การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 48
The 48th Electrical Engineering Conference (EECON-48)
วันที่ 19-21 พฤศจิกายน 2568 ณ โรงแรมฟูราม่า จังหวัดเชียงใหม่

A Sliding Mode Control Approach to Hybrid Manual-Automatic Velocity Regulation for a Raspberry Pi-Based Remote Vehicle

Vorrachit Cheakpimai[1], Nattavit Piamvilai[*1], Suriyotai Supanyapong[1], Nattaphat Thanavittayapaisan[1], and Polatip Thongpet[1]

[1*]Department of Electrical Engineering Technology, College of Industrial Technology,
King Mongkut's University of Technology North Bangkok,
E-mail: nattavit.p@cit.kmutnb.ac.th[*]

Abstract

This paper presents the design and implementation of a remote-controlled vehicle system that combines manual input with autonomous control using a Sliding Mode Control (SMC) algorithm executed on a Raspberry Pi 5 platform. The proposed system integrates throttle, steering, and gear inputs from the user with real-time closed-loop velocity regulation. The control architecture employs encoder-based feedback and layered communication between a Raspberry Pi 5 and Raspberry Pi Pico W, enabling low-latency PWM signal generation and stable actuation. To evaluate system performance, experiments were conducted on a custom-built chassis dynamometer under both no-load and mechanical full load conditions across multiple gear levels. The results demonstrate the SMC algorithm's ability to maintain consistent tracking accuracy, reject disturbances, and bound duty cycle variations within a narrow range. These findings validate the feasibility of applying robust control strategies on low-cost embedded platforms for hybrid manual–automatic vehicular systems.

Keywords: Sliding Mode Control, Raspberry Pi 5, Remote Vehicle, Embedded Control, PWM, Velocity Regulation, Encoder Feedback, Raspberry Pi Pico W

1. Introduction

Reliable control in vehicular systems under uncertain mechanical loads and external disturbances remains a core challenge, particularly in semi-manual settings where users operate the throttle, steering, and gear shifting. Maintaining consistent performance under such hybrid input conditions demands real-time, robust control strategies. a promising approach is SMC, known for its robustness against model uncertainties and external perturbations. By driving system states toward a sliding surface and maintaining them there, SMC achieves fast convergence and strong disturbance rejection. As control platforms integrate manual input with automation, SMC ensures accurate and stable closed-loop control—even with active user interaction.

This paper presents the design and implementation of an SMC-based control system applied to a four-wheel model vehicle, incorporating a discrete gear-shifting mechanism. While users manually operate input devices, the SMC algorithm autonomously adjusts motor speed through real-time PWM duty cycle modulation to maintain target velocity.

System validation was performed using a custom-built chassis dynamometer capable of simulating realistic road conditions. Experiments were conducted at three discrete gear levels—Gear 1 (15 km/h), Gear 2 (30 km/h), and Gear 3 (40 km/h)—under both no-load and load scenarios. Key performance metrics such as motor current and PWM duty cycle were recorded. Results confirm that the proposed control scheme effectively maintains stability and tracking performance under hybrid manual–automatic operation.

2. Methodology

2.1 Overview of the Remote-Controlled Vehicle System with Sliding Mode Control

Figure 1 provides an overview of the system architecture, illustrating the communication and control flow among key modules, including the user interface, central server, Raspberry Pi 5, Raspberry Pi Pico W, encoder, and actuators. It shows how user input is captured and propagated through each layer to enable real-time actuation via closed-loop feedback, ensuring responsiveness and accuracy.

The vehicle operates under a semi-manual framework, where users control the steering wheel, throttle, and gear level. At the same time, the embedded controller computes control actions based on sensor feedback and real-time processing. Each module plays a specific role, the user interface captures input, the server manages communication, the Pi 5 performs computation, the Pico W executes actuation, and the encoder supplies velocity feedback; all modules are interconnected via network and serial communication for synchronized operation.

