io/4qcn5 to your catkin workspace and build it.
2. Open three terminal windows, and execute the following in each of the terminal:

```
    roscore
```

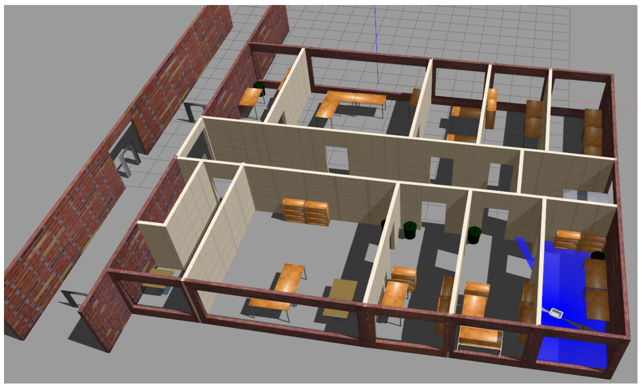
```
roslaunch romr_robot romr_house.launch
```

to launch the *ROMR* world (operational environment model in Gazebo [10]) with the robot spawned and the sensor model active, see Figure 16a, and at the third terminal run

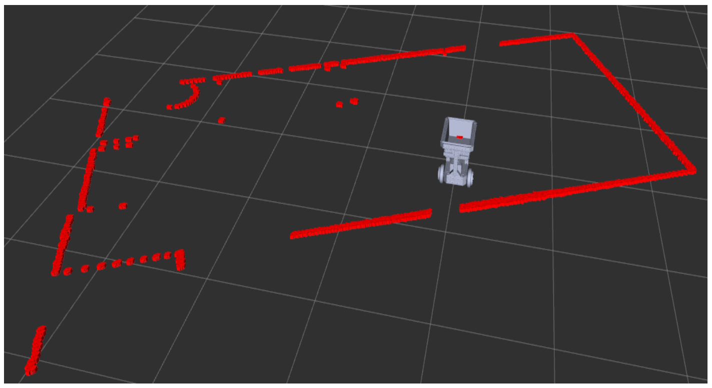
```
roslaunch romr_robot romr_rviz.launch
```

to visualise in Rviz, see Figure 16b.

3. Navigate the robot within the operational environment using ROS rqt plugin, keyboard or the framework described in sub-subsection 6.2.2.



(a) The operational environment model with the robot model together with the sensor model. The blue lines are the lidar sensor scan of the environment in Gazebo.



(b) Rviz visualisation. The red dotted lines are the lidar scan showing the location of obstacles (or objects) within the robot environment.

Figure 16: *ROMR* simulation platform.

7.3 Application of the *ROMR* for SLAM

As stated earlier, *ROMR* provides a framework for evaluating and developing SLAM algorithms. The SLAM problem is usually to build a map of an unknown environment, i.e, mapping while simultaneously keeping track of the estimates of a robot's pose (the position x, y and the orientation). Given a series of control and sensor observations [5], [32], the map is built and the pose is estimated. We evaluated the Hector-SLAM algorithm [18] on our RPlidar sensor ($P25$) to build the map of our real laboratory environment. The adaptive Monte Carlo localization (AMCL) [6] approach was used to localise the robot within the built map. The advantage of the Hector-SLAM technique over other 2D SLAM techniques such as Gmapping [12], Google Cartographer [14] is that it only requires laser scan data and does not need odometry data to build the map. To generate the map, the following steps have to be followed:

1. Set up the RPlidar as described in sub-subsection 6.1.5.
2. Download or clone the Hector-SLAM packages at https://github.com/tu-darmstadt-ros-pkg/hector_slam.git to your ROS workspace, and set the coordinate frame parameters according to the instruction at the ROS wiki page http://wiki.ros.org/hector_slam. Build the ROS workspace including the Hector-SLAM and RPlidar packages (*catkin_make*), then proceed to the next step.
3. Open four terminal windows, run the following in each of the terminal windows:

```
roscore # starts the roscore node
```

```
roslaunch rplidar_ros rplidar.launch # launches the rplidar node
roslaunch hector_slam_launch tutorial.launch # launches the hector-slam algorithm
```

