
SunFounder GalaxyRVR Kit for Arduino

Release 1.0

www.sunfounder.com

Feb 27, 2024

CONTENTS

1	Assemble Video	3
2	Play Mode	5
2.1	Quick Guide	6
2.2	Avoid(E)	10
2.3	Follow(F)	10
2.4	STT(J)	10
3	Course Mode	13
3.1	Lesson 1 Unveiling the Mars Rover	13
3.2	Lesson 2 Understanding and Making Rocker-Bogie System	19
3.3	Lesson 3: Entering the World of Arduino and Coding	22
3.4	Lesson 4: Mastering the TT Motor	32
3.5	Lesson 5: Unleashing Mars Rover Mobility	46
3.6	Lesson 6: Exploring the Obstacle Avoidance Module	50
3.7	Lesson 7: Enhancing Rover Navigation with Ultrasonic Module	59
3.8	Lesson 8 Advanced Obstacle Avoidance and Intelligent Following System	66
3.9	Lesson 9: Lighting the Way with RGB LED Strips	70
3.10	Lesson 10: Exploring the Mars Rover Visual System - Servo and Tilt Mechanism	75
3.11	Lesson 11: Exploring the Mars Rover Visual System - Camera and Real-time Control	82
3.12	Lesson 12: Driving the Rover with the App	94
3.13	Lesson 13: Investigating the Mars Rover Energy System	103
4	Hardware	111
4.1	SunFounder R3 Board	111
4.2	GalaxyRVR Shield	113
4.3	ESP32 CAM	119
4.4	Camera Adapter Board	121
4.5	Ultrasonic Module	127
4.6	IR Obstacle Avoidance Module	128
4.7	4 RGB LEDs Strip	132
4.8	Servo	134
4.9	TT Motor	136
4.10	Solar Panel	139
4.11	18650 Battery	140
5	FAQ	141
5.1	Q1: Compilation error: SoftPWM.h: No such file or directory	141
5.2	Q2: avrdude: stk500_getsync() attempt 10 of 10: not in sync: resp=0x6e?	141
5.3	Q3: How can I use the STT mode on my Android device?	143

5.4	Q4: About the ESP32 CAM Firmware	145
5.5	Q5: How to Flash New Firmware to an ESP32 CAM?	145

Thanks for choosing our GalaxyRVR.

Note: This document is available in the following languages.

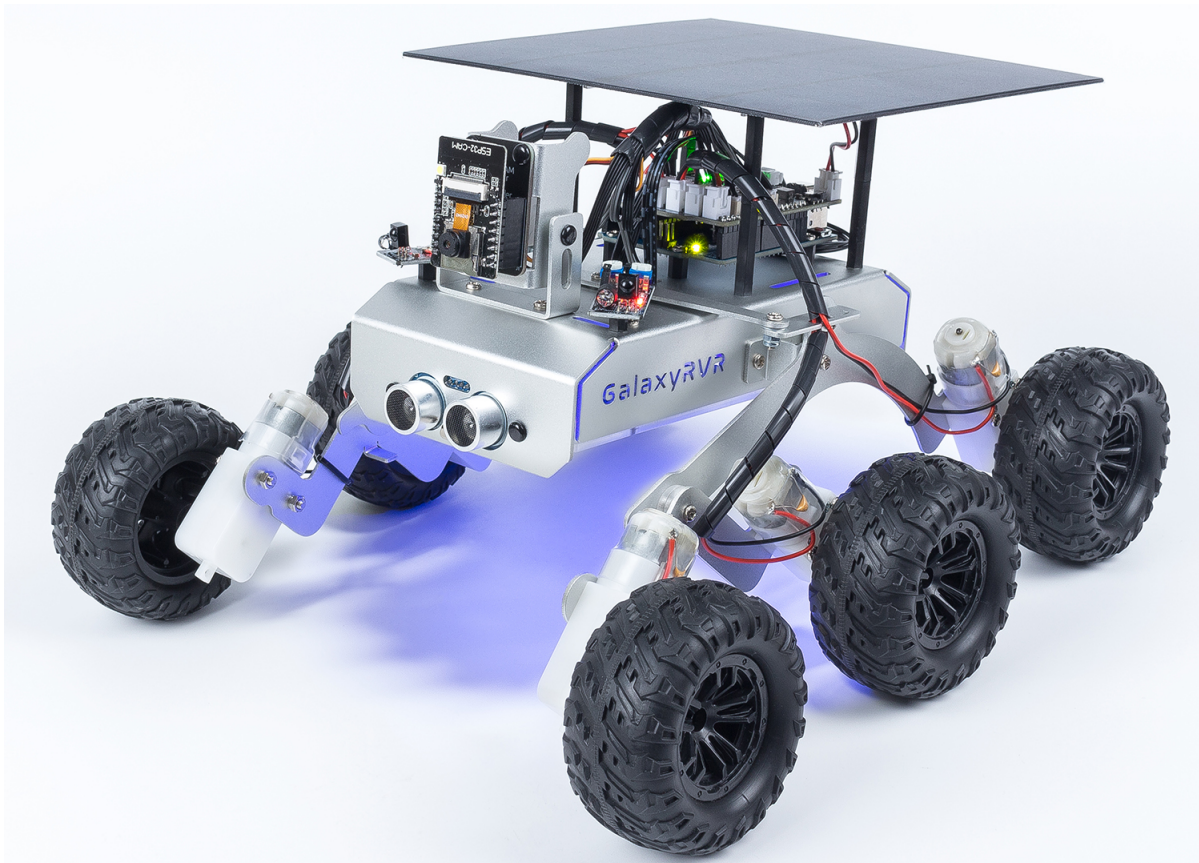
-
-
-

Please click on the respective links to access the document in your preferred language.

Imagine piloting a rover on the desolate terrain of Mars, exploring alien landscapes and confronting the unknown. Sounds like a dream for NASA engineers, right?

Not anymore.

Welcome to the world of the GalaxyRVR, a Mars Rover kit from SunFounder, designed to bring interplanetary exploration right into your living room.



Built upon the same universal rocker-bogie system employed by NASA's real Mars rovers, GalaxyRVR is capable of traversing all kinds of terrains – be it rocky mountains, sandy beaches, or grassy fields. The suspension system ensures smooth mobility, making the alien world of Mars feel just like home.

Equipped with a high-definition camera module, GalaxyRVR offers a first-person view, providing an immersive piloting experience as if you're truly navigating the Red Planet. Its advanced obstacle avoidance module and ultrasonic module ensure it can dodge obstacles, embodying a spirit of autonomy and resilience.

Delve into the Cosmos: The Courses

- *Play Mode*: Want a quick start? Play Mode is designed for those who can't wait to start their Martian journey. With factory-preloaded code, simply assemble your GalaxyRVR and control it using the convenient app to explore its multitude of functions.
- *Course Mode*: Intrigued by the underlying technology? Course Mode is for those who want to delve deeper. Understand the principles behind the GalaxyRVR's design, learn to code, and empower your rover with a variety of exciting features.

This documentation is your guide to exploring the intricacies of GalaxyRVR. It includes detailed assembly instructions, programming guides, and insights into the working principles of a Mars Rover, all aimed to stimulate your curiosity and foster creativity.

The GalaxyRVR isn't just a toy, but a gateway to Mars, an educational tool, and a catalyst for limitless imagination. Start your cosmic journey today with GalaxyRVR, and let your world become an extension of Mars.

ASSEMBLE VIDEO

Overview

If you want to learn while assembling the various components, you can refer to *Course Mode*. In Course Mode, each lesson comes with assembly videos, component principles, and related test code, allowing you to learn as you go. So, you can skip this chapter.