The process begins with a USB-connected steering wheel on a local computer, which transmits x, y positions and gear selections to the server over a dedicated socket protocol [3]. The server, using a static IP, forwards this data to the Pi 5 via a lightweight socket interface. The Pi 5 calculates speed from encoder pulses and communicates with the Pico W over UART. It then runs the SMC algorithm to compute speed control, steering angle, and PWM duty cycle, which are sent to the Pico W for final actuation. The Pico W sends PWM signals to the DC motor and angle commands to the servo, completing the control loop. This modular setup ensures reliable performance and supports flexible testing of control strategies

*Corresponding Author

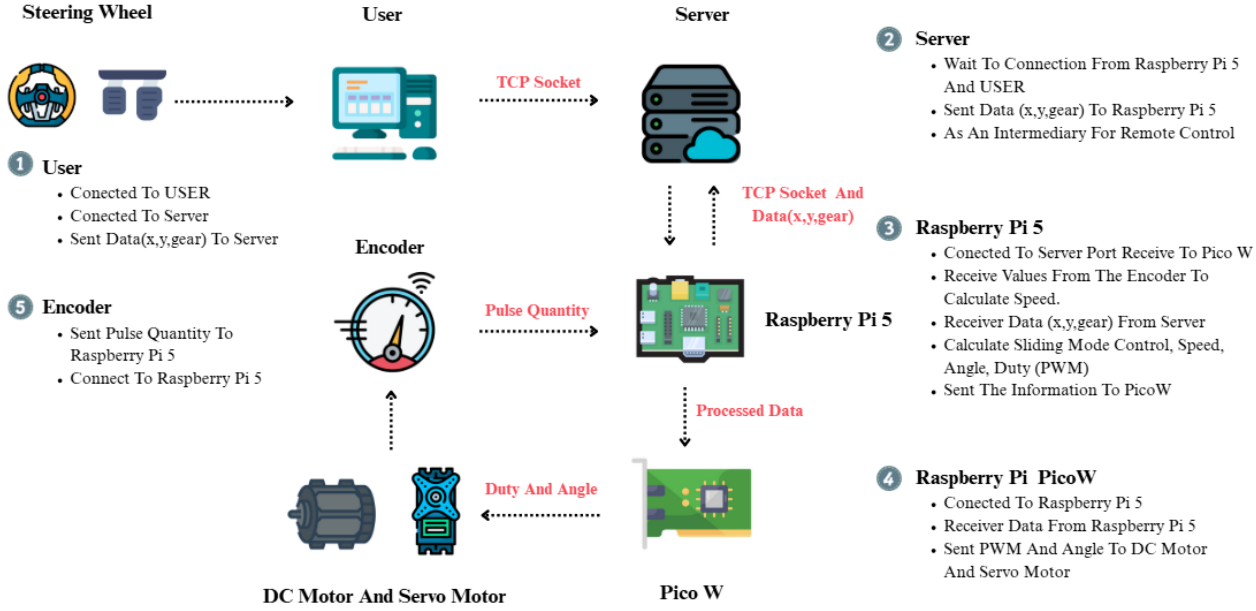

Fig. 1. Overview of the proposed remote-controlled vehicle system architecture, illustrating the data flow from the user's steering wheel, through a central server, to the Raspberry Pi 5 and Pico W for SMC-based motor control.

2.2 Sliding Mode Control (SMC) Methodology

SMC is a nonlinear control strategy that offers robustness against model uncertainties and external disturbances [1]. It is particularly effective for regulating dynamic systems such as mobile vehicles. This study implements SMC on a Raspberry Pi 5 to control vehicle speed under varying load conditions [4-5], using encoder feedback to obtain real-time velocity measurements. The SMC framework consists of three phases sliding surface definition, reaching phase, and sliding phase. The primary control objective is to drive the system state such that the sliding surface s converges to zero and remains there, i.e., $s(t) = 0$, indicating that the system has reached the desired dynamic behavior. As shown in Eq. (1).

$$s(t) = e(t) + \lambda \dot{e}(t) \tag{1}$$

where

$s(t)$ = sliding surface at time t

$e(t)$ = tracking error at time t

$\dot{e}(t)$ = derivative of the tracking error

λ = A positive constant that defines the slope of the sliding surface

During the reaching phase, the control input is designed to drive the system state toward the sliding surface by applying a discontinuous control law, as shown in Eq. (2).