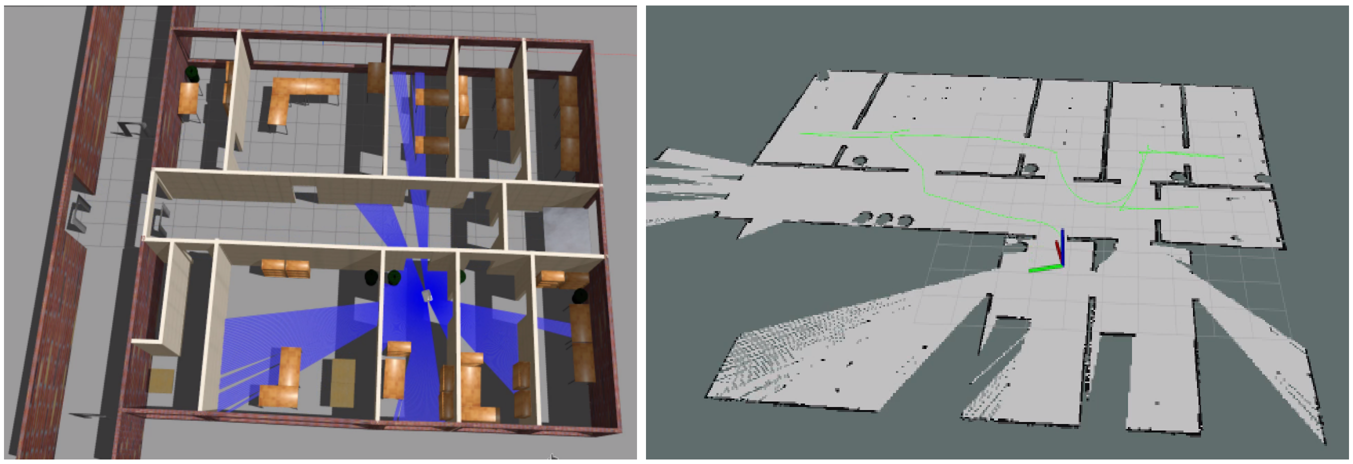
While all the nodes are running, navigate the robot around the environment by employing any of the control approaches implemented in sub-subsections 6.2.1, 6.2.2, and 6.2.3. While building the map, it is recommended to move at a low speed such that a quality map is created.

4. After the mapping is completed, execute the following at the fourth terminal:

```
rosrun map_server map_saver -f laboratory_map # saves the built map
```

Take note of the location where the map is saved. It would be required for localisation and autonomous navigation. The saved map can be viewed on your screen by running:

```
rosrun map_server map_server laboratory_map.yaml # view the saved map
```



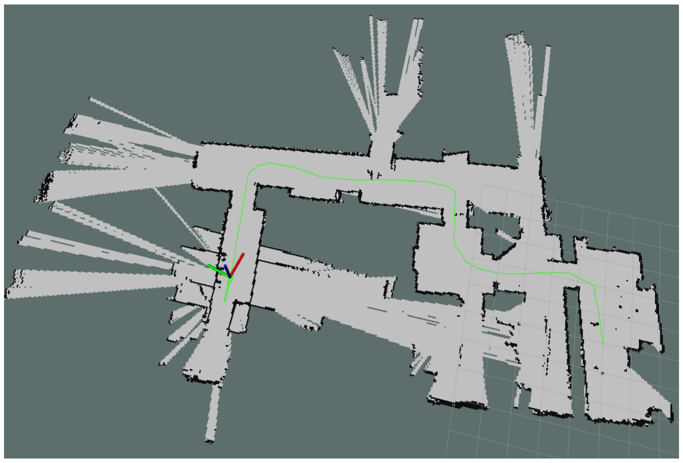
(a) Virtual laboratory world in Gazebo. (b) Occupancy grid map of the environment in Rviz.

Figure 17: Generation of the 2D occupancy grid map of the environment using the 360° lidar sensor ($P25$) and a "Hector-SLAM" algorithm [18]. (a) The blue lines represent the lidar scan. (b) The pale grey areas indicate the unoccupied (free) spaces that the robot can navigate, the black lines represent occupied areas not transversal by the robot, and the green line represents the robot's trajectory.

Figure 17 shows the map built in the virtual laboratory environment with the *ROMR*. Figure 18a shows the map built in the real-laboratory world. For the localisation of the robot, the employed AMCL approach uses a particle filter to track the pose of the robot [30]. It maintains a probability distribution over a set of all the possible robot poses [6], and updates this distribution using the data from the *ROMR* odometry and rplidar scan (P25). Figure 18b shows the localisation of the *ROMR* within the 2D occupancy grid map, where the dark green clusters denotes the AMCL particles representing the estimates of the location of the robot.