If you want to quickly assemble and start playing, you can watch the assembly video first. After completing the assembly, proceed to *Play Mode* to learn about the app installation and connection. Then, you can use the app to control the GalaxyRVR.

Videos

- For the assembly of the GalaxyRVR, we provide printed assembly instructions in the kit. Here is the PDF version of it:
 - Component List and Assembly Instructions
- If you are having difficulty understanding the printed assembly instructions, you can instead follow our step-by-step assembly video.

Note:

- The assembly videos are in a series of 7.
 - You can check out other videos from the playlist in the upper-right corner, or let it play automatically.
-

Tips

- When inserting the ESP32 CAM into the Camera Adapter, be aware of its orientation. It should align properly with the ESP32 Adapter.



- After assembly, you can secure the motor wires to the plate using cable ties to prevent them from getting tangled in the wheels.

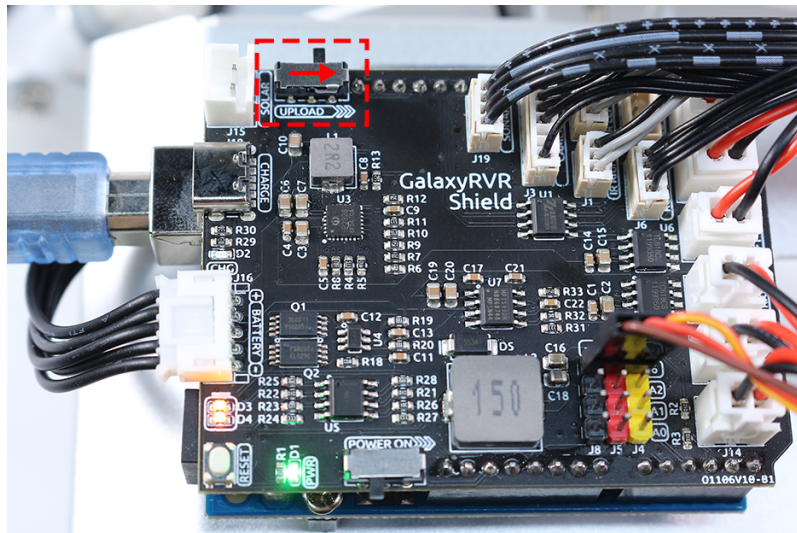
PLAY MODE

Are you eager to kick-start your Martian journey right away? Our Play Mode is perfectly tailored for those of you who just can't wait to dive in. Equipped with factory-preloaded code, you can jump straight into the action after assembling your GalaxyRVR. Use the user-friendly SunFounder Controller app to explore its myriad of functions including first-person driving, switching between obstacle avoidance and follow modes, and even voice control.

But before you set off to explore the Red Planet, let's make sure you are fully equipped and ready for the adventure. Below is a Quick Guide to assist you in this exciting journey!

Note:

- If your R3 board has already been uploaded with other code, but you want to continue using Play Mode, you will need to download the relevant code.
 - GalaxyRVR Codes
- Then, open the `galaxy-rvr.ino` file located in the `galaxy-rvr-main\galaxy-rvr` directory.
- Move the switch to the right and then click **Upload**.
- *Q1: Compilation error: SoftPWM.h: No such file or directory*



2.1 Quick Guide

1. Let's start the GalaxyRVR.

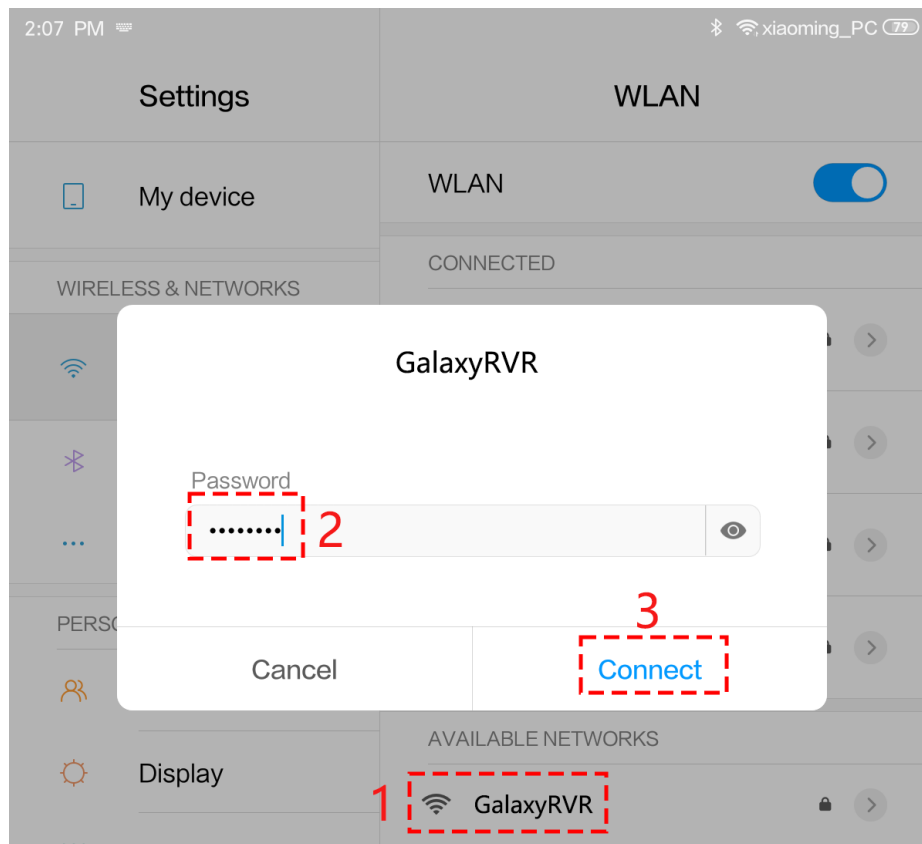
- The first time you use your GalaxyRVR, it is recommended that you plug in a Type-C USB cable to fully charge the battery first. Then turn the power on.
- To activate the ESP32 CAM, move the mode switch to the **Run** position, and press the **reset** button to reboot the R3 board. You will then observe a cyan light flashing on the bottom strip.

2. Install [SunFounder Controller](#) from **APP Store(iOS)** or **Google Play(Android)**.

3. Connect to the GalaxyRVR Network.

For optimal communication between your mobile device and the Rover, you'll need to connect them to the same local area network (LAN) provided by GalaxyRVR.

- Find GalaxyRVR on the list of available networks on your mobile device (tablet or smartphone), enter the password 12345678, and connect to it.



- The default connection mode is **AP mode**. After you've connected, there might be a prompt warning that there is no internet access on this network. If so, choose "Stay connected".

Confirm

This WLAN network has no access to the internet. Other WLAN networks are available. Switch networks?