This phase is critical to ensure that the system enters the desired sliding mode in finite time, regardless of initial conditions or external disturbances. The magnitude of the control gain plays a key role in determining how quickly the trajectory converges to the sliding surface, and improper tuning may lead to excessive control effort or slow convergence.

$$u_{smc} = -K \times sgn(s(t)) \tag{2}$$

where

u_{smc} = The corrective PWM signal from the SMC controller

K = positive gain that dictates the intensity of the correction

$sgn(s(t))$ = is the signum function, which returns +1, −1, or 0 based on the sign of $s(t)$

Once the system reaches the sliding surface, it enters the sliding phase, where system dynamics are constrained to follow the linear behavior of the surface. This phase provides SMC with robustness, enabling effective rejection of disturbances and model uncertainties.

However, the discontinuous control law may cause chattering—high-frequency switching that can degrade actuator performance [2], [6-7]. While this study employs the basic signum function, future work may introduce a boundary layer or saturation function to reduce chattering. The resulting control signal is then used to compute a PWM duty cycle transmitted to the motor driver, enabling real-time closed-loop speed control in the vehicle.

2.3 Execution Flowchart of the SMC-Based Control System

Figure 2 shows the flowchart of the SMC-based control system implemented on Raspberry Pi 5 and Raspberry Pi Pico W. The process starts with input data from a central server, including steering wheel positions x, y and gear selection. At the same time, encoder signals from a brushless DC motor are captured to compute the vehicle's real-time speed.

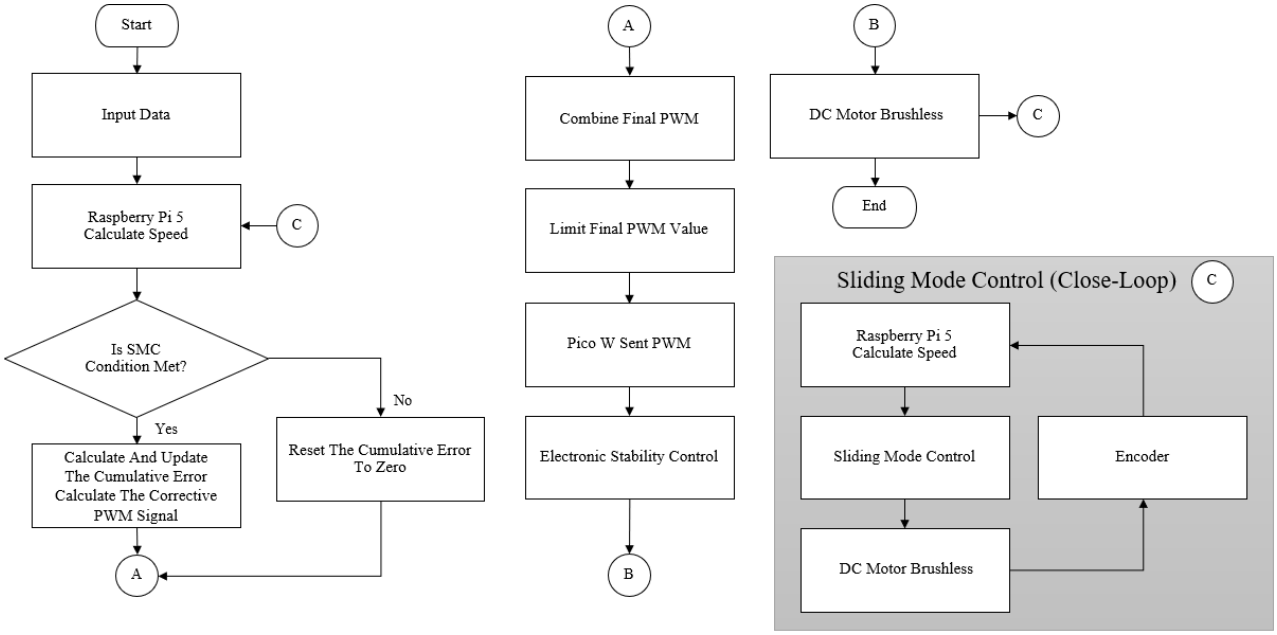

Fig. 2. System flowchart of the SMC-based motor control architecture, showing data acquisition, feedback processing, and PWM signal generation

The system first checks whether the SMC activation condition is met. This condition is triggered when the y-axis throttle value reaches 1.0, If the condition is not satisfied, the cumulative control error is reset to zero and the system operates in open-loop mode. Otherwise, SMC is activated.

