Automatic Complex-terrain Vehicle with Transformable Wheels

Xinyi Zhu, Haoran Zhang, Ruikai Yang, and Qianhui Zheng, *Student, Shanghai Jiao Tong University*

***Abstract*—In this report, we propose a transformable robot with wheel-leg desgin. When navigating obstacles and sandy terrain areas, robots with traditional wheels often struggle to complete exploration tasks. Some passive deformable wheels, which rely on ground friction, can be significantly affected by environmental interference. We aim to achieve active switching between two wheel states using a four-bar linkage system. When the wheels fold into a circular shape, the overall vehicle height decreases, allowing for rapid travel over flat surfaces and low-clearance areas. When the wheels expand into a leg-like shape, they can effectively climb onto platforms and traverse sandy terrain and other special environments. To reduce energy loss during mechanical transmission, we placed the motor at the inner axis of the wheel and the servo at the outer axis. To prevent interference between their operational circuits, we added a slip ring on the motor shaft to ensure the wires do not tangle. The diameter of our wheel is 82.32mm when retracted and 123.48mm when expanded, achieving a stretch ratio of 150%, perfectly completing the task of climbing a 60mm step. On smooth surfaces, the robot's wheels remain in the retracted state, reaching a speed of 0.018m/s. On sandy terrain, thanks to the larger diameter of the expanded wheels, the robot's speed slightly increases to 0.023m/s. Our robot successfully passed an environmental simulation composed of height limits, sandy terrain, steps, and turning indications, demonstrating its motion stability. Thanks to the support of distance sensors, cameras, and the Raspberry Pi's algorithms, our robot can autonomously adjust its direction based on the environment, completing exploration tasks entirely independently without human intervention. We believe our design is meaningful for the development of robots with transformable wheels.**

***Index Terms*—Transformable wheel, four-bar linkage, step-climbing, auto nevigation**

I. Introduction

With advanced technology, unknown areas are becoming known. From the corners of the Earth to the depths of space, the steps of human exploration are reaching places previously unimaginable. In such environments, the living conditions are highly possible to be severe for human beings. We need a multi-functional robot to serve as our sensors and explore these places. In this paper, we aim to proposed an improved design of transformable wheels to address the needs of Mars exploration. Surface soil on Mars is sand-like, so we require the robot to move on sand and climb steps. For safe navigation, we need the robot to meet height restrictions and learn obstacle avoidance. Performing well under these constraints, we believe our design of transformable wheels means something for the development of human exploration.

Xinyi Zhu, Haoran Zhang, Ruikai Yang, and Qianhui Zheng are with the Mechanical Engineering major, UM-SJTU Joint Institute, Shanghai Jiao Tong University, Shanghai, China, 200240.

Previous researches have given us many inspirations on different ways of transformation for our robot. The elastic-leg based jumping robot demonstrates powerful jumping capabilities with an adjustable take-off angle but faces limitations in mobility and high voltage requirements [1]. The origami-based transformable wheel robot effectively navigates narrow slits with minimal mechanical parts but struggles with deformation stiffness and limited force generated by the SMA springs [2]. The wheel-and-leg transformable robot achieves high degrees of freedom with a single motor but relies heavily on friction and gravity, requiring in-place turns before wheel deployment [3][4][5][6]. Meanwhile, the transformable wheel with an umbrella-like leg shows improved stability in rough terrains but lacks smooth movement on plain areas, especially after transformation [7]. As shown in Fig. 1, these designs excel in their respective fields, but some limitations make them less suitable for our current Mars exploration applications.

Other designs, such as those using wheel-track or leg-track mechanisms, offer faster and more stable performance on sandy terrains. However, a common problem for them is that they all struggle with height constraints and require better motors [8][9]. Our approach, the active wheel-leg transformable robot, aims to enhance stability when climbing obstacles, improve control during turns, and increase adaptability to sandy environments. As we will demonstrate later, our robot can stably handle various terrain challenges while accurately capturing signals from the surrounding environment. This makes our design highly aligned with the application goals of the Mars rover. Despite these advancements, our design still faces challenges like limited velocity due to higher torque requirements and low synchronicity of wheels after turning. We will try to discuss these problems in later sections.

II. DESIGN

A. Overview of the wheel

Fig. 2 is the CAD drawing and the real wheel opening process.

B. Graphical linkage synthesis

In this part, we will discuss how we design the wheels. According to the game rule, the total height can't exceed 0.09m and the wheel need to climb on a stairs of 0.06m. To makes the transformable wheel able to climb on the stairs while not violating the rule, we design the wheel radius to be 40mm.

No.	Method	Major Findings	Unsolved Issues	Refs.
L1	Elastic-leg based jumping robot	A four bar linkage leg with elastic elments allows powerful jumping with adjustable take-off angle.	Mobility is limited and sensors are fixed. Foldable wings need further research. Volatge required is high.	[1]
L2	Origami based transformable wheel robot	Magic ball origami structure helps robot path through narrow slits with minimum mechanical parts.	The deformation stifness and the force generated by the SMA springs are currently limited.	[2]
L3	Wheel-and-Leg transformable robot	Use one motor for two modes within one wheel. Achieve high DoF with simple mechanism.	Rely heavily on friction/gravity to open/close the wheel. Need to turn in place before opening wheels.	[3][4] [5][6]
L4	Transformable wheel with umbrella-like leg	The umbrella-like structure shows improved stability and adaptability in rough terrains.	After transformation, This robot might not move smoothly enough on plain area.	[7]
L5	Others (wheel-track, leg-track, bigger wheel...)	Achieve faster and stabler performance when moving on sand with simple mechanism.	Difficult to pass through height constraints. Require better performance moter/servo.	[8][9]
L6	Our approach (Active wheel-leg transformable robot)	Higher stability when climbing onto obstacles, better control when making turns, and greater adability to sandy environments.	Limited velocity due to a higher requirement on the torque provided by motors. Low synchronicity of wheels after turning.	This work

Fig. 1. The literature review table.

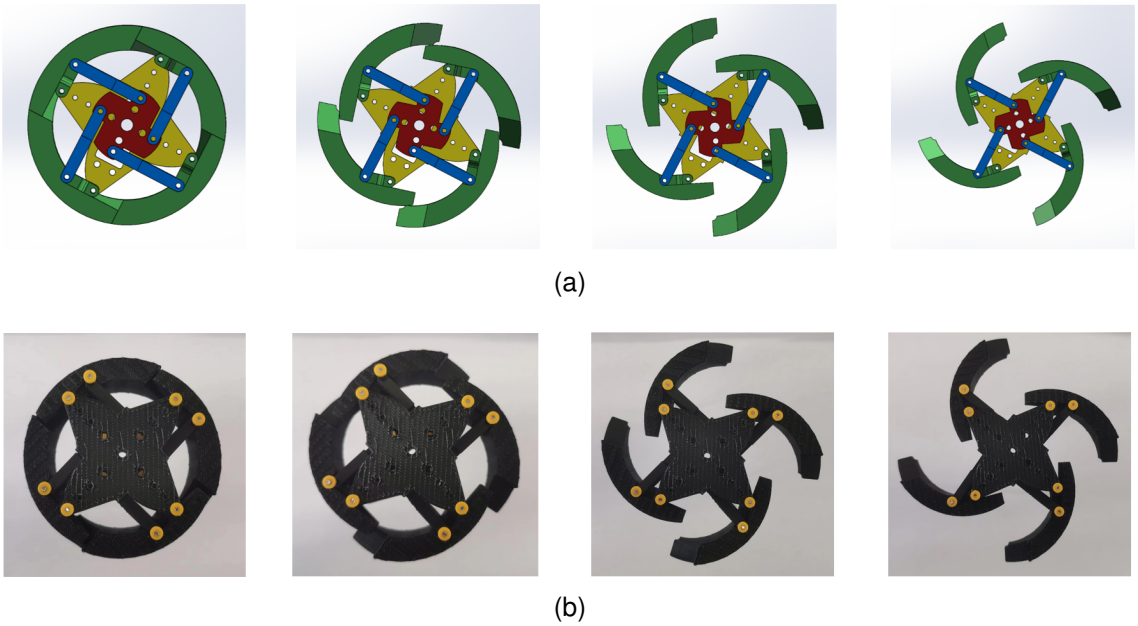


Fig. 2. (a) The simulated wheel opening process (b) The real wheel opening process.

We use two-position synthesis and the designed length of the linkages to design the four bar linkage and the following is the design process. Fig. 3 is the synthesis of our wheel.

- To get the desired leaf motion, draw the link CD in its two desired positions, C_1D_1 to C_2D_2
- Finding the bisectors of C_1C_2 and D_1D_2,where O_1 and O_2 are on
- To make the desired link length to be 30mm, draw a circle of 30mm from C_1 and D_1
- Intersect the circle with each bisector and get O_1 and O_2

C. Linkage size

The opening and closed size are shown in Fig. 4.