Stay connected

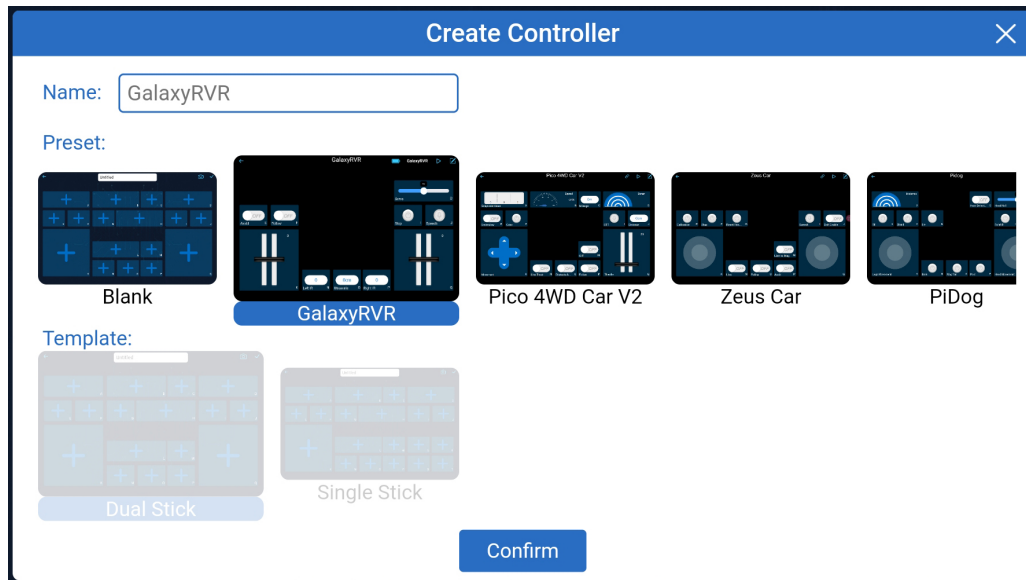
Switch networks

4. Set up a controller.

- To create a controller on SunFounder Controller, tap the + icon.




- Preset controllers are available for some products, here we choose **GalaxyRVR**. Give it a name, or simply tap **Confirm**.

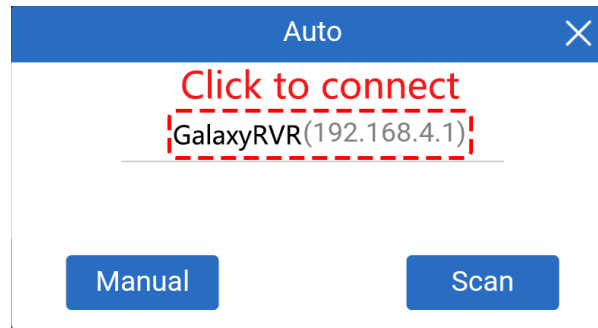



- Once inside, the app will automatically search for the GalaxyRVR. After a moment, you will see a prompt saying “Connected Successfully”.



Note:

- If not connected, please confirm that your Wi-Fi is connected to GalaxyRVR.
- You can also tap the  button to connect manually. After a short wait, you should see GalaxyRVR(IP) appear. Tap on it to establish a connection.



- Now, tap the  button enables you to view the live video feed from the camera and control the car using the provided widgets.



5. Let's now delve into the functions of each widget:

- **Servo(D)**: Controls the tilt mechanism of the Rover's camera, allowing you to observe a wider range.
- **Avoid(E)**: Switches the Rover into obstacle avoidance mode. The factory-set detection distances for each obstacle avoidance module may differ. Manual adjustment is required.
- **Follow(F)**: Toggles the Rover into follow mode. The factory-set detection distances for each obstacle avoidance module may differ. Manual adjustment is required.
- **Stop(I)**: Immediately halts all Rover movements.
- **STT(J)**: Press this widget to initiate voice recognition and make the Rover perform corresponding actions.
- **HeadLamp(M)**: To turn the LED on/off on the ESP32 CAM.
- **Throttle Widgets on K and Q**: The throttle widget in the K area controls the Rover's left motors, while the one in the Q area controls the right motors. Coordinating both widgets allows the GalaxyRVR to move in any direction.
- **Left IR(N)**: Displays the readings from the left obstacle avoidance module.
- **Ultrasonic(O)**: Shows the distance measured by the ultrasonic module.
- **Right IR(P)**: Displays the readings from the right obstacle avoidance module.

2.2 Avoid(E)

Tap the **Avoid(E)** widget to activate the obstacle avoidance mode.

Before enabling this mode, you may need to adjust the detection ranges of the sensors according to your current environment, as the factory settings may not be ideal for all situations.

If the detection range of the two infrared modules is too short, the Mars Rover might bump into obstacles. Conversely, if the range is too long, the Rover might start swerving too far away from an obstacle, potentially disrupting its navigation.

Here's how you can fine-tune the settings:

1. Start by adjusting the right obstacle avoidance module. During transportation, collisions may cause the transmitter and receiver on the infrared module to tilt. Therefore, you need to manually straighten them.
2. Place an obstacle about 20 cm directly in front of the right module. The box in which our Rover kit came is a good choice for this! Now, turn the potentiometer on the module until the indicator light on the module just lights up. Then, keep moving the obstacle back and forth to check if the indicator light comes on at the desired distance. If the light doesn't turn on at the correct distance or if it remains on without going out, you'll need to adjust the other potentiometer.
3. Repeat the same process for the other module.

2.3 Follow(F)

When you're ready to activate follow mode, simply tap the **Follow(F)** widget. If you haven't previously adjusted the detection distance of the obstacle avoidance modules, you will need to follow the steps in [Avoid\(E\)](#) first.

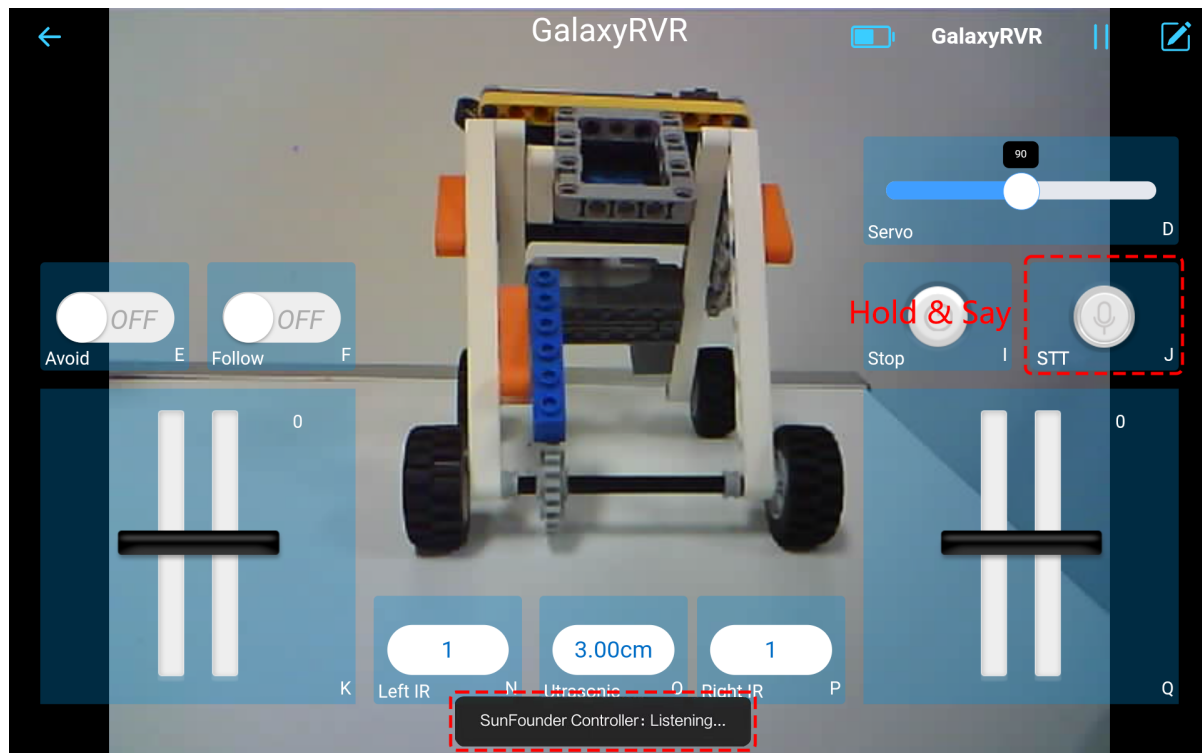
Once in follow mode, the GalaxyRVR will move towards an object in front of it or turn left or right to follow the object's movement.