The controller begins by calculating the tracking error $e(t)$ as the difference between the target speed v_{target} (set by the user via gear and throttle) and the actual speed v_{actual} as shown in Eq. (3).

$$e(t) = v_{\text{target}} - v_{\text{actual}} \tag{3}$$

where

$e(t)$ = is the tracking error at time t

v_{target} = the desired speed of the vehicle

v_{actual} = the current speed measured in real time

Next, the derivative of the tracking error $\dot{e}(t)$ is computed to capture the rate of change of the error, as shown in Eq. (4).

$$\dot{e}(t) = \frac{e(t) - e(t - \Delta t)}{\Delta t} \tag{4}$$

where

$e(t)$ = is the tracking error at time t

$\dot{e}(t)$ = derivative of the tracking error

Δt = Time elapsed since the previous measurement

These two values, $e(t)$ and $\dot{e}(t)$, are used to form the sliding surface $s(t)$, which governs the switching logic in SMC and is defined as shown in Eq. (1). Based on this surface, the system then applies the SMC control law to compute the corrective control signal u_{smc}, as shown in Eq. (2). This signal ensures robust and stable convergence toward the condition $s(t) = 0$, even in the presence of external disturbances. Finally, the generated u_{smc} is combined with a nominal PWM value from the open-loop path $u_{\text{open\_loop}}$ to produce the final control output u_{final}, as shown in Eq. (5),

$$u_{\text{final}} = u_{\text{open\_loop}} + u_{\text{smc}} \tag{5}$$

where

u_{final} = the composite PWM signal that incorporates both open-loop and feedback-based control

$u_{\text{open\_loop}}$ = the nominal PWM value

u_{smc} = the corrective PWM signal from the SMC controller

Before transmission, u_{final} it is constrained within safe operating bounds and sent via serial communication to the Pico W.

The Pico W forwards the bounded PWM to the motor driver and applies necessary steering corrections. An ESC is integrated to improve dynamic response, enhance speed tracking, and reduce overshooting to compute the real-time velocity used in v_{actual}, the system utilizes feedback from an optical encoder attached to the motor shaft.

The number of pulses pulse_count is measured within a defined sampling interval Δt, allowing the calculation of pulse frequency in hertz, as shown in Eq. (6).

$$\text{pulse\_per\_sec} = \frac{\text{pulse\_count}}{\Delta t} \quad (6)$$

where

pulse_per_sec	=	the number of pulses per second (Hz)
pulse_count	=	the number of pulses counted during time interval
Δt	=	sampling time in seconds

given the number of pulses per wheel revolution pulses_per_rev and the wheel diameter wheel_diameter the linear speed in meters per second is then calculated, as shown in Eq. (7).

$$\text{speed}_{\text{mps}} = \frac{\text{pulse\_per\_sec}}{\text{pulses\_per\_rev}} \times \pi \times \text{wheel\_diameter} \quad (7)$$

where

$\text{speed}_{\text{mps}}$	=	the speed in meters per second
pulse_per_sec	=	the number of pulses per second (Hz)
pulses_per_ref	=	the number of pulses per full wheel revolution
wheel_diameter	=	the diameter of the wheel in meters

To express this result in kilometers per hour, value is converted using the factor 3.6, as shown in Eq. (8).

$$\text{speed}_{\text{km/h}} = \text{speed}_{\text{mps}} \times 3.6 \quad (8)$$

where

$\text{speed}_{\text{km/h}}$	=	the converted speed in kilometers per hour
$\text{speed}_{\text{mps}}$	=	the speed in meters per second

This computed speed value is then used as the feedback input to the SMC loop on the Raspberry Pi 5, by integrating this real-time velocity measurement with the control law described previously in Eqs. (1)–(5), the system can continuously adjust control efforts to maintain target performance even under dynamic load conditions and external disturbances.

Before transmission, the final PWM output is constrained within safe operating bounds to prevent overdriving the actuators and is sent via serial communication to the Pico W, which forwards the bounded PWM signal to the motor driver and applies steering corrections through the servo motor as needed. An ESC mechanism is also integrated into the system to improve dynamic response, enhance speed tracking precision, and reduce overshooting during rapid transitions. Finally, encoder feedback is continuously routed back to the Pi 5 in real time, completing the high-frequency closed-loop control cycle and ensuring stability throughout operation.