1) The shrink size:

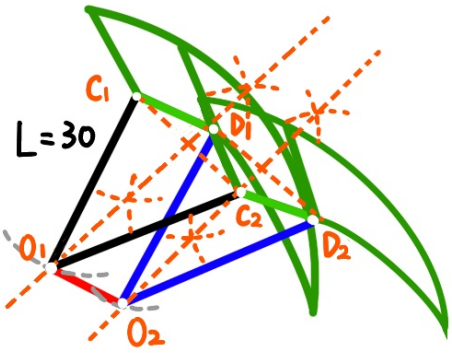


Fig. 3. The two position synthesis.

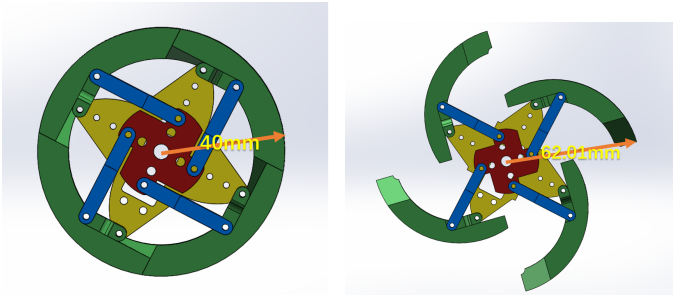


Fig. 4. (a) The closed size (b) the opening size.

- Theoretical: We design the wheel with a diameter of 80mm. Then the shrink size of the wheel is of the radius 40mm.
- Practical: The radius of the wheel is of 41 mm.

2) The expansion size:

- Theoretical: According to the CAD drawing shown in Fig. 5, the maximum opening angle the wheel can expand is $46.4°$
  The radius of the open wheel mode of the wheel is 62.01mm, which is more than 60mm to guarantee that the vehicle is able to climb up the stairs.

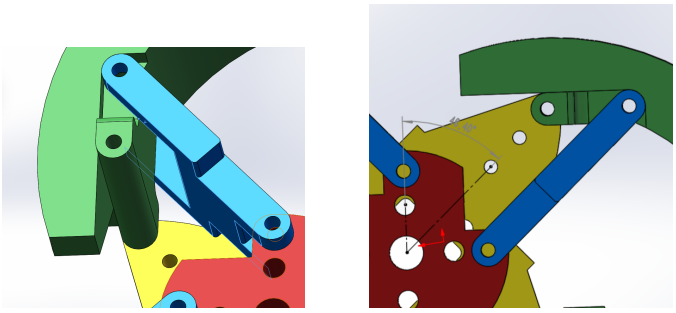


Fig. 5. (a) Boundary condition of expansion (b) maximum expansion angle.

- Practical: The maximum opening radius can reach 62mm.

D. Design & Creative Design

1) Active opening mechanism:

We independently control of drive and deformation to enhance system flexibility and responsiveness. The motor is used to control the star-shaped part of the wheel to make the whole wheel moving forward together.The servo motor control the central disc, which responsible for the four bar linkages to open or close the wheel.
A potential issue with passive deformation wheels in many published work is that if only one wheel encounters an obstacle and deforms, the rest of the wheels remain unchanged.To address this issue, active deformation wheels offer a solution. An active system can monitor the status of all wheels and adjust their shape as needed, ensuring the vehicle's stability and control.

2) Slip ring:

When using the active opening mechanism, the wiring of the sever motor will be a problem. To solve the problem of rotational entanglement between servo motor and the car body, we use slip ring. This ensures the continuous transmission of power and signals between the stationary and rotating parts of the system.
The published work offers a different solution: they use a hollow shaft brush-less motor. However, the cost is so high that it becomes impractical for our application. The high expense of the hollow shaft brush-less motor makes it unsuitable for projects with budget constraints, despite its advantages in terms of reliability and performance.
Therefore, the slip ring presents a more cost-effective and feasible alternative. This solution balances the need for functionality and budget, making it an ideal choice for ensuring the active opening mechanism operates smoothly and reliably.

3) Other designs:

- The thickness of the wheel: We make the thickness of the wheel to be 20mm which is not too thin. The increased thickness provides a larger surface area, which enhances support on soft, sandy terrain. The additional thickness contributes to greater stability, reducing the chances of the vehicle tipping over or getting stuck. Thicker wheels can grip the sand better, providing improved traction.
- Tread patterns: We add tread patterns on the tire surface to add the tire's friction coefficient. This can increase traction and prevent slipping.

E. Prototype

Fig. 6 is the CAD drawing of the whole car.
Fig. 7, Fig. 8 and Fig. 9 are different views of our prototype.

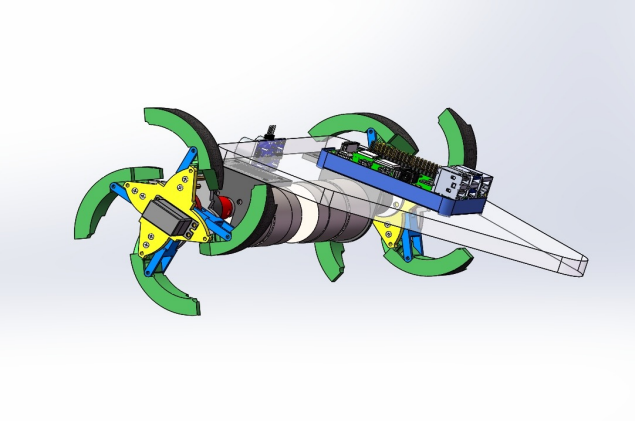


Fig. 6. The CAD drawing of our car.

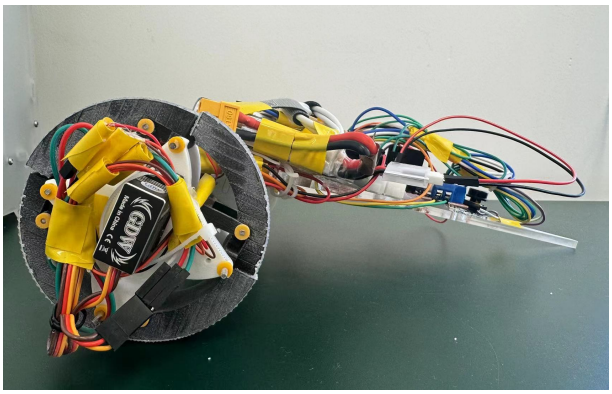


Fig. 7. The prototype side view.

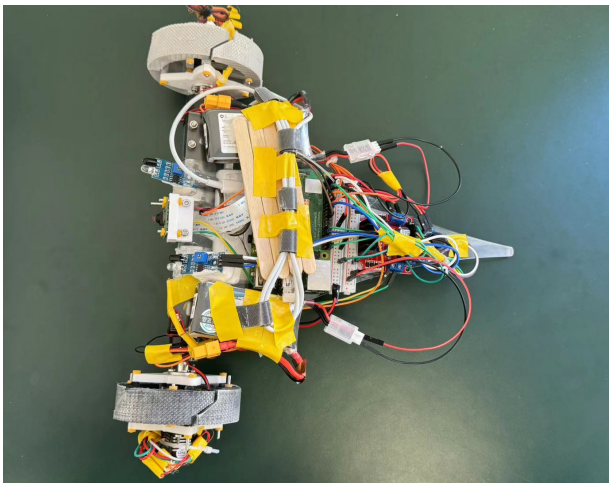


Fig. 8. The prototype top view.

III. MANUFACTURING

A. Selection of materials

1) Part materials:

- Wheel: The material must be lightweight, and cost-effective. It should also be easy to modify and support tensile loads with minimal displacement. Given these needs, Polylactic Acid is ideal. PLA offers uniformity, low weight, affordability, and sufficient tensile strength, making it a suitable choice.
- Car body: To minimize the burden on the wheels, it is essential to keep the car body lightweight. PMMA, is a suitable choice due to its combination of high strength and durability while remaining relatively light.

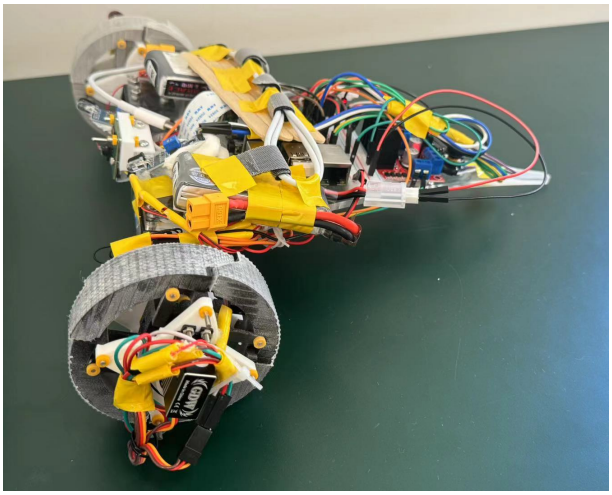


Fig. 9. The prototype top left view.

2) Motor:

We use the metal motor instead of other choices, like plastic. Metals provide higher mechanical strength and durability, allowing motors to withstand operational stresses and loads without deformation or failure. Also, their performance benefits often outweigh the expense.

3) Sensor:

The materials of ultrasonic sensors include piezoelectric ceramics, plastic housings, acoustic coupling materials, and PCB materials. These materials are resistant to aging, robust, corrosion-resistant, and have a long service life, ensuring the stable performance and reliability of the ultrasonic module.