2.4 STT(J)

STT stands for Speech to Text. The SunFounder Controller app integrates with your mobile device's voice recognition engine. Hence, when you tap and hold the **STT(J)** widget on the SunFounder Controller and speak into your mobile device,

Your device will capture your speech, convert it into text, and send it to the GalaxyRVR. If this text matches the pre-set commands in your code, the Rover will carry out the corresponding actions.

The following are the commands currently preset in the code. Speak any of the following commands and observe how the GalaxyRVR responds.



- stop: All movements of the rover can be stopped.
- forward: Let the rover move forward.
- backward: Let the rover move backward.
- left: Let the rover turn left.
- right: Let the rover turn right.

Note: The STT (Speech to Text) function requires an internet connection and Google services on Android devices. However, this doesn't work with the pre-set AP (Access Point) mode on the GalaxyRVR.

In AP mode, the GalaxyRVR creates a local Wi-Fi network that your mobile device can connect to, but it does not provide internet access.

To use the STT function on Android, switch the Rover's code from AP to STA mode as outlined in [Q3: How can I use the STT mode on my Android device?](#).

Note: iOS devices, using an offline voice recognition engine, work fine in both AP and STA modes.

COURSE MODE

Welcome to the Course Mode, a structured journey through 13 enlightening lessons. We kick off with a fascinating glimpse into the history of Mars rovers, setting the stage for our own rover assembly.

From the second lesson, we'll start getting hands-on. Each lesson spotlights a unique component crucial to our Mars rover. You'll not only understand its purpose and significance but also master its assembly and integration into the rover.

Whether you are a novice eager to dive into the world of Mars rovers or an experienced hobbyist looking to hone your skills, these lessons are crafted to provide a comprehensive understanding of the mechanics, electronics, and software that bring our GalaxyRVR to life. By the end of the course, you'll not only have assembled your very own Mars rover, but you'll also possess the knowledge to troubleshoot, modify, and further experiment with your rover.

So, buckle up for an exciting journey into the world of Mars rovers! Dive in and start exploring the red planet from the comfort of your home.

3.1 Lesson 1 Unveiling the Mars Rover

Welcome to Lesson 1: Understanding the Mars Rover. Today, we dive into the thrilling world of Mars rovers—our remote explorers on the Red Planet. We will learn about their evolution, their functions, and the technological marvels that they are. Furthermore, you'll channel your creativity to design your own rover and hone your presentation skills by explaining your unique design. Get ready to explore Mars from your classroom!

3.1.1 Learning Objectives

- Gain an understanding of the evolution and purpose of Mars rovers
- Express creativity by designing your own Mars rover
- Enhance presentation skills by sharing and explaining your rover design

3.1.2 Materials

- Mars Rover images and technical specifications for reference
- Documentary video on the history of Mars rovers
- Computer with internet access for research and viewing documentary
- Presentation slides or interactive whiteboard for lesson delivery
- Drawing paper, pencils, and coloring materials for rover design activity
- Worksheets for guided note-taking, reflection, and design planning

3.1.3 Steps

Step 1: What are Mars Rovers?

Before we dive into Mars rovers, let's first acquaint ourselves with Mars itself. As we can see from the images and models, the surface of Mars is marked with craters, mountains, valleys, and dust storms, painting a picture of a landscape that is both fascinating and challenging.



Can you imagine what it would be like to navigate through such a rugged terrain? Now, suppose you have the task of designing a rover for Mars.

- What considerations will you keep in mind given the terrain and conditions of Mars?
- What features will you equip it with to ensure it can perform its functions effectively?
- What tasks do you envision your Mars rover would need to accomplish?

Remember, a Mars rover is a robot designed to explore Mars, study its environment, and send data back to Earth. So think about aspects such as movement, communication, power supply, scientific research capabilities, and durability under Mars' extreme conditions.

Let's take a moment to brainstorm and share our ideas. It's interesting to think like engineers and scientists, isn't it?

We'll delve deeper into actual Mars rover designs and their functions in the following steps, so keep your creative ideas in mind as we progress.

Step 2: Exploring the History of Mars Rovers

Next, we'll embark on a journey through time by watching a documentary that details the history of Mars rovers. The documentary takes us from the first attempt at deploying a rover on Mars, the Soviet Mars 3 rover which unfortunately didn't succeed upon landing in 1971, to NASA's first successful Mars rover, Sojourner, in 1997.

Our journey doesn't stop there, as we venture further to understand the adventures of the most advanced rovers yet: Spirit, Opportunity, Curiosity, and Perseverance.

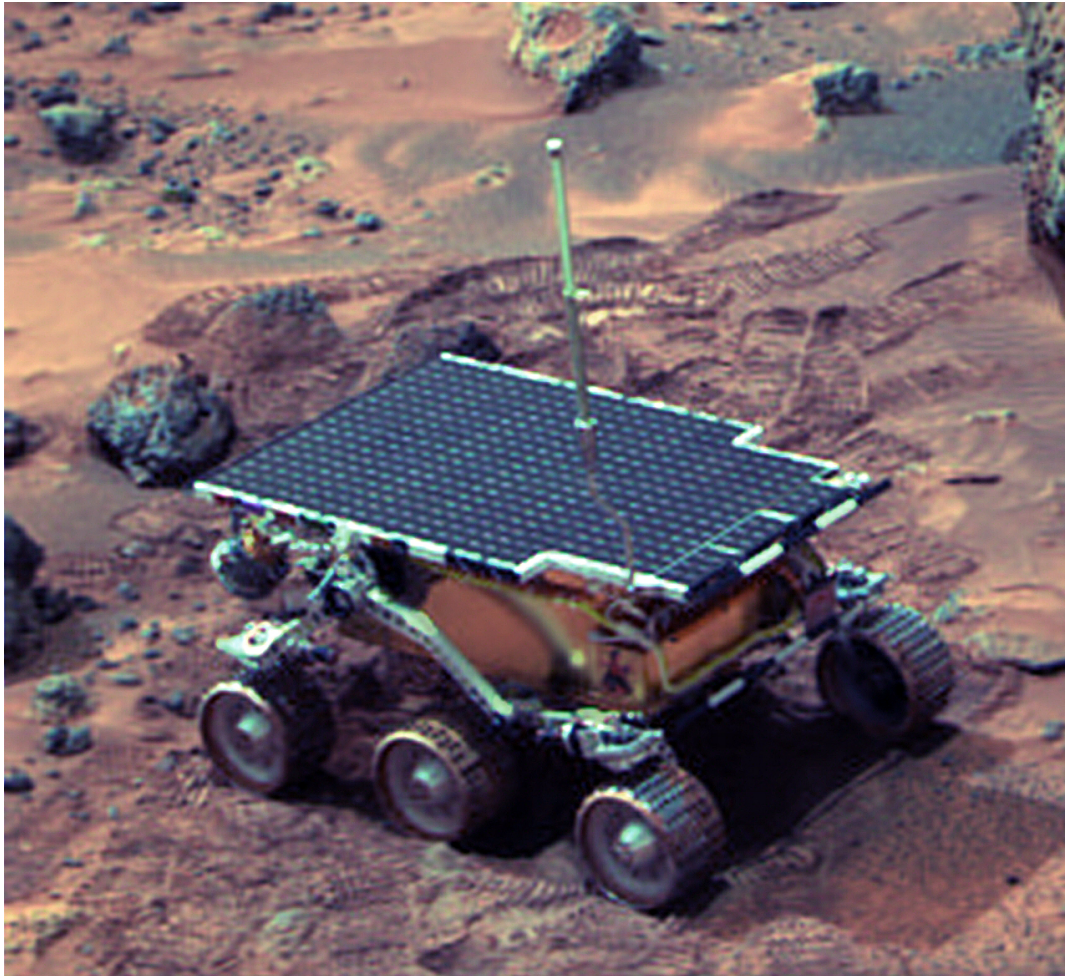
This documentary not only presents a historical context but also provides a comprehensive understanding of the progressive scientific and engineering milestones that have led to the current Mars exploration era.