3. Implementation

3.1 Circuit Diagram and Hardware Implementation of the SMC-Based Control System

The complete hardware architecture of the proposed SMC-based control system is illustrated in Fig. 3, which depicts the interaction between the power supply, processing unit, actuator interface, and feedback components. This modular design separates high-level computation from time-sensitive actuation, enhancing system reliability, maintainability, and scalability for embedded vehicular control applications.

At the system's core is a 12 V lithium-ion battery (15,000 mAh, 3 A continuous discharge), which serves as the primary power source. It delivers energy to subsystems through two dedicated buck converters, ensuring appropriate voltage levels and minimizing the risk of overvoltage or current surges. The first converter steps the voltage down to 5 V, supplying power to the Raspberry Pi 5 and the servo motor, both sensitive to fluctuations. The second converter provides a stable 7 V to the ESC, which controls the brushless DC (BLDC) motor. This separation of power domains reduces interference between digital and analogue sections, improving overall stability and safety.

The Raspberry Pi 5 functions as the central control unit. It runs the SMC algorithm, processes encoder signals, and communicates via UART (RX-TX-GND) with a secondary microcontroller—the Raspberry Pi Pico W. To prevent power rail noise, the same 5 V supply does not power the Pico W but instead draws power through the Pi 5's USB port, improving signal integrity and simplifying wiring.

After computing the PWM duty cycle based on control laws and real-time velocity feedback, the Pi 5 sends commands to the Pico W. Acting as the low-level actuator controller, the Pico W generates two outputs: a PWM signal to the ESC for motor speed control, and a servo angle command via GPIO 14. The servo motor shares power and ground with the Pi 5 to ensure synchronized and noise-free operation.

The ESC, powered by 7 V, delivers three-phase outputs (A, B, and C) to the BLDC motor. In parallel, three Hall-effect sensors (Hall A, B, and C) provide rotor position and velocity feedback to the Pi 5, enabling accurate speed estimation and closed-loop control. This feedback ensures the control algorithm adapts precisely to real-time operating conditions.

Overall, the proposed architecture supports robust, real-time embedded control for hybrid manual–automatic vehicle systems using cost-effective and widely available components. The separation of power, computation, and actuation layers enhances system modularity, safety, and flexibility, making the platform well-suited for experimental validation of advanced control strategies.