7.4 Proposed maintenance for the *ROMR*

Finally, to increase the life span of *ROMR*, predictive and corrective maintenance is necessary. This aims to maintain or repair the robot to ensure that the robot works at its maximum efficiency. Predictive maintenance such as listening to any abnormal noise; performing a visual inspection of all the parts of the robot; checking the energy storage level; checking for vibrations, mechanical defects, improper connections, calibration errors, etc., are proposed before each operation. This is intended to reduce the probability of failure during operation. Furthermore, curative maintenance on the other hand should be performed after detecting a failure.


(a) Generated the 2D occupancy grid map of the operational environment with the $P25$ lidar and the Hector-SLAM algorithm. The robot's trajectory is represented with the green line.

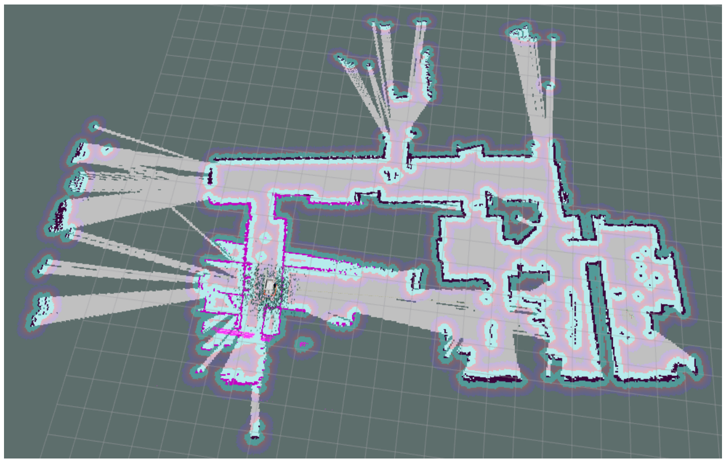

(b) Localising the $ROMR$ within the map with the Adaptive Monte Carlo Localisation (AMCL) algorithm. The dark green clusters are the AMCL particles that represents the estimates of the location of the robot.

Figure 18: Applying $ROMR$ to a real-world SLAM problem.

8. Conclusion

In this paper, we presented an ros-based open-source mobile robot ($ROMR$) for research and industrial applications. We provided the detailed information about the hardware design, the architecture, the operation instructions, and the advantages it offers compared to the commercial platforms. The entire design utilises off-the-shelf electronic components, additive manufacturing technologies and aluminium profiles that are commercially available to speed-up the re-prototyping of the framework for custom or general-purpose applications. We implemented several control techniques that can enable a non-robotics expert to operate the robot easily and intuitively. Furthermore, we demonstrated the applicability of the $ROMR$ for logistics problems through implementing navigation, simultaneous localisation and mapping (SLAM) algorithms, which are a fundamental prerequisite for autonomous robots. The experimental validation of the robustness and performance is illustrated in the video https://osf.io/v8tq2. Future work will focus on porting the whole platform to ROS 2. As an open-source platform, the scientific community has been granted permission to use all the design files published at https://doi.org/10.17605/OSF.IO/K83X7. This open-source strategy supports a rapid progress in the development of intelligent mobile robots and dexterous systems.

Declaration of interest

None

CRediT Author Statement

Nwankwo Linus: Conceptualization, construction, software simulation, experimentation, and writing (original draft preparation).
Fritze Clemens: Software simulation, experimentation, and reviewing
Konrad Bartsch: Construction and CAD design
Elmar Rueckert: Supervision, validation, reviewing and editing

Acknowledgements

This project has received funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) No #430054590 (TRAIN).

References

[1] Robert D. Atkinson. Robotics and the future of production and work. 2019. url: https://d1bcsfjk95uj19.cloudfront.net/sites/default/files/2019-robotics-future-production.pdf.

[2] D. Betancur-Vásquez, M. Mejia-Herrera, and J.S. Botero-Valencia. Open source and open hardware mobile robot for developing applications in education and research. *HardwareX*, 10:e00217, 2021. url: https://www.sciencedirect.com/science/article/pii/S2468067221000468.

[3] Giuseppe Coviello and G. Avitabile. Multiple synchronized inertial measurement unit sensor boards platform for activity monitoring. *IEEE Sensors Journal*, PP:1–1, 03 2020. url: https://www.researchgate.net/publication/340110168_Multiple_Synchronized_Inertial_Measurement_Unit_Sensor_Boards_Platform_for_Activity_Monitoring.