B. Selection of manufacturing methods

1) Wheel:

The parts of the wheel are highly customized so the material-adding process is 3D printing. This method allows us to easily tailor components according to specific requirements. It's convenient to iterate designs and produce custom parts as needed through 3D printing.

2) Car body:

For the PMMA car body, we employed laser cutting method to manufacturing them. Due to the relatively large size and small, fixed height, the 2D CAD drawing is adequate. This makes laser cutting much faster and better than 3D printing in producing the car body.

C. Procedure for manufacturing

1) Wheel:

The highly customized wheel parts are manufactured using 3D printing. (Some parts' CAD drawings are shown in Fig. 10). The process involves creating a detailed 3D CAD model, selecting the appropriate printing material, and using a 3D printer to build the parts layer by layer. Then we assemble the parts together.

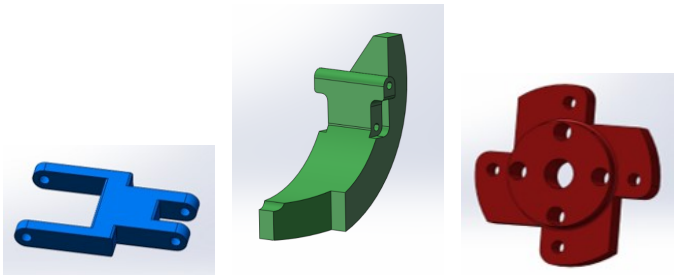


Fig. 10. The highly customized parts of the wheel.

2) Car body:

The PMMA car body is manufactured using laser cutting. A precise 2D CAD drawing is created, and PMMA sheets are prepared. The drawing is based on the designed size and the prepared screw hole for the following assembly. The laser cutter is configured with the CAD drawing, and the PMMA sheet is cut according to the design.

D. Assembly procedure

Fig. 11 is the explosion figure of the wheel and the motor.

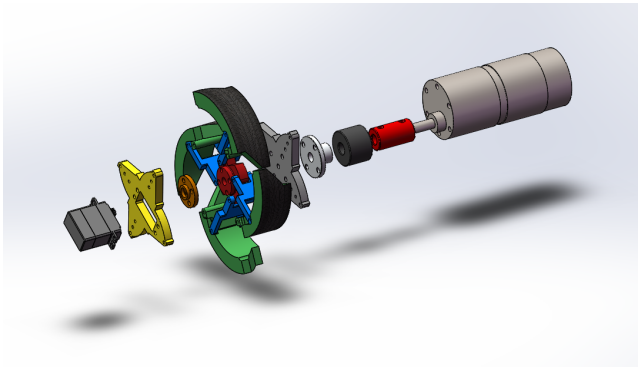


Fig. 11. Explosion view of the assembly.

1) Wheel:

We use smooth shafts and tight shaft sleeves as revolute joints because it is at low cost and it is easy to assemble and maintain. Also, as the wheel is a linkage system, we need the wheel to move smoothly on the shaft so assembling by the tight shaft sleeves can both ensure the smoothness of rotation and the structural stability. The assembly of the wheel is shown in Fig. 12.

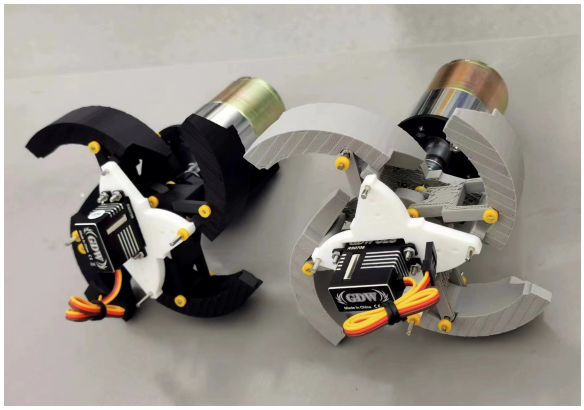


Fig. 12. The assembly of the wheel.

2) Wheel and the motor:

Because the length of the slip ring is longer than the motor shaft, to connect the motor to the wheel, the length of the shafts may need to be increased. The connection of the two shafts is achieved by using a coupling. Then, we use another flange coupling to connect the shaft to the wheel.

3) Motor and car body:

Attach the motor mounting bracket to the car body using screws, bolts, or other appropriate fasteners. Then secure the motor to the bracket using screws.

4) Sensor:

We use two infrared obstacle avoidance modules. We use super-light clay to make a support of the infrared obstacle avoidance module. As we need to set a certain angle of the position of the sensor and don't have a requirement on the shape of the support. Using clay to assemble is easy and flexible.

5) Camera:

We 3D printed a support and used a shaft and sleeves to fix the camera on the support. Then we use the screw to fix the support with the camera on the car body. The support is shown in Fig. 13.

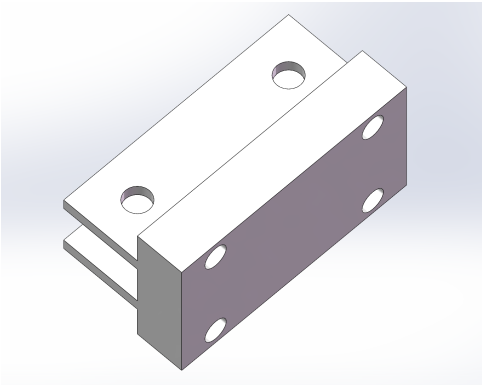


Fig. 13. The support of the camera.

IV. CONTROL OF SENSORS AND ACTUATORS

We develop an open loop to control our car. Each sensor is responsible for the different functions of the car.

A. Overall Control Logic

We divide the whole control into four parts, including how to go in a straight line, when to transform wheels, how to determine which way to turn, and how to decide which line to stop at. Fig. 14 and Fig. 15 are the control logic for the 4 sub-tasks and we will discuss them in detail afterward.

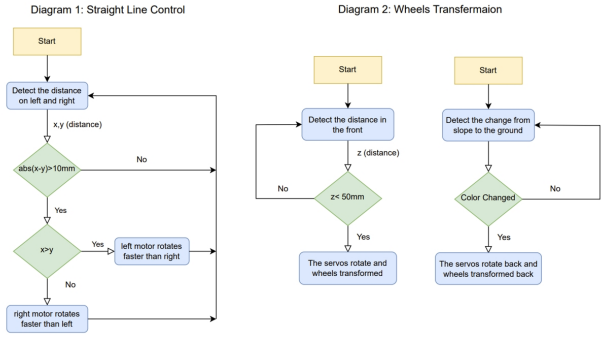


Fig. 14. Flowchart I.

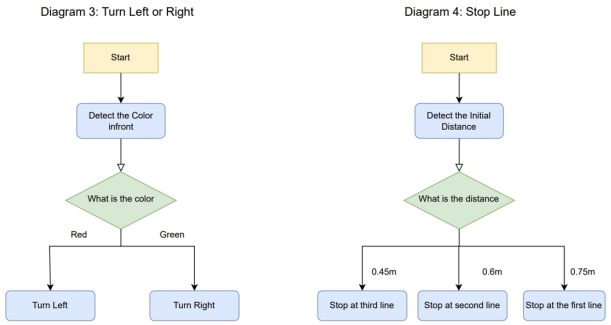


Fig. 15. Flowchart II.

Fig. 16 is our overall circuit diagram, including distance sensors, a camera as the sensors, and motors and servos as the actuators.

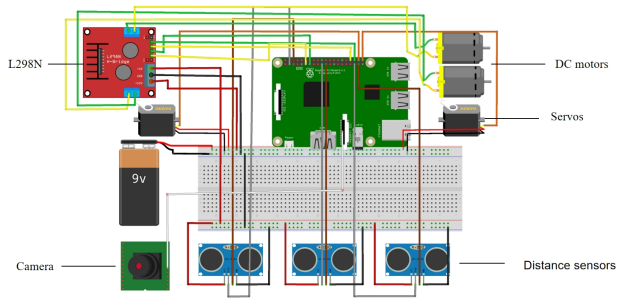


Fig. 16. Circuit Diagram.

B. Sub-tasks I: Going straight line

The distance sensor is mainly responsible for this part, we intend to do the PID control to make sure the car moves in a straight line, however, due to the inaccuracy of the sensors, we give it up. Instead, we add a delay when the step is detected, so that the wheel that first arrives at the step will stop by the step and wait for the slower one to be close to the step, so the going straight line task is physically fixed.

C. Sub-task II: wheels transformation

When we detected the distance was very close to the step, we transformed our wheels using the servos. The wheels remain opened until the second height limit is determined, and the wheels will transform back to pass the height limit.

D. Sub-task III: Turning

Using the camera to decide which way to turn, and the motors work during the pre-decided time to make a turn that is roughly 90 degrees.

E. Sub-task IV: Stopping

Based on the time it takes from starting to the first time wheel transformation, we acknowledge which line we start and then decide how long to go by controlling the working time of the motors.