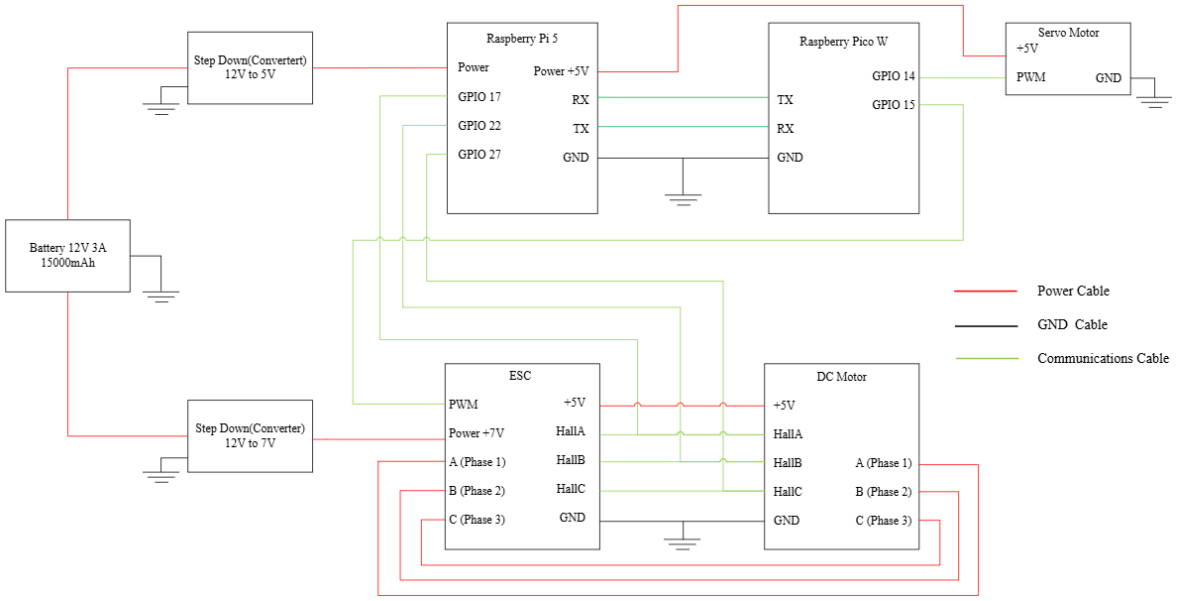

Fig. 3. Circuit diagram of the SMC-based vehicle control system, showing power distribution, communication links, and signal pathways between the main controller, actuator interface, and motor driver.

4. Results and Discussion

4.1 Performance Evaluation of the SMC-Based Control System

To evaluate the performance of the proposed SMC-based control architecture, a series of experiments was conducted using a custom-designed chassis dynamometer capable of replicating realistic road load conditions. Tests were carried out at three discrete gear settings—Gear 1 (15 km/h), Gear 2 (30 km/h), and Gear 3 (40 km/h)—under two test conditions: no-load, where the vehicle wheels rotate freely without contact with the ground, and load, where the vehicle is placed on the dynamometer with applied mechanical resistance to simulate real driving conditions.

The primary performance metrics recorded were the average PWM duty cycle (%) and current draw (A), representing control effort and energy consumption, respectively. Fig. 4. as shows in the box plot comparing current draw across the three gears under both no-load and load conditions. It is evident that the current consumption remains low and highly stable in all gear levels under no-load. In contrast, the load condition significantly increases, especially in higher gears. Notably, Gear 3 under load reaches peak current values exceeding 3.3 A, indicating increased torque demand and mechanical stress.

Fig. 5. as shown in the box plot for the PWM duty cycle. Despite the added load and speed variation, the PWM values remain tightly bound across all gears, demonstrating the SMC controller's ability to preserve control stability, reject disturbances, and prevent overshooting.

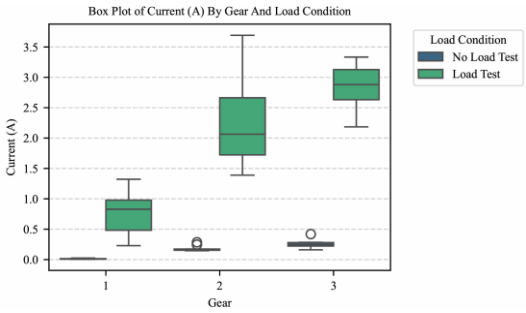

Fig. 4. Box plot of motor current (A) across three gear levels under load and no-load conditions. Load simulates driving resistance; no-load allows free wheel rotation.

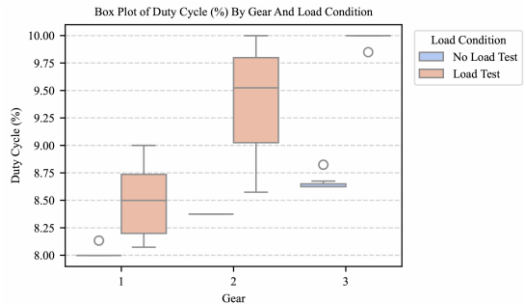

Fig. 5. Box plot of PWM duty cycle across gear levels. SMC maintains tight bounds under load and no-load conditions.

Under the no-load condition shown in Table 1, the system maintained low and consistent duty cycle values with minimal current draw. For instance, Gear 1 recorded an average duty cycle of 8.010 % and a current of only 0.011 A, whereas Gear 3 slightly increased to 0.267 A due to higher target speed. The small variation in duty cycle reflects precise control performance when external torque is negligible.

Table 1. Experimental results under no-load condition. The SMC performance demonstrates stable duty cycles with minimal current draw across all gears.