Step 3: Summarize the Mars Rovers

After watching the documentary, let's summarize the different Mars rovers that have been sent on the red planet.

- **Sojourner** (1997)

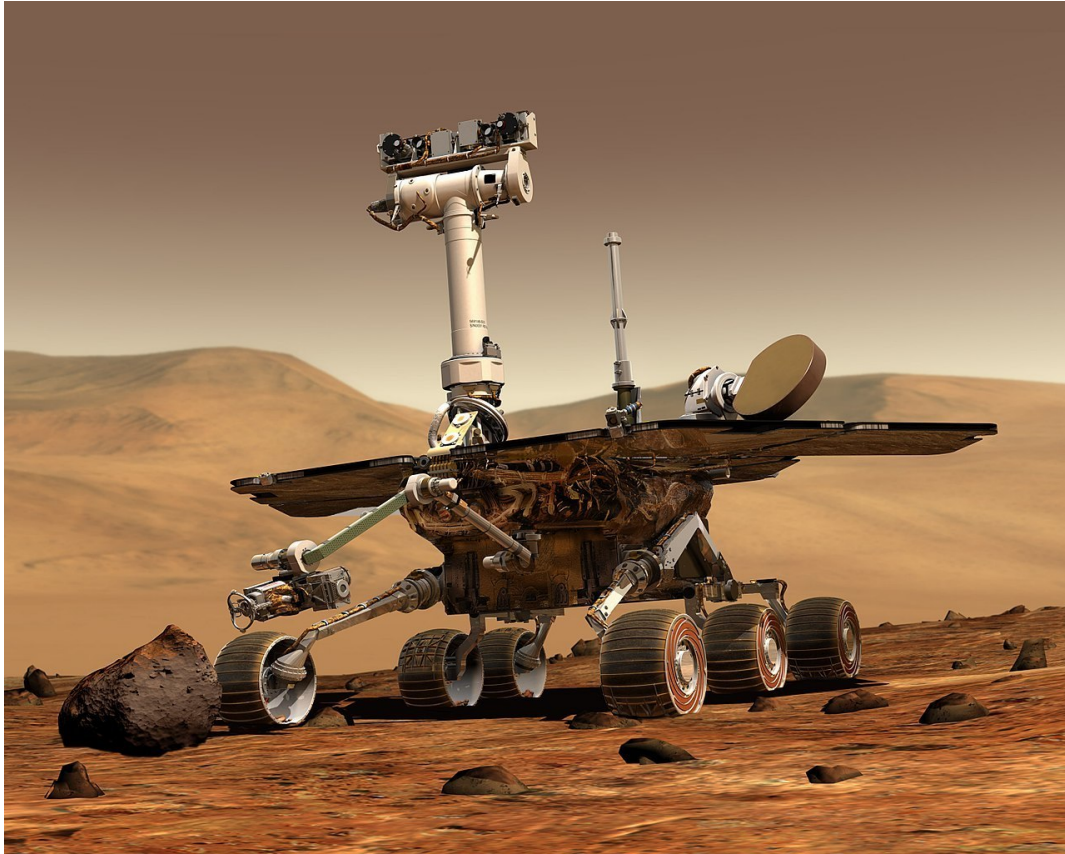
Sojourner, the pioneer of Mars rovers, embarked on its journey as a part of the Mars Pathfinder mission. It made a successful landing in the Ares Vallis region on July 4, 1997. As the first wheeled vehicle to roam on a planet other than Earth, Sojourner marked a significant milestone in Martian exploration. Although it was operational on Mars for only 92 Martian days, or sols, it set the groundwork for future exploratory rovers.



- **Spirit** (2004–2010) and **Opportunity** (2004–2018)

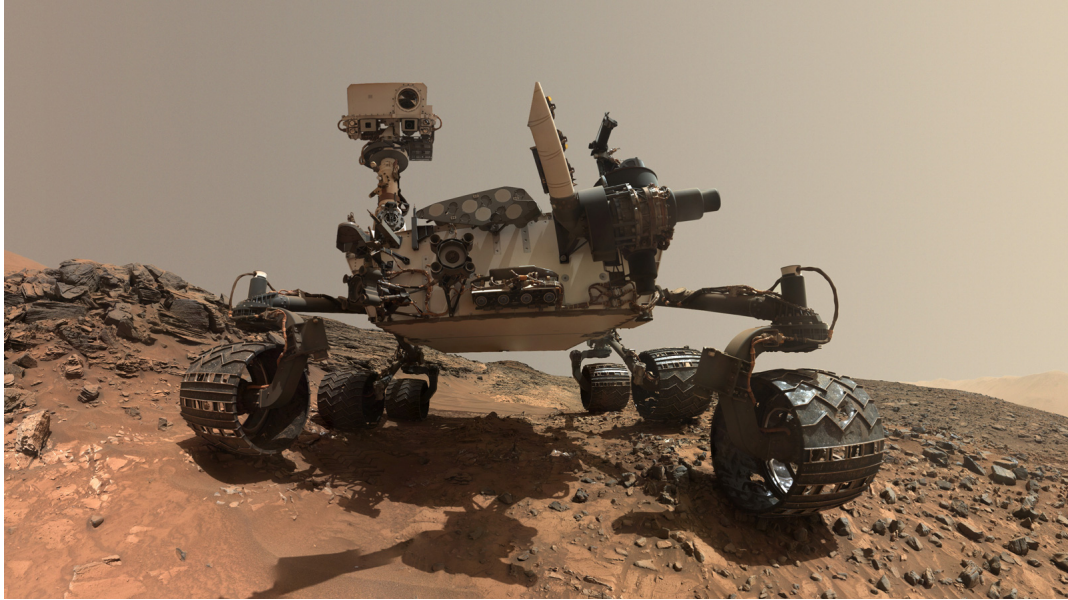
Spirit and **Opportunity** are twin rovers of NASA's Mars Exploration Rover (MER) mission. **Spirit**, also known as MER-A, operated on Mars from 2004 to 2010.

On the other hand, **Opportunity**, or MER-B, had a remarkably long run from 2004 to 2018. Together, they greatly expanded our understanding of the Martian surface and geological history.



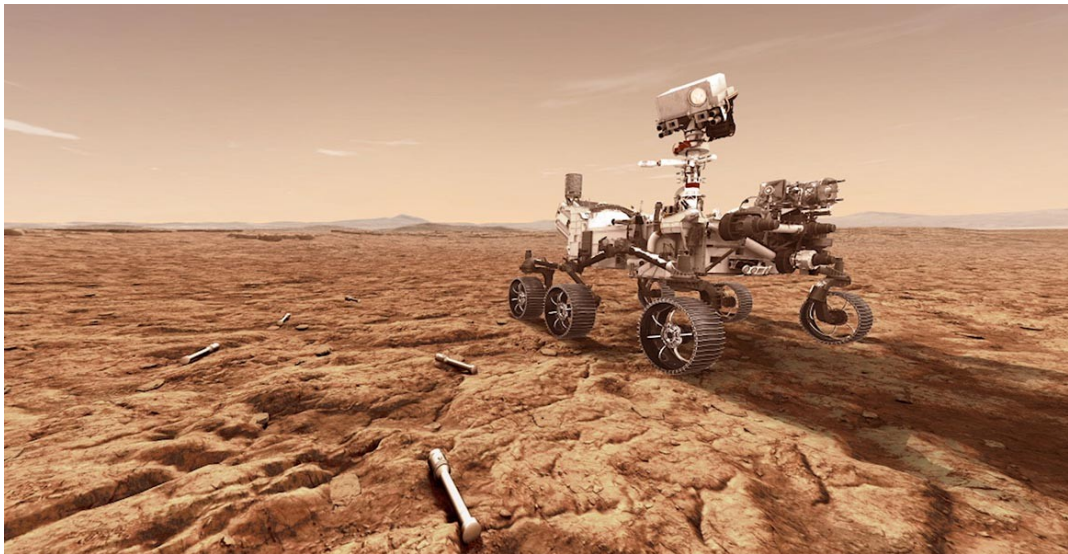
- **Curiosity** (2012–present):

Curiosity, a car-sized Mars rover, was designed to explore the Gale crater on Mars as part of NASA's Mars Science Laboratory (MSL) mission. Since its arrival in 2012, **Curiosity** has made numerous significant discoveries, including evidence of past liquid water on Mars.