V. ANALYSIS

A. Classification of the designed linkage

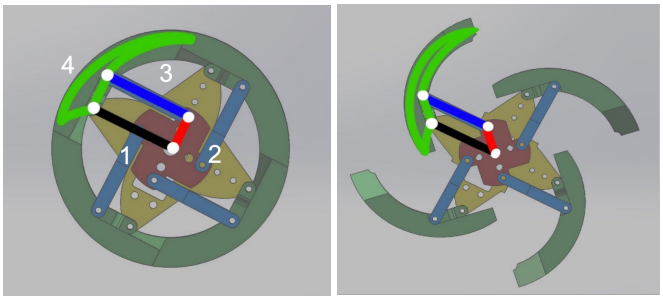


Fig. 17. Four bar linkage mechanism of the designed wheel. The red link is the input and the green link is the output. (a) Wheel mode. (b) Leg mode.

The designed linkage mechanism can be divided into four identical sections, each of which functions as a simplified four-bar linkage, as shown in Fig. 17. Each section contains four links and four revolute joints. Then, the degree of freedom (DOF) is calculated to be 1:

$$L = 4, J_1 = 4, J_2 = J_m = 0$$

$$M = 3 \times (4 - 1) - 2 \times 4 = 1DOF.$$

This implies that the mechanism has one input (the central disk connected to the servo motor) corresponding to one output

(the lobe of the wheel), ensuring a predictable and controlled movement for the entire system.

Given the dimensions: shortest link $S = 12$ mm, longest link $L = 29.6$ mm, and the remaining two links $P = 12$ mm and $Q = 29.6$ mm, we find that $S + L = P + Q$. This configuration classifies it as a special-case Grashof linkage with two equal pairs, specifically an S2X parallelogram. Consequently, the mechanism is a crank-rocker with double changing points if there are no constraints on the input. However, since the input is restricted by the design of the wheel, a further position analysis is required.

B. Position analysis for transformation

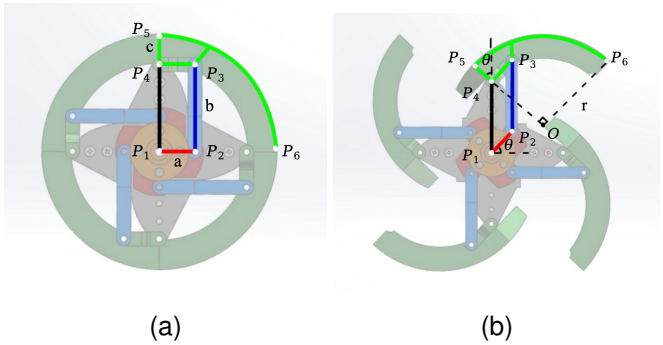


Fig. 18. The position analysis for the transformation. (a) Wheel mode. (b) Leg mode.

As is shown in Fig. 18, we fix the center of the wheel as the origin and derive the six points' positions as follows:

$$P_1(0,0), P_2(a\cos\theta, a\sin\theta), P_3(a\cos\theta, a\sin\theta + b)$$

$$P_4(0,b), P_5(-c\sin\theta, b + c\cos\theta)$$

$$P_6(a\sin\theta + r\cos\theta, b - a\cos\theta + r\sin\theta)$$

Therefore, the relation between the expansion radius R and the input angle θ after is derived as:

$$R = P_1P_6 = \sqrt{(a\sin\theta + r\cos\theta)^2 + (b - a\cos\theta + r\sin\theta)^2}.$$

The parameters in this expression are given as:

$$a = 12mm, b = 29.6mm, r = 40mm, \theta_{max} = 46.4^\circ$$

by our intentional design introduced in the Design section.

Then we employed Matlab to plot the angle variation of the expansion radius, which is shown in Fig. 19. From the plot, we observe that the radius keeps increasing before the input angular displacement reaches its maximum. Therefore, the maximum expansion radius of the wheel is $62.01mm$.

To ensure that the expansion radius is sufficient to pass the obstacle, we calculate the maximum height that our wheel can overcome and the angle at which our wheel contacts a 60mm obstacle during real-world testing. According to the sketch shown in Fig. 20, the ideal maximum height of the obstacle is:

$$h_{max} = x_6 + y_6 = a\sin\theta + r\cos\theta + b - a\cos\theta + r\sin\theta.$$

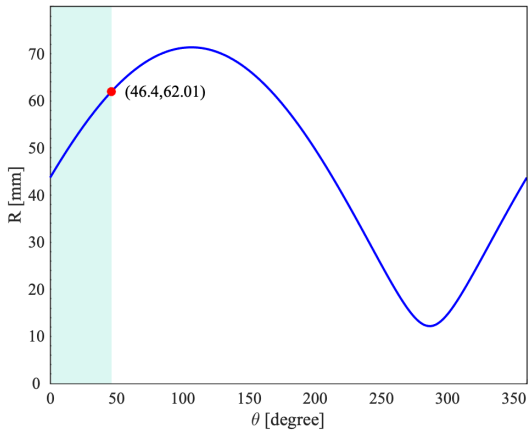


Fig. 19. The relation between the maximum wheel radius and the input angular displacement.

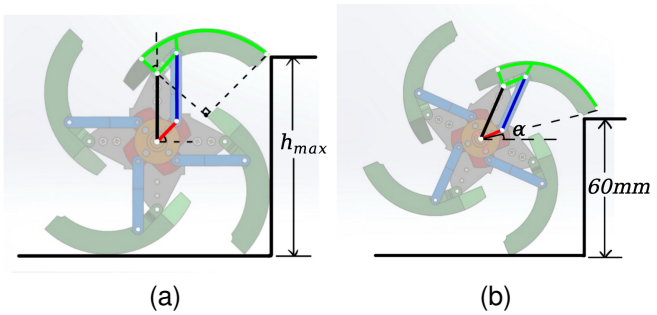


Fig. 20. The position analysis for climbing the steps. (a) The ideal maximum height. (b) The real objective of 60mm.

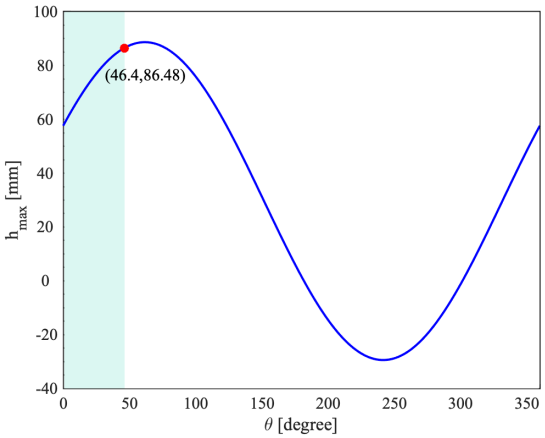


Fig. 21. The relation between the ideal maximum height and the input angular displacement.

We again employed Matlab to find the maximum point. As shown in Fig. 21, the ideal maximum height of the obstacle is 86.48mm, much greater than the targeted 60mm.

The angle α when the leg first contacts 60mm obstacle is calculated as:

$$\alpha = \arccos \frac{\sqrt{R_{max}^2 - (60 - a\sin\theta - r\cos\theta)^2}}{R_{max}}$$

The result shows $\alpha = 27.38°$ by taking in the parameters. This angle is then used in the force analysis to calculate the initial torque we need when passing the obstacle.

C. Force analysis

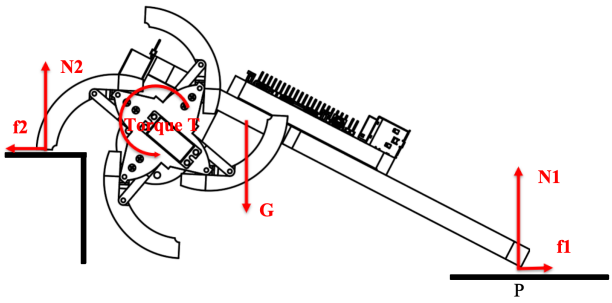


Fig. 22. Free body diagram of the wheel passing obstacles.

We perform a quasi-static analysis to find the minimum torque required for the motor to climb an obstacle. From the free body diagram shown in Fig. 22, three equilibrium equations of the whole wheel are derived:

$$\sum F_y = 0 \Rightarrow N_1 + N_2 - G = 0$$

$$\sum F_x = 0 \Rightarrow f_1 = f_2$$

$$\sum M_P = 0 \Rightarrow G \cdot l_1 - N_2 \cdot l_0 = 0$$

Also, the equilibrium equation of the wheel is:

$$\sum M_{wheel} = -R\cos\alpha \cdot N_2 - R\sin\alpha \cdot f_2 + T = 0.$$

Taking in the parameters $\mu = 0.3, R = 62.01mm, G = 12N, l_0 = 27cm, l_1 = \frac{2}{3}l_0 = 18cm, \alpha_0 = 27.38°$, we are able to derive the expression of the torque provided by the motor T as:

$$T = 0.536sin(\alpha) + 0.162cos(\alpha).$$

Using Matlab, we plot the angular variation of torque, as shown in Fig. 23. The maximum torque required for the motor is $T_{max} = 0.56N \cdot m$. Thus, the minimum torque for the selected motor should be $T_{motor} > 0.56N \cdot m$. Our selected motor, the JGA370 DC gear motor with a no-load speed of 12 RPM and a rated voltage of 12V, has a rated loaded torque of 9 kgf·cm (0.88 N·m), which exceeds the required 0.56 N·m. Therefore, the motor selection is justified and will perform well.