[4] Rafal Cupek, Marek Drewniak, Marcin Fojcik, Erik Kyrkjebø, Jerry Chun-Wei Lin, Dariusz Mrozek, Knut Øvsthus, and Adam Ziebinski. Autonomous guided vehicles for smart industries – the state-of-the-art and research challenges. In Valeria V. Krzhizhanovskaya, Gábor Závodszky, Michael H. Lees, Jack J. Dongarra, Peter M. A. Sloot, Sérgio Brissos, and João Teixeira, editors, *Computational Science – ICCS 2020*, pages 330–343, Cham, 2020. Springer International Publishing. url: https://link.springer.com/chapter/10.1007/978-3-030-50426-7_25.

[5] J.-A Fernández-Madrigal and Jose Luis Blanco. *Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods.* 01 2012. url: http://ingmec.ual.es/~jlblanco/papers/IGI_SLAM_book_appendices.pdf.

[6] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. pages 343–349, 01 1999. url: http://robots.stanford.edu/papers/fox.aaai99.pdf.

[7] Giuseppe Fragapane, René de Koster, Fabio Sgarbossa, and Jan Ola Strandhagen. Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda. *European Journal of Operational Research*, 294(2):405–426, 2021. url: https://www.sciencedirect.com/science/article/pii/S0377221721000217.

[8] Giuseppe Ismael Fragapane, Dmitry A. Ivanov, Mirco Peron, Fabio Sgarbossa, and Jan Ola Strandhagen. Increasing flexibility and productivity in industry 4.0 production networks with autonomous mobile robots and smart intralogistics. *Annals of Operations Research*, 308:125–143, 2022. url: https://link.springer.com/content/pdf/10.1007/s10479-020-03526-7.pdf.

[9] Guanhao Fu, Ehsan Azimi, and Peter Kazanzides. Mobile teleoperation: Feasibility of wireless wearable sensing of the operator's arm motion. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, January 2022. url: https://doi.org/10.1109/IROS51168.2021.9636838.

[10] gazebosim.org. Simulate before you build. Available online, url: https://gazebosim.org/home.

[11] Felix Grimminger, Thomas Flayols, Jonathan Fiene, Alexander Badri-Spröwitz, Ludovic Righetti, Avadesh Meduri, Majid Khadiv, Julian Viereck, Manuel Wuthrich, Maximilien Naveau, Vincent Berenz, Steve Heim, and Felix Widmaier. An open torque-controlled modular robot architecture for legged locomotion research. *IEEE Robotics and Automation Letters*, PP:1–1, 02 2020. url: https://arxiv.org/abs/1910.00093.

[12] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007. url: https://ieeexplore.ieee.org/document/4084563.

[13] P. A. Hancock, Illah Nourbakhsh, and Jack Stewart. On the future of transportation in an era of automated and autonomous vehicles. *Proceedings of the National Academy of Sciences*, 116(16):7684–7691, 2019. url: https://www.pnas.org/doi/abs/10.1073/pnas.1805770115.

[14] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016. url: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45466.pdf.

[15] Nirmal Baran Hui and Dilip Kumar Pratihar. *Design and Development of Intelligent Autonomous Robots*, pages 29–56. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. url: https://doi.org/10.1007/978-3-642-11676-6_3.

[16] Wonse Jo, Jaeeun Kim, Ruiqi Wang, Jeremy Pan, Revanth Krishna Senthilkumaran, and Byung-Cheol Min. Smartmbot: A ros2-based low-cost and open-source mobile robot platform. *arXiv preprint arXiv:2203.08903*, 2022. url: https://doi.org/10.48550/arXiv.2203.08903.

[17] Lukas Kaul. Hoverboard motors turned into an rc skater. Available online, June 10, 2019, url: https://blog.arduino.cc/2019/06/10/hoverboard-motors-turned-into-an-rc-skater/.

[18] Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, Karen Petersen, Uwe Klingauf, and Oskar Von Stryk. Hector open source modules for autonomous mapping and navigation with rescue robots. pages 624–631, 01 2014. url: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.435.9600&rep=rep1&type=pdf.

[19] Sascha Kolski. *Mobile Robots: Perception & Navigation*. IntechOpen, London, United Kingdom, Feb 2007. url: https://doi.org/10.5772/36.

[20] lastminuteengineers.com. How nrf24l01+ wireless module works & interface with arduino. url: https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/.

[21] Shuang Li, Jiaxi Jiang, Philipp Ruppel, Hongzhuo Liang, Xiaojian Ma, N. Hendrich, Fuchun Sun, and Jianwei Zhang. A mobile robot hand-arm teleoperation system by vision and imu. 03 2020. url: https://doi.org/10.48550/arXiv.2003.05212.