- Perseverance (2021–present):

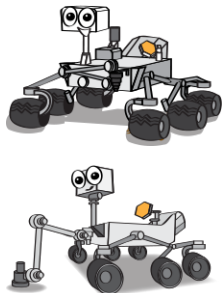
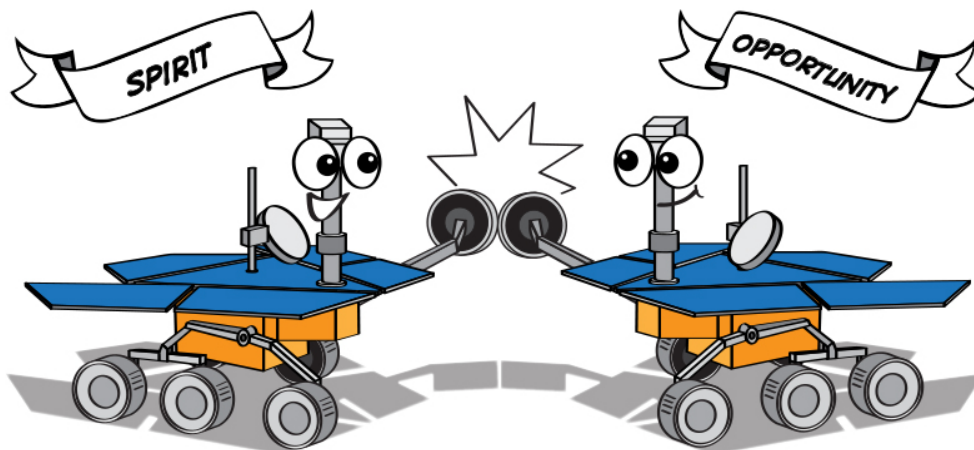
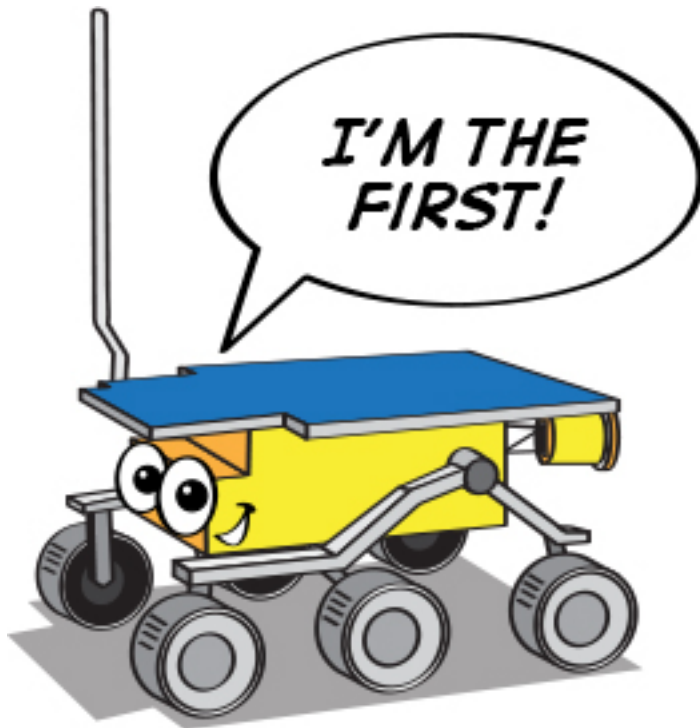
Perseverance, also known as Percy, is the most recent rover to arrive on Mars. It's designed to explore the Jezero crater as part of NASA's Mars 2020 mission. Along with its scientific instruments, **Perseverance** also carries Ingenuity, a small experimental Mars helicopter, marking another first in Martian exploration.



Now, let's have a discussion. Reflect on the evolution of these rovers.

- How do the designs of these rovers differ? How are they similar?
- How did the mission objectives influence the design of each rover?
- What advancements in technology can you identify between each rover?
- What features do you think the next Mars rover should have?
- Share your thoughts and reflections, as well as any questions you might have!

Step 4: Art Activity: Draw Your Own Mars Rover



For our next activity, let's put our knowledge and creativity to work by designing our very own Mars rover. Consider the key characteristics of the rovers we've studied so far, but also think about the unique attributes you would want to incorporate in your design.

Materials you'll need:

- Drawing paper
- Pencils and erasers
- Colored pencils, crayons, or markers

Drawing Instructions:

1. Start with the body of the rover. What shape will it be? How large?
2. Consider the wheels. How many will your rover have? What size and shape will they be?
3. Don't forget about the instruments. What scientific equipment will your rover carry? Cameras, drills, spectrometers, or something entirely new?
4. Lastly, consider any unique features. Does your rover have solar panels, or does it use a different power source? Can it communicate directly with Earth, or does it need a relay satellite?

Once everyone has completed their drawings, we'll share them with the class. Explain your design choices and the mission you envision for your rover.

Step 5: Present Your Mars Rover Designs

Now that everyone has completed their Mars Rover drawings, it's time to share them! As you present, discuss the thought process behind your design. What is your rover's mission? How does the design support this mission?

Remember, there are no wrong answers in this activity. The purpose is to stimulate your creativity and deepen your understanding of Mars rover technology.

Step 6: Reflection and Conclusion

As we conclude our Mars Rover lesson, let's take a few minutes to reflect on what we have learned. How do our rover designs reflect the advancements in technology and scientific objectives? How might the real Mars rovers continue to evolve in the future?

Remember, the exploration of space, like any STEAM field, is all about asking questions, solving problems, and using creativity. Keep exploring, keep asking questions, and keep being curious!

3.2 Lesson 2 Understanding and Making Rocker-Bogie System

In our previous lesson, we learned about the Mars rovers and their basic structure. One interesting aspect that we notice when looking at the evolution of Mars rovers is the consistency in their suspension system. Despite the advancement in technology, all the rovers from Sojourner to Perseverance have been designed using a similar type of suspension system known as the Rocker-Bogie system.

But why stick with the Rocker-Bogie system, you might wonder? What benefits does this particular design offer for Mars exploration?

In today's lesson, we're going to dig deeper into the science and engineering behind the Rocker-Bogie system, then build one.

Let's embark on this exciting engineering journey!

3.2.1 Learning Objectives

- Understand the design principle of the Rocker-Bogie suspension system and its advantages.
- Learn how to design and make a basic model of the Rocker-Bogie suspension system.
- Apply basic principles of physics to explain how the Rocker-Bogie suspension system overcomes complex terrains.