The servo is responsible for folding and unfolding the wheel-leg transforming mechanism. Following a similar calculation procedure, we calculate the minimum torque of the selected servo. The free body diagram during wheel unfolding is shown in Fig. 24. Using the isolation method, we focus on one of the four parts of the wheel. The relationship derived for the torque provided by the servo is:

$$T - F_{wheel} \cdot L > 0.$$

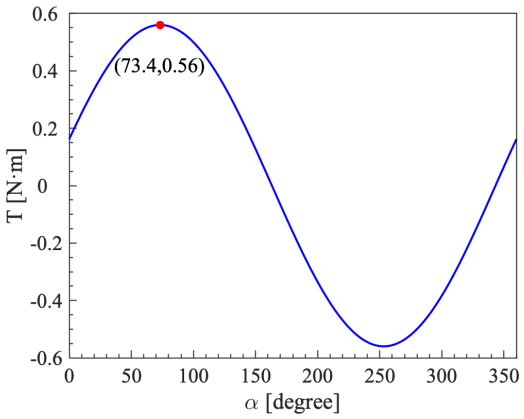


Fig. 23. The relation between the torque offered by the motor and the rotation angle of the center.

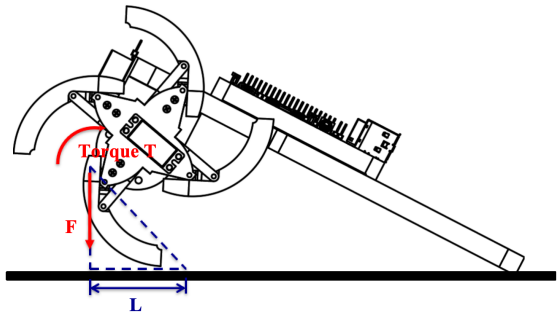


Fig. 24. Free body diagram of the wheel during wheel unfolding.

When the wheel is fully closed, L is at its maximum of around 30mm. Substituting the variables, we can calculate the minimum torque:

$$T > F_{wheel} \cdot L = 0.478N \cdot m.$$

We should also consider that the gravity of the other parts of the car and the supporting force from the ground will slightly increase the required torque. However, the torque of our selected servo is 7.5 kg·cm (0.74 N·m), which is significantly greater than the 0.478 N·m needed. Therefore, the selected servo is more than adequate for the task.

D. Rolling speed analysis

The motor we chose has a maximum no-load speed of 12RPM and a stall torque of 15kgf·cm (1.471N·m). The loaded torque we have is around 0.56N·m according to the force analysis above. Assume the motor has a linear torque-speed relationship, we calculate the loaded speed, ω, as follows:

$$\omega = \omega_{max}(1 - \frac{T}{T_s}) = 7.43rpm.$$

Given this rotational speed, we are able to calculate the rolling speed for different deformation modes of a wheel. In the shrink mode, the radius of the wheel $R_s = 40mm$. In the expansion mode, the radius of the wheel $R_e = 62.01mm$. Therefore, the rolling speed under the rated power are:

$$v_{shrink} = 2\pi R_s\omega = 0.031m/s$$

$$v_{expansion} = 2\pi R_e \omega = 0.048 m/s.$$

Considering the varying surface conditions, we account for different friction factors. The friction factor between PLA material and sand is approximately 0.5. We calculate the required motor torque in the sand as 0.67N·m, following a similar procedure to our force analysis but with the adjusted friction factor. Assuming the motor has a linear torque-speed relationship, we determine the loaded speed in sand, ω', as:

$$\omega' = \omega_{max}(1 - \frac{T}{T_s}) = 6.53 rpm.$$

The rolling speeds are then calculated as:

$$v_{ground} = v_{shrink} = 0.031 m/s$$

$$v_{sand} = 2\pi R_e \omega' = 0.042 m/s.$$

VI. EXPERIMENT

A. Demonstration of Load Carrying Capacity

We first show that the initial height of our car can pass the height limits. And then we put the load on the car and see if it can get over the step. Fig. 25 demonstrates the capacity of our car to climb the step with loads.

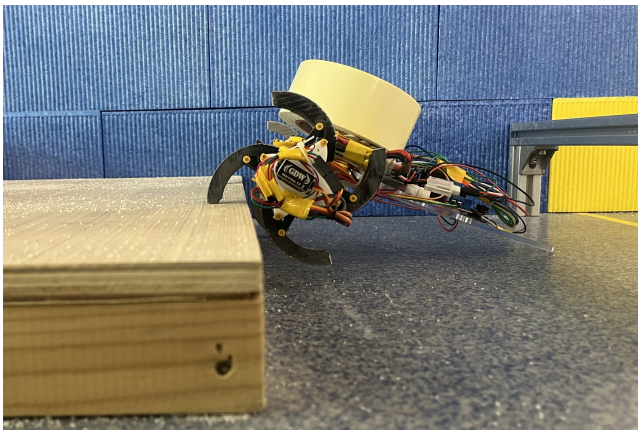


Fig. 25. Demonstration of load carrying capacity.

The weight of the car itself is around 1.2kg and the $T_{max} = 0.56m \cdot N$ as we have calculated before, the max weight in theoretical is $T_{max}/(mgRcos\theta)$. And we can finally obtain the value to be 0.92 kg according to the calculations.

The actual weight that it can carry is around 0.75kg, the reason may be that the mass isn't exactly loaded on the mass center of the car, and this will result in a relative error in the capacity:

$$E = \frac{0.92 - 0.7}{0.92} = 18.47\%.$$

B. Ability of Transformation

The ability of wheel transformation is detected by the 0.02mm-precision caliper. The theoretical value is obtained by the CAD drawing.

From Fig. 26, we can directly observe that the wheels can easily pass the height limit, and as Fig. 25 shows, when it's transformed into leg mode, it can easily climb up the step.

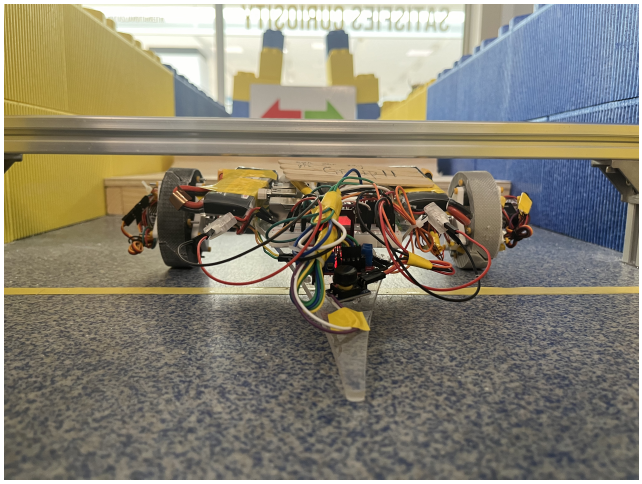


Fig. 26. Demonstration of the car passing the height limits.

TABLE I
THE SPEED ON THE SMOOTH SURFACE

	Actual dimension[mm]	Theoretical value[mm]	Error
Expansion	123.48	124.02	0.43%
Shrinkage	82.32	80.20	2.6 %

The error for the expansion part is mainly due to the PWM control, we can not reach the extreme value of the theoretical maximum, since we can't find the exact position. The servo may break down if we set the PWM to exceed the maximum. The error for the shrinkage mainly comes from the frictional material's thickness, which is roughly 1mm thick. Also, the sand may disturb the working state of the servo, making the servo work less accurate.

C. Speed On The Smooth Surface

We measure the speed by taking the average of 3 different parts (15cm each) as Fig.27a, in one run, and take the average of the time consumption.

TABLE II
THE SPEED ON THE SMOOTH SURFACE

	speed $[m/s^2]$	deviation from theoretical value
Trail 1	0.0179	42.2%
Trail 2	0.0182	41.3%
Trail 3	0.0180	42.0%

The error may come from that the original voltage for the motors should be roughly 12V, but our input is around 8.4V, which may lead to a much slower speed.

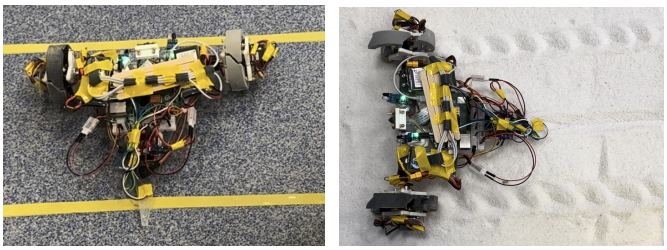
D. Speed On The Sand

we take the 60cm as a unit and record the time that the car passes the sand, and evaluate the average speed on the sand, like Fig. 27b.

Still, the error mainly comes from that the original voltage for the motors should be roughly 12V, but our input is around 8.4V, which may lead to a much slower speed. Also, the trail of the car is not that straight, which may also influence the final speed evaluation.