[22] Dave Madison. How to use an rc controller with an arduino. Available online, August 27, 2020 at url: https://www.partsnotincluded.com/how-to-use-an-rc-controller-with-an-arduino/.

[23] Alexandra Markis, Maximilian Papa, David Kaselautzke, Michael Rathmair, Vinzenz Sattinger, and Mathias Brandstötter. Safety of mobile robot systems in industrial applications. 05 2019. url: https://www.researchgate.net/publication/337339431_Safety_of_Mobile_Robot_Systems_in_Industrial_Applications.

[24] odriverobotics.com. Getting started. Available online at url: https://docs.odriverobotics.com/v/0.5.4/getting-started.html.

[25] Luigi Pagliarini and Henrik Lund. The future of robotics technology. *Journal of Robotics, Networking and Artificial Life*, 3:270, 02 2017. url: https://www.researchgate.net/publication/315986147_The_future_of_Robotics_Technology.

[26] Jordi Palacín, Elena Rubies, and E. Clotet. The assistant personal robot project: From the apr-01 to the apr-02 mobile robot prototypes. *Designs*, 6:66, 07 2022. url: https://www.mdpi.com/2411-9660/6/4/66.

[27] Vijay M. Pawar, James Law, and Carsten Maple. Manufacturing robotics: The next robotic industrial revolution. 2016. url: https://www.ukras.org.uk/wp-content/uploads/2021/01/UKRASWP_ManufacturingRobotics2016_online.pdf.

[28] R. Prinz, R. Bulbul, J. Scholz, M. Eder, and G. Steinbauer-Wagner. Off-road navigation maps for robotic platforms using convolutional neural networks. *AGILE: GIScience Series*, 3:55, 2022. url: https://agile-giss.copernicus.org/articles/3/55/2022/.

[29] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. volume 3, 01 2009. url: http://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf.

[30] Ioannis Rekleitis. A particle filter tutorial for mobile robot localization. 01 2004. url: https://www.cim.mcgill.ca/~yiannis/particletutorial.pdf.

[31] Nils Rottmann, Nico Studt, Floris Ernst, and Elmar Rueckert. Ros-mobile: An android application for the robot operating system. *arXiv preprint arXiv:2011.02781*, 2020.

[32] Elmar Rueckert. Simultaneous localisation and mapping for mobile robots with recent sensor technologies. Master's thesis, Technical University Graz, 2010. url: https://cps.unileoben.ac.at/wp/MScThesis2009Rueckert.pdf,ArticleFile.

[33] Filippo Sanfilippo, Øyvind Stavdahl, and Pål Liljebäck. Snakesim: A ros-based rapid-prototyping framework for perception-driven obstacle-aided locomotion of snake robots. In *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1226–1231, 2017.

[34] Raivo Sell, Anton Rassõlkin, Ruxin Wang, and Tauno Otto. Integration of autonomous vehicles and industry 4.0. *Proceedings of the Estonian Academy of Sciences*, 68:389–394, 12 2019. url: http://vana.kirj.ee/public/proceedings_pdf/2019/issue_4/proc-2019-4-389-394.pdf.

[35] Jahanzaib Shabbir and Tarique Anwer. A survey of deep learning techniques for mobile robot applications. *arXiv preprint arXiv:1803.07608*, 2018. url: https://arxiv.org/abs/1803.07608.

[36] Sebastian Thrun. Probabilistic algorithms in robotics. *AI Magazine*, 21, 07 2000.

[37] Hendrik Unger, Tobias Markert, and Egon Müller. Evaluation of use cases of autonomous mobile robots in factory environments. *Procedia Manufacturing*, 17:254–261, 2018. 28th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2018), June 11-14, 2018, Columbus, OH, USAGlobal Integration of Intelligent Manufacturing and Smart Industry for Good of Humanity.

[38] wiki.ros.org. 3d visualization tool for ros. Available online August 16, 2022, url: http://wiki.ros.org/rviz.

[39] wiki.ros.org. Learning urdf step by step. Available online at url: http://wiki.ros.org/urdf/Tutorials.

[40] Junpei Zhong, Chaofan Ling, Angelo Cangelosi, Ahmad Lotfi, and Xiaofeng Liu. On the gap between domestic robotic applications and computational intelligence. *Electronics*, 10:793, 03 2021. url: https://www.mdpi.com/2079-9292/10/7/793.