No load				
Gear	**Avg Duty Cycle (%)**		**Avg Current(A)**	
1	8.010 %		0.011 A	
2	8.374 %		0.159 A	
3	8.646 %		0.267 A	
	Duty Cycle (%)		**Current(A)**	
Gear	**Max**	**Min**	**Max**	**Min**
1	8.133 %	7.998 %	0.025 A	0.006 A
2	8.374 %	8.374 %	0.172 A	0.146 A
3	8.824 %	8.624 %	0.149 A	0.211 A

Under the load condition shown in Table 2, the system required significantly more control effort. Gear 3 recorded the highest average current of 2.869 A, along with a duty cycle of 9.985 %. However, even in this scenario, the duty cycle remained within a narrow operating range (9.999 % max, 9.849 % min), confirming the robustness of the SMC algorithm in rejecting disturbances and maintaining target velocity.

Table 2. Experimental results under load condition. Increased current reflects higher torque demand, while duty cycles remain tightly bounded, confirming control stability.

Load				
Gear	**Avg Duty Cycle (%)**		**Avg Current(A)**	
1	8.279 %		0.793 A	
2	9.178 %		2.077 A	
3	9.985 %		2.869 A	
	Duty Cycle (%)		**Current(A)**	
Gear	**Max**	**Min**	**Max**	**Min**
1	9 %	7.998 %	1.322 A	0.228 A
2	9.799 %	8.574 %	3.692 A	1.390 A
3	9.999 %	9.849 %	3.334 A	2.185 A

These findings confirm the effectiveness of the SMC scheme in regulating vehicle speed under dynamic and resistive environments. The controller compensates for load-induced variations while maintaining bounded actuation, which helps reduce system stress and enhances long-term reliability. Performance consistency across gear levels supports the suitability of this control approach in hybrid manual–automatic vehicle systems requiring real-time feedback control.

5. Conclusion

This paper presented the design and implementation of a robust SMC-based control system for a remote-controlled vehicle using Raspberry Pi 5 and Pico W. The system combines manual throttle, steering, and gear input with automated speed regulation through a real-time feedback loop driven by encoder data. To ensure precision and operational stability, a structured and modular hardware architecture—featuring independent power regulation, UART-based communication, and closed-loop motor control—was developed.

Experimental validation was done using a chassis dynamometer under no-load and loaded conditions. The results confirmed that the SMC algorithm effectively maintained target speeds while compensating for mechanical disturbances. The PWM duty cycles remained tightly bound, and current responses were stable across all gear levels, demonstrating robustness and control accuracy.

Overall, the findings demonstrate the feasibility of deploying sliding mode control on low-cost embedded platforms for hybrid manual–automatic vehicle systems. As future work, the system can be enhanced by incorporating wireless control input, adaptive gain tuning, and replacing the basic signum function with a boundary layer or saturation function to reduce chattering. These improvements could increase energy efficiency and adaptability under more complex or time-varying load conditions.

Reference

[1] L. G. Wu, J. Liu, S. Vázquez, and S. K. Mazumder, “Sliding mode control in power converters and drives: A review,” *IEEE/CAA J. Automatica Sinica*, vol. 9, no. 3, pp. 392–406, Mar. 2022.

[2] X. Xiong, S. Kamal, and S. Jin, “Adaptive gains to super-twisting technique for sliding mode design,” presented at *IEEE Conference* (arXiv preprint), May 2018.

[3] H. Yajima and K. Takami, “Inter-Vehicle Communication Protocol Design for a Yielding Decision at an Unsignalized Intersection and Evaluation of the Protocol Using Radio Control Cars Equipped with Raspberry Pi,” *Computers*, vol. 8, no. 1, Art. 16, 2019.

[4] X. Wang, “Field oriented sliding mode control of surface-mounted hybrid electric vehicles under automotive dynamometer load testing,” *Proc. IEEE Veh. Tech. Conf.*, 2018.

[5] R. Singh *et al.*, “MRAS-based Integral Sliding Mode Control of Electric Vehicles under Speed and Load Torque Fluctuations,” in *Proc. IEEE PEDES*, Dec. 2022.

[6] K. Rsetam *et al.*, “GPIO-based continuous sliding mode control for networked control systems under communication delays with experiments on servo motors,” *IEEE/CAA J. Autom. Sinica*, vol. 12, no. 1, pp. 99–113, Jan. 2025.

[7] T. Nayl, “Design and experimental evaluation of a novel sliding mode control scheme for articulated vehicle,” *Mechatronics*, 2018.