TABLE III
THE SPEED ON THE SAND

	speed $[m/s^2]$	deviation from theoretical value
Trail 1	0.0222	47.2%
Trail 2	0.0228	45.7%
Trail 3	0.0230	45.2%



(a) (b)

Fig. 27. Demonstration of the forward capacity of the car. (a) On the smooth surface. (b) On the sand surface.

VII. DISCUSSION

Since our robot is an improved version of Cao's design (OmniWheg), we mostly want to compare our work with that mentioned by Mr Cao [6]. To be noted, the car designed by Cao has four transformable wheels, compared to the only two wheels on our robot [6]. Their car does not meet any maximum height constraints, so the overall size and weight of the car are big, which requires a higher-torque motor ($1.779N \cdot m$) and servo ($0.876N \cdot m$) as described on page 3 of [6]. Thanks to their analysis provided, we have got a reference when selecting motors and servos. After our own experiments, we find out that we actually do not need motors with torques being that high. We finally chose two motors used in our prototype to be the JGA25-370 DC geared reduction motor. The reduction rate is 500, the speed is 12 rounds per minute, and the torque is $9Kgf \cdot cm$ or $0.882N \cdot m$ under a voltage of $12V$. With this choice, we immediately reduced about 0.2 kilograms from the total weight of the robot body, which means great relaxation for the servo.

As a result, the servo we finally chosen is also much smaller and lighter compared to the ones mentioned in Cao's paper [6]. Our choice, GDW DS290MG servo, offers $6Kgf \cdot cm$ or $0.588N \cdot m$ under a voltage of $8.4V$. With a much lighter design of the car body, the torque provided is actually more than enough. For our prototype, the two motors are powered with a battery with $8.3V$ voltage while the two servos are powered with another battery with 8.2 voltage. In future improvements, the voltage provided for the motor can be changed to $12V$, and then the rotating speed of the wheels can be more than doubled. This is very likely to help us out of the problem that the robot is moving too slowly on normal terrains.

Our design features numerous innovations. Firstly, in the design of the variable wheels, we generally followed Cao's four-bar-linkage system [6]. However, to prevent the wheels from spinning idly when stuck in sand, we added a layer of rubber to the outer surface of the wheels to increase friction. As the results of our experiment turned out, our robot was never stuck in the sand. Additionally, we modified the hollow structure between the inside of the wheels and the ground, allowing sand that enters the wheels to be expelled as they roll forward. These improvements enhance our robot's ability to traverse sandy terrain, enabling it to handle more complex exploration tasks.

Beyond the wheels, our design adheres to a minimalist philosophy. The chassis is primarily made from a single laser-cut acrylic sheet, significantly reducing the vehicle's weight while ensuring the rear end provides adequate support, thereby lessening the load on the motor and servo. For electronic control, we use Python language and Raspberry Pi for precise control. This not only provides technical support for more comprehensive information gathering and environmental detection but also confirms the practical feasibility of implementing navigation through computer vision technology. In our experiments, the robot successfully captured color signals ahead through its front camera and converted them into left/right turn execution signals.

Overall, our work represents an innovative practice in the research of variable-shaped wheeled robots. We addressed new constraints such as height limits, sandy terrain, and visual signal recognition by proposing new designs in various aspects and selecting new hardware to support these changes. Theoretically, we demonstrated that the wheel-leg type transformable wheel can be reduced to meet certain height restrictions. Although the servo motor adapted for this wheel is difficult to scale down proportionally, it can still meet new application conditions. In terms of practical application, we proved that the introduction of the Raspberry Pi significantly enhances the robot's intelligent exploration capabilities, enabling it to recognize external conditions for autonomous navigation. Additionally, through experiments, we developed an improved design better suited for sandy terrain, laying a solid foundation for future deployment of such exploration robots in more complex terrains and harsher environments. We believe that our work has played a crucial role in advancing and supporting the development of variable-shaped wheels.

VIII. CONCLUSION

In this paper, we presented an innovative design for a transformable robot equipped with variable-shaped wheels to address the challenges of Mars exploration, particularly navigating sandy terrain and overcoming obstacles. Our objectives were to create a robot capable of adjusting its wheel shape for enhanced mobility and stability in various terrains, and to validate this design through theoretical analysis and practical experiments.

We aimed to develop a robot with transformable wheels that could adapt to different terrains and constraints, such as height limits and sandy surfaces. Our design utilized a four-bar linkage system to actively switch between circular and leg-like wheel shapes. Additionally, we incorporated a slip ring to manage wiring entanglement and used a combination of motors and servos for wheel transformation and movement. Theoretical analysis included position and force calculations,

while practical experiments tested the robot's performance on different surfaces.

The findings of our study are noteworthy. We successfully achieved a stretch ratio of 150%, with the wheels' diameter being 82.32mm when retracted and 123.48mm when expanded, which enabled the robot to climb a 60mm step. Performance tests showed that on smooth surfaces, the robot reached a speed of 0.018m/s with retracted wheels, and on sandy terrain, the expanded wheels increased the speed to 0.023m/s. The robot maintained stability across simulated environments with height limits, sandy terrain, steps, and directional changes. Furthermore, autonomous navigation was successfully achieved using sensors and a Raspberry Pi for environment-based adjustments.

Our design provides a significant contribution to the development of transformable wheeled robots, especially for complex terrain exploration. The ability to actively switch wheel shapes enhances the robot's adaptability and effectiveness in various environments, making it a valuable tool for exploration missions. The integration of advanced control systems and intelligent navigation capabilities positions our robot as a versatile and autonomous solution for future exploration tasks.

In conclusion, our work demonstrates the feasibility and advantages of using transformable wheels for exploration robots. The theoretical and practical results validate our design's effectiveness in overcoming terrain challenges and performing autonomous navigation. This innovative approach offers a robust foundation for further development and deployment of robots in more complex and demanding exploration missions.

REFERENCES

[1] Y.-S. Kim, G.-P. Jung, H. Kim, K.-J. Cho, and C.-N. Chu, "Wheel Transformer: A Miniaturized Terrain Adaptive Robot with Passively Transformed Wheels," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Karlsruhe, Germany, 2013, pp. 5625-5630.

[2] D. Y. Lee, J. S. Kim, S. R. Kim, J. S. Koh, and K. J. Cho, "The deformable wheel robot using magic-ball origami structure," in *Proc. Int. Design Eng. Tech. Conf. Comput. Inf. Eng. Conf. (IDETC/CIE)*, Portland, OR, USA, 2013, vol. 55942, p. V06BT07A040, American Society of Mechanical Engineers.

[3] C. Zheng, S. Sane, K. Lee, V. Kalyanram, and K. Lee, "α-WaLTR: Adaptive Wheel-and-Leg Transformable Robot for Versatile Multiterrain Locomotion," *IEEE Trans. Robot.*, vol. 39, no. 2, pp. 941-958, Dec. 2022.

[4] Y.-S. Kim, G.-P. Jung, H. Kim, K.-J. Cho, and C.-N. Chu, "Wheel transformer: A wheel-leg hybrid robot with passive transformable wheels," *IEEE Trans. Robot.*, vol. 30, no. 6, pp. 1487-1498, Nov. 2014.

[5] Y.-S. Kim, G.-P. Jung, H. Kim, K.-J. Cho, and C.-N. Chu, "Wheel transformer: A miniaturized terrain adaptive robot with passively transformed wheels," in *Proc. 2013 IEEE Int. Conf. Robot. Autom. (ICRA)*, Karlsruhe, Germany, May 2013, pp. 5625-5630.

[6] R. Cao, J. Gu, C. Yu, and A. Rosendo, "Omniwheg: An omnidirectional wheel-leg transformable robot," in *Proc. 2022 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Kyoto, Japan, Oct. 2022, pp. 5626-5631.

[7] Y. She, C. J. Hurd, and H. J. Su, "A transformable wheel robot with a passive leg," in *Proc. 2015 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Hamburg, Germany, 2015, pp. 4165-4170.

[8] F. Zhou, X. Xu, H. Xu, T. A. Zou, and L. Zhang, "Transition mechanism design of a hybrid wheel-track-leg based on foldable rims," *Proc. Inst. Mech. Eng. Part C: J. Mech. Eng. Sci.*, vol. 233, no. 13, pp. 4788-4801, Jul. 2019.

[9] Z. Luo, J. Shang, G. Wei, and L. Ren, "A reconfigurable hybrid wheel-track mobile robot based on Watt II six-bar linkage," *Mech. Mach. Theory*, vol. 128, pp. 16-32, Oct. 2018.

APPENDIX

A. Contribution of each team member

Qianhui Zheng is currently a student at UM-SJTU JI, majoring in Mechanical Engineering, with plans to pursue further studies in Computer Science at the University of Michigan starting in the fall of 2024.

She is tasked with designing and manufacturing the transformable wheel and the whole car. She contributed to the design of CAD drawings for every wheel component and the selection of actuators. Additionally, she actively participated in assembling the overall structure.