3.2.2 Materials

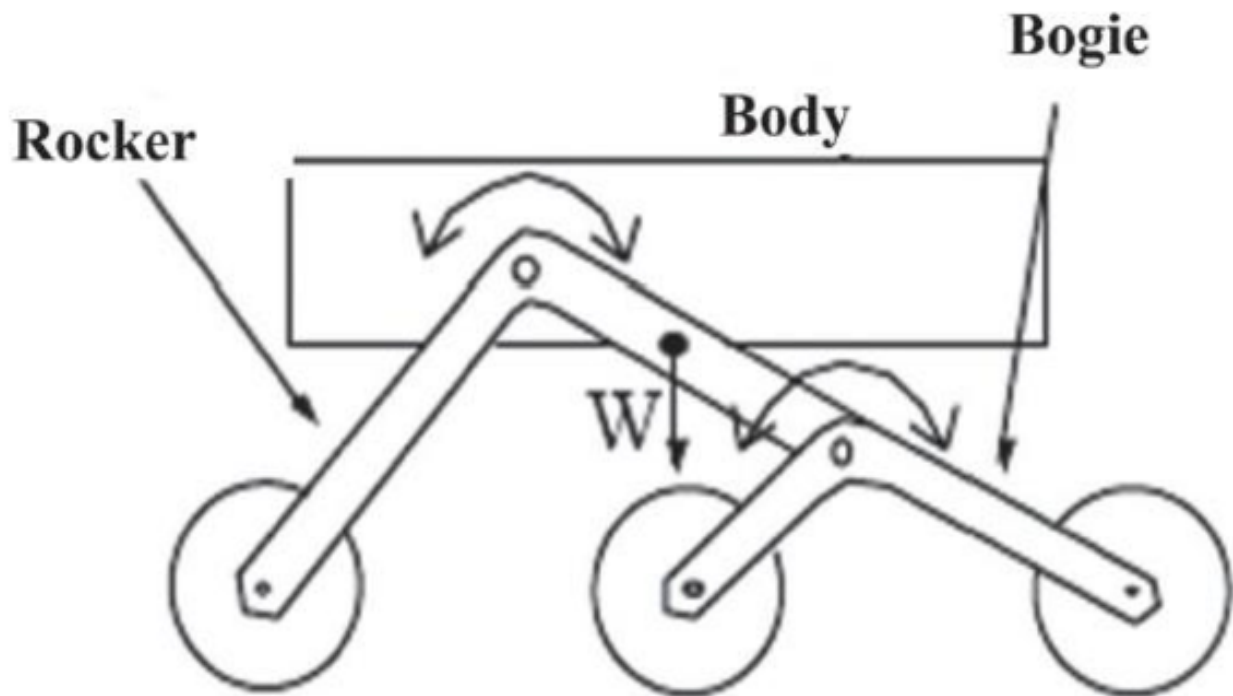
- Blueprints and reference materials (such as NASA Mars Rover design drawings and videos on how the Rocker-Bogie suspension system works)
- Mars Rover structure kit
- Basic tools and accessories (e.g. screwdriver, screws, etc.)

3.2.3 Steps

Step 1: Unraveling the Rocker-Bogie System

The Rocker-Bogie system is like a mountain goat of mechanics - designed to keep all wheels of the rover grounded while it navigates over rough and rocky terrains. It's specially built for handling Mars' unpredictable landscape, including steep inclines and sizable boulders. This system skips springs and instead leverages the geometry of its six wheels and their interactions to conquer tricky terrain. It's a shining example of how clever mechanical design can surmount environmental hurdles.

Let's dive into the two main parts of this system - the “rocker” and the “bogie”.



- The “rocker” part of the system is like the two large limbs on either side of the rover’s body. These limbs, or rockers, connect to each other and the rover’s body, or chassis, through a mechanism called a differential. Just like two legs walking, the rockers rotate in opposite directions relative to the chassis, making sure that most of

the wheels keep in contact with the ground. The body of the rover maintains the average angle of both rockers. One end of a rocker connects to a wheel, while the other end connects to the bogie.

- The “bogie” part of the system is like a mini-limbed creature attached to the rocker. It’s a smaller linkage system that pivots in the middle to the rocker and has a wheel at both ends.

With this basic understanding, let’s hop to the next step of our adventure.

Step 2: Seeing the System in Action

Below is a GIF that showcases the unique features of the Rocker-Bogie suspension system and illustrates how it enables Mars rovers to navigate the challenging Martian terrain.

After watching the gif, let’s have a discussion! Think about these questions:

- Why do you think the Rocker-Bogie suspension system is suitable for Mars exploration?
- Can you describe how the Rocker-Bogie system works in your own words?
- What are the key features of the Rocker-Bogie system that help the rovers to negotiate rough terrain?

Feel free to share your thoughts and insights about the Rocker-Bogie suspension system.

Step 3: Building it

Now that we’ve learned about the Rocker-Bogie system, it’s time to build our own.

Materials you need:

- GalaxyRVR Kit
- Basic tools like screwdriver and wrench
- Follow the steps provided in the assembly instructions of the GalaxyRVR Kit to construct the suspension system of the Rover.

Please note that patience and precision are essential here, make sure you correctly place every piece and secure it tightly.

In the meantime, discuss with your peers about the design and function of each component you are assembling. This will not only help in understanding the design but also its practical application in Mars exploration.

Remember, don’t worry if you encounter any issues during the assembly or testing. This is all part of the engineering process! Troubleshooting problems is how we learn and innovate.

Step 4: Summary and Reflection

During the assembly of the suspension system, did you notice that all the moving parts utilize self-locking nuts? Have you ever wondered why?

Self-locking nuts are a type of fastener that includes a rubber ring inside a regular nut. This design ensures that the assembled parts won’t easily loosen and fall off due to vibrations during movement.

In addition, it also ensures that the parts can rotate within a certain range.

So during assembly, you need to tighten the screw and self-locking nut with a socket and screwdriver first, then loosen it a bit. This ensures that there’s room for free rotation between the parts without them being too loose.

In this lesson, we not only learned about the Rocker-Bogie System but also built one ourselves. Furthermore, we can manually simulate how it allows the Mars Rover to move smoothly over various rough terrains.

Armed with this knowledge and experience, we are now better equipped to venture deeper into the unknown realms of Martian exploration. Let’s continue to unravel the mysteries of the red planet.

3.3 Lesson 3: Entering the World of Arduino and Coding

In our previous lesson, we successfully built the Rocker-Bogie Suspension System. However, to make it functional, we need to provide it with power, a control board, and programming to dictate its movements.

So in this lesson, we're going to get acquainted with the control board and the programming platform we'll be using.

3.3.1 Course Objectives

- Understand the basic concepts and functions of Arduino.
- Learn about SunFounder R3 Board.
- Install Arduino IDE and get familiar with its interface.
- Learn about basic syntax rules for Arduino programming.

3.3.2 Course Materials

- SunFounder R3 Board
- Arduino IDE
- USB Cable
- Computer

3.3.3 Steps

Step 1: Introduction to Arduino

You may have often heard the term “Arduino” in various contexts, but what exactly is it and why has it become so popular?