Xinyi Zhu is studying the bachelor degree in mechanical engineering in SJTU-JI and is going to pursue a dual degree of computer science in University of Michigan. She is responsible for the design and manufacture of the transformable wheels. She draw some element parts of the wheel and engage in the manufacturing process. Also, she actively engage in the assembly of the whole car body. During the testing, she also help to test the functions and make adjustment on the car body.

Ruikai Yang is currently majoring in the mechanical engineering at Joint Institute in SJTU. He plans to further study in computer science in the University of Michigan in 2024 Fall. He enjoys combining the knowledge in his two majors. He is responsible for the design of car body. He designed and drafted CAD drawings for some support structure and coordinated part of the laser cutting and 3D printing tasks. He also participated in the adjustments of parameters for the controlling code.

Haoran Zhang is currently majoring in the mechanical engineering at Joint Institute in SJTU. He plans to further study in computer science in the University of Michigan in 2024 Fall. He is responsible for the the main part of the controlling code, and also in the circuit design and control logic design.

B. Gantt Chart

Fig. 28 shows the process of our project. We have a reasonable plan and finish all the tasks on time.

C. Budget table and Spending Table

Product	Budget (RMB)
Servos	200
Motors	200
Sensors	100
3D Printing	200
Battery	100
Others	200
Total	1000

Fig. 29. The budget table.

Fig. 29 shows our budget plan. The main expenses in our budget were on purchasing actuators and 3D printing. Items such as screws, bearings, and wires can be obtained for free at the Student Innovation Center.

The total spending for the project amounted to 1011.42 RMB, as shown in Fig. 30, which is very close to but overall exceeds our budget of 1000 RMB. We acknowledge that we bought some redundant and unnecessary items due to issues discovered with the initial prototype, leading to minor design modifications and ultimately causing the budget overrun.

D. Raspberry Pi programming code

The codes for the automatic control are posted in the next few pages.

Product	Parameter	Price (RMB)
Shafts	2mm*15mm*10, 2mm*20mm*10, 2mm*25mm*10, 2mm*30mm*10, 2mm*35mm*10, 2mm*40mm*10	50
Sleeves	4mm*7mm*12mm*2, 4mm*7mm*10mm*2	11.32
Rechargeable lithium battery	5V 3A	39.8
Power bank	5V, 5000ma*h, 15W	68.49
JGA25-370 DC gear motor	12V 12RPM*2	44
XC37GB30-C motor & support	13.5Kg*cm 12V*2	78
GDW DS290MG servo & raddar	6Kg*cm 4.8-8.4V*2	388.5
Conductive slide ring	inner7mm outer22mm*2	106
Coupler	6mm-6mm*3, 6mm-7mm *3	45
Flange coupling	TF1908 4mm*2 , TF2612 4mm*2, TF2612 6mm*2	77.14
Step-down module	MINI560, MP1584	17
Electromagnet	P20/15 24V M4	13.27
GP2Y0A21YK0F distance sensor	4-50cm detection range*2	46.6
Anti-slip tape	2.5mm	23.6
Total		1011.42

Fig. 30. The spending table.

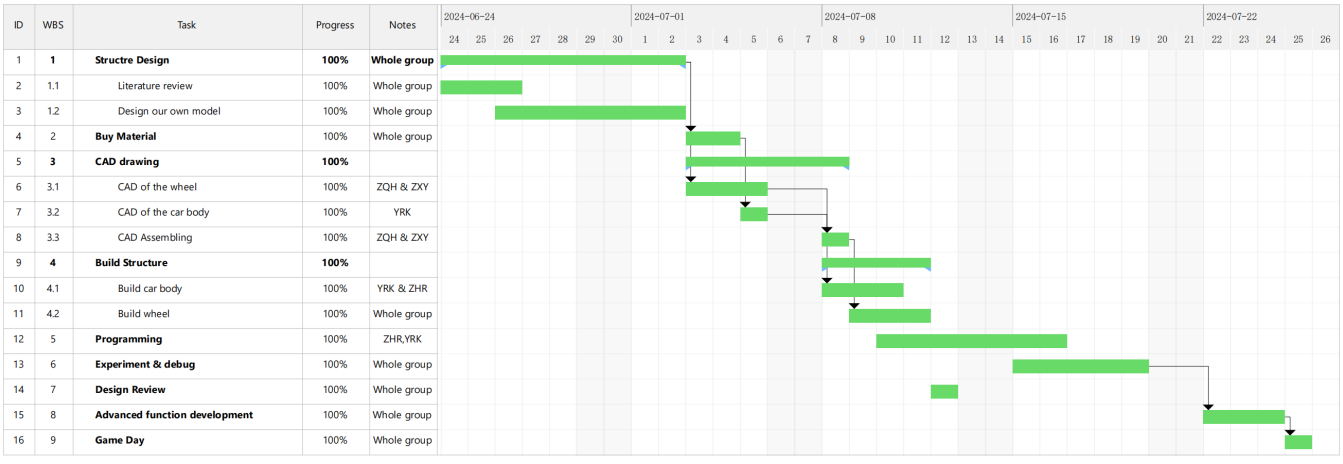


Fig. 28. Gantt chart.

```
import cv2
from picamera2 import Picamera2
import RPi.GPIO as GPIO
from time import sleep
import time
import numpy as np

GPIO.setmode(GPIO.BCM)

class Motor():
    def __init__(self,Ena,In1,In2):
        self.Ena = Ena
        self.In1 = In1
        self.In2 = In2
        GPIO.setup(self.Ena,GPIO.OUT)
        GPIO.setup(self.In1,GPIO.OUT)
        GPIO.setup(self.In2,GPIO.OUT)
        self.pwm = GPIO.PWM(self.Ena,100) # operates at 100 Hz (check?)
        self.pwm.start(0)

    def moveF(self,x=100,t=0): # by default, it operates at 50% speed
        GPIO.output(self.In1,GPIO.LOW)
        GPIO.output(self.In2,GPIO.HIGH)
        self.pwm.ChangeDutyCycle(x) # x% of the speed
        sleep(t)

    def moveB(self,x=100,t=0): # by default, it operates at 50% speed
        GPIO.output(self.In1,GPIO.HIGH)
        GPIO.output(self.In2,GPIO.LOW)
        self.pwm.ChangeDutyCycle(x) # x% of the speed
        sleep(t)

    def stop(self,t=0):
        GPIO.output(self.In1,GPIO.LOW)
        GPIO.output(self.In2,GPIO.LOW)
        self.pwm.ChangeDutyCycle(0)
        sleep(t)

class Servo:

    def __init__(self, pwm_pin, frequency=50):
        # Initialize GPIO settings
        #GPIO.setmode(GPIO.BCM)
        self.pwm_pin = pwm_pin
        GPIO.setup(self.pwm_pin, GPIO.OUT)

        # Initialize PWM settings
        self.pwm = GPIO.PWM(self.pwm_pin, frequency)
        self.pwm.start(0)

    def set_pwm(self,pwm):
        GPIO.output(self.pwm_pin, True)
        self.pwm.ChangeDutyCycle(pwm)
        sleep(1)

    def cleanup(self):
        # Stop PWM and clean up GPIO
        self.pwm.stop()
        GPIO.cleanup()
```

```
class DistanceSensor:
    def __init__(self, trig_pin, echo_pin):
        self.trig_pin = trig_pin
        self.echo_pin = echo_pin
        GPIO.setup(self.trig_pin, GPIO.OUT)
        GPIO.setup(self.echo_pin, GPIO.IN)

    def get_distance(self):
        # Ensure trigger pin is low
        GPIO.output(self.trig_pin, GPIO.LOW)
        time.sleep(0.000002)

        # Create a 10us pulse on the trigger pin
        GPIO.output(self.trig_pin, GPIO.HIGH)
        time.sleep(0.00001)
        GPIO.output(self.trig_pin, GPIO.LOW)

        # Listen for echo
        while GPIO.input(self.echo_pin) == 0:
            pass
        echo_start_time = time.time()
        while GPIO.input(self.echo_pin) == 1:
            pass
        echo_stop_time = time.time()

        # Calculate distance
        ping_travel_time = echo_stop_time - echo_start_time
        echo_travel_distance = ping_travel_time * 34300
        distance = echo_travel_distance / 2
        return round(distance, 1)

    def continuous_detection(self, delay=0.1):
        try:
            while True:
                distance = self.get_distance()
                print(f'Distance: {distance} cm')
                time.sleep(delay)
        except KeyboardInterrupt:
            GPIO.cleanup()
            print('GPIO CleanCed up and Good to Go')

class Car():
    def __init__(self, motor_left_pins, \
    motor_right_pins,sensor_left_pins,left_servo,right_servo):
        self.motor_left = Motor(motor_left_pins[0], \
            motor_left_pins[1], motor_left_pins[2])
        self.motor_right = Motor(motor_right_pins[0], \
            motor_right_pins[1], motor_right_pins[2])
        self.sensor_left = DistanceSensor(sensor_left_pins[0], sensor_left_pins[1])
        self.servo_left = Servo(left_servo,50)
        self.servo_right = Servo(right_servo,50)

    def forward(self, speed=100, duration=0):
        self.motor_left.moveF(speed)
        self.motor_right.moveF(speed)
        if duration > 0:
            sleep(duration)
            self.stop()
```

```
    def backward(self, speed=100, duration=0):
        self.motor_right.moveB(speed)
        self.motor_left.moveB(speed)
        if duration > 0:
            sleep(duration)
            self.stop()

    def turn_left(self, speed=100, duration=0):
        self.motor_left.moveB(speed)  # Stop the left motor
        self.motor_right.moveF(speed)  # Move the right motor forward
        if duration > 0:
            sleep(duration)
            self.stop()

    def turn_right(self, speed=100, duration=0):
        self.motor_right.moveB(speed)  # Stop the right motor
        self.motor_left.moveF(speed)  # Move the left motor forward
        if duration > 0:
            sleep(duration)
            self.stop()

    def stop(self, duration=0):
        self.motor_left.stop(duration)
        self.motor_right.stop(duration)
        if duration > 0:
            sleep(duration)
    def auto_straight(self,dis):
    # front_dis = self.sensor_front.get_distance()
        left_dis = self.sensor_left.get_distance()
        sleep(0.3)
        print(f"left_dis:{left_dis}")
        if(left_dis-dis>1):
            self.motor_left.moveF(0)
            self.motor_right.moveF(0)
            print("left")
        elif(left_dis-dis<-1):
            self.motor_left.moveF(0)
            self.motor_right.moveF(0)
            print("right")
        else:
            self.motor_left.moveF(100)
            self.motor_right.moveF(100)
            print("front")
        sleep(0.2)

    def Wheels_open(self):
        self.servo_left.set_pwm(5)
        self.servo_right.set_pwm(3)

    def Wheels_closed(self):
        self.servo_left.set_pwm(3)
        self.servo_right.set_pwm(5)
```

```
def main():
    #motor_left_pins = (16, 20, 21)
    # Define the GPIO pins for the motors, sensors, and servos
    motor_left_pins = (16, 20, 21)
    motor_right_pins = (22, 5, 6)

    sensor_left_pins = (19, 26)

    left_servo_pin = 17  # GPIO pin for the left servo
    right_servo_pin = 27  # GPIO pin for the right servo
    sensor = 24
    sensor_2 = 25
    GPIO.setup(sensor, GPIO.IN)#sensor related
    GPIO.setup(sensor_2, GPIO.IN)
    # Initialize the Car
    my_car = Car(motor_left_pins, motor_right_pins, \
        sensor_left_pins, left_servo_pin, right_servo_pin)
    my_car.stop(duration = 0.5)
    buttom_pin = 23
    GPIO.setup(buttom_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    kn=0

    try:
        while True:
            EXIT = False
            print("Loop")
            kn=kn+1
            print(kn)
            if (GPIO.input(buttom_pin) == GPIO.HIGH):
                print("1")
                while(GPIO.input(buttom_pin) == GPIO.HIGH):
                    pass
                distance = GPIO.input(sensor)
                print(f"Dis: {distance} cm")
                count=0
                while (distance == 1):
                    #distance = GPIO.input(sensor)
                    #print(f"Dis: {distance} cm")
                    #my_car.auto_straight(left_dis_init)
                    distance = GPIO.input(sensor)
                    my_car.forward(speed=100, duration =0.5)
                    count=count+0.5
                    print(distance)
                    print(count)
                    if (GPIO.input(buttom_pin) == GPIO.HIGH):
                        EXIT = True
                        break
                if(EXIT==True):
                    break
                print("count=")
                print(count)
                mode=0
                if(count<18):
                    mode=1
                elif(count>33):
                    mode=3
                else:
                    mode=2
```

```
print("Opening Wheels...")
sleep(2)
my_car.stop(duration = 0.2)
my_car.Wheels_open()
sleep(3)
distance_2 = GPIO.input(sensor_2)
while (distance_2 == 1):
    distance_2 = GPIO.input(sensor_2)
    print(f"Dis: {distance_2} cm")
    my_car.forward(speed=100, duration = 0.2)
    if (GPIO.input(buttom_pin) == GPIO.HIGH):
        EXIT=True
        break
if(EXIT==True):
    break
my_car.Wheels_closed()
if(GPIO.input(buttom_pin) == GPIO.HIGH):
    break
sleep(2)
direction = 0
picam2 = Picamera2()
dispW=1280
dispH=720
picam2.preview_configuration.main.size = (dispW,dispH)
picam2.preview_configuration.main.format = "RGB888"
picam2.preview_configuration.controls.FrameRate=30
picam2.preview_configuration.align()
picam2.configure("preview")
picam2.start()

# define variables
fps=0
pos=(30,60)
font=cv2.FONT_HERSHEY_SIMPLEX
height=1.5
weight=3
myColor=(0,0,255)
GhueLow = 51
GhueHigh = 91
GsatLow = 43
GsatHigh = 168
GvalLow = 45
GvalHigh = 236
RhueLow = 168
RhueHigh = 179
RsatLow = 150
RsatHigh = 255
RvalLow = 0
RvalHigh = 255

while direction == 0:
    if(GPIO.input(buttom_pin) == GPIO.HIGH):
        EXIT=True
        break
    tStart=time.time()

    # grab a frame from Pi camera
    frame= picam2.capture_array()
```

```
frameHSV=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

# add FPS text
cv2.putText(frame,str(int(fps))+' \
    FPS',pos,font,height,myColor,weight)

# define my masks
GlowerBound=np.array([GhueLow,GsatLow,GvalLow])
GupperBound=np.array([GhueHigh,GsatHigh,GvalHigh])
GmyMask=cv2.inRange(frameHSV,GlowerBound,GupperBound)

RlowerBound=np.array([RhueLow,RsatLow,RvalLow])
RupperBound=np.array([RhueHigh,RsatHigh,RvalHigh])
RmyMask=cv2.inRange(frameHSV,RlowerBound,RupperBound)

# define my objects
GmyObject=cv2.bitwise_and(frame,frame, mask=GmyMask)
RmyObject=cv2.bitwise_and(frame,frame, mask=RmyMask)

# get the contour
Gcontours,junk=cv2.findContours(GmyMask, \
    cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
Rcontours,junk=cv2.findContours(RmyMask, \
    cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
if len(Gcontours)>0:
    # sort all contours that satisfy our HSV requirement
    Gcontours=sorted(Gcontours,key=lambda \
        x:cv2.contourArea(x),reverse=True)

    # get the largest contour
    Gcontour=Gcontours[0]

    # define a rectangle that surrounds the largest contour
    x1,y1,w1,h1=cv2.boundingRect(Gcontour)
    cv2.rectangle(frame,(x1,y1),(x1+w1,y1+h1),(0,255,0),3)

    if len(Rcontours)>0:
        # sort all contours that satisfy our HSV requirement
        Rcontours=sorted(Rcontours,key=lambda \
            x:cv2.contourArea(x),reverse=True)

        # get the largest contour
        Rcontour=Rcontours[0]

        # define a rectangle that surrounds the largest contour
        x2,y2,w2,h2=cv2.boundingRect(Rcontour)
        cv2.rectangle(frame,(x2,y2),(x2+w2,y2+h2),(0,0,255),3)
        if x2+w2/2>x1+w1/2:
            print('Green is on the left')
            direction =1
        if x2+w2/2<x1+w1/2:
            print('Green is on the right')
            direction =2
print()
cv2.imshow("Camera", frame)

# exit the program when key 'q' is pressed
if cv2.waitKey(1)==ord('q'):
```

```
                        break
                    # stop timing
                    tEnd=time.time()

                    # calculate frames per second
                    loopTime=tEnd-tStart
                    fps=.9*fps + .1*(1/loopTime)

                    print("direction value is:")
                    print(direction)
                    my_car.forward()# moving
                cv2.destroyAllWindows()

                if(EXIT==True):
                    break
                sleep(15)

                if(direction == 1):
                    my_car.turn_left(speed=100,duration=6.3)
                    print("turing left")
                    #try setting parameters here for testing
                    #mode=3
                    if(mode==1):
                        my_car.forward(speed =100, duration = 51)
                    elif(mode==2):
                        my_car.forward(speed =100, duration = 43)
                    else:
                        my_car.forward(speed =100, duration = 34)
                    my_car.turn_left(speed=100,duration=6)
                    print("turing left")
                    my_car.forward(speed =100, duration = 22)
                elif(direction == 2):
                    my_car.turn_right(speed=100,duration=6.9)
                    print("turning right")
                    #try setting parameters here
                    #mode=1
                    if(mode==1):
                        my_car.forward(speed =100, duration = 53)
                        my_car.turn_right(speed=100,duration=6.6)
                    elif(mode==2):
                        my_car.forward(speed =100, duration = 44)
                        my_car.turn_right(speed=100,duration=6.7)
                    else:
                        my_car.forward(speed =100, duration = 34)
                        my_car.turn_right(speed=100,duration=6.9)
                    print("turning right")
                    my_car.forward(speed =100, duration = 26)
                my_car.stop(duration=0.5)
                break
            sleep(0.5)
    except KeyboardInterrupt:
        GPIO.cleanup()
        print('GPIO Good to Go')
    finally:
        GPIO.cleanup()

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