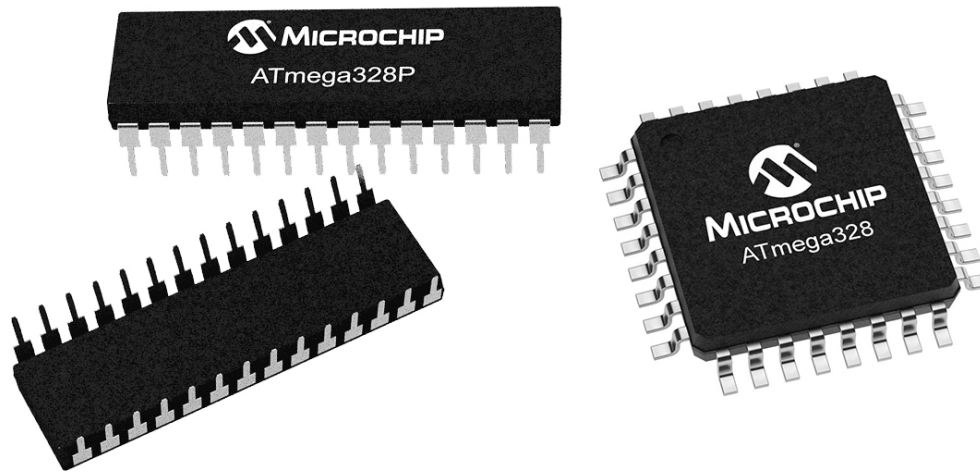
Arduino is an open-source electronics platform that is easy to use for both hardware and software applications. It's designed to make digital devices and interactive objects that can sense and control the physical world around them.

Sure, let's break it down:

- **Open-source:** Think of open-source like a community garden. Everyone can use it, everyone can contribute to it, and everyone can benefit from it. With open-source, both the designs of the physical parts (the hardware) and the programming instructions (the software) are shared freely. This means anyone can use them, improve them, or create their own versions. It's all about sharing and creativity!



- **Microcontroller:** A microcontroller is like the brain of the Arduino. It's a tiny computer that can run simple software. Although it's not as powerful as the computer you're using now, it's perfect for simple tasks like understanding messages from sensors or lighting up an LED (a small light).



- **Development Board:** Imagine the development board as the body that supports the brain. It's the board where the microcontroller sits and it contains other parts that help the microcontroller interact with the world. These parts include things like oscillators (that help with timing), voltage regulators (that control power levels), and connectors for power and data (like the plugs and switches in your house).

			
Arduino UNO R3	Arduino Mega 2560 Rev3	Arduino Leonardo	Arduino UNO Mini Limited Edition
			
Arduino Due	Arduino Micro	Arduino Zero	Arduino UNO WiFi Rev2

- **Arduino IDE:** This is like the teaching classroom for your Arduino. It's a program that runs on your computer where you can write the instructions that tell your Arduino what to do. These instructions are written in a programming language based on C++. Once you've written your instructions, you can send them to the Arduino board using a USB cable, just like handing in your homework!



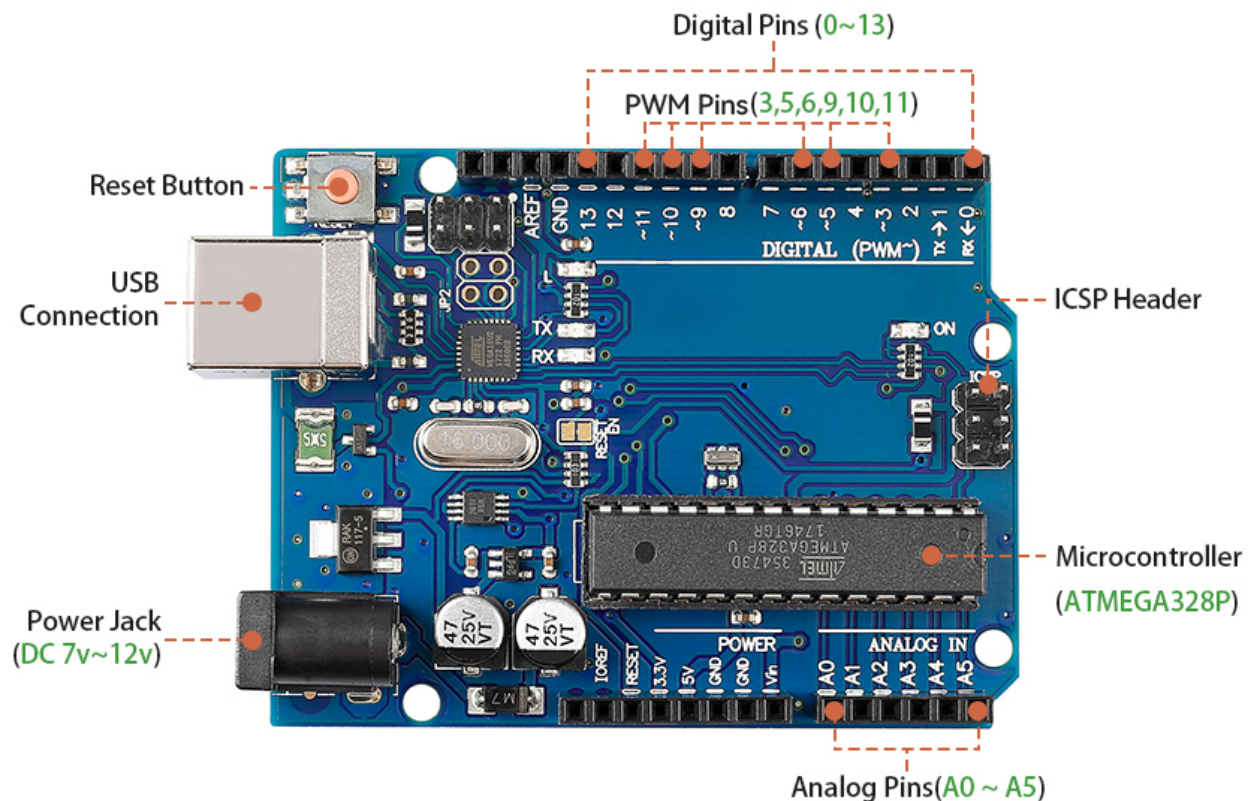
Now that you understand these basic ideas, you're well on your way to becoming an Arduino expert!

Then we'll dive into some hands-on activities to get you acquainted with Arduino programming and engineering principles. Get ready for an exciting learning journey!

Step 2: Getting to Know Your SunFounder R3 Board

Inside your kit, you'll find a blue board, seemingly a tiny city filled with small metallic towers and pathways. But don't let it intimidate you! This is the SunFounder R3 development board, a kind of Arduino board that can be used to program and control a vast array of electronic devices and projects.

Let's understand its key features in simple terms:



- **14 Digital Pins:** Think of these pins like little messengers. They can be programmed to send (output) or receive (input) simple “yes” or “no” messages to other parts of your Mars Rover. These messages are actually “on” or “off” signals that the board uses to control things like lights or motors.

- Six of these special pins can even send messages in a kind of secret code called PWM (Pulse Width Modulation). This code can be used to control how bright a light is, how fast a motor spins, or even where a moving part positions itself.
- **6 Analog Pins:** These pins are like the board's six special senses. They can read signals from different types of sensors (like a temperature sensor) and then translate these signals into a language that the board can understand and use in its programming.
- **USB Connection:** This is like the board's umbilical cord. You can use it to connect your board to your computer. This connection allows your computer to "teach" the board what to do by sending it a program you write.
- **Power Jack:** This is the board's food supply. You can connect a power supply, like a battery or an AC-to-DC adapter, to this jack to "feed" your board the electricity it needs to work.
- **ICSP Header:** This is like a special entrance for programming the board. It can be used if you have an external programmer (a special device for "teaching" the board).
- **Reset Button:** If you press it, it's like telling the board to forget what it was just doing and start its program over from the beginning.

With these basics, you'll be all set to begin your programming adventures with the SunFounder R3 board!

Step 3: Install Arduino IDE

Now that we understand what Arduino and the Arduino board are, it's time to start putting that knowledge to use. We're going to install the Arduino IDE, which is the software we'll use to program our Arduino board.

The latest version of the Arduino IDE is version 2.0. It's packed with features and is super user-friendly. However, you should know that it does have some system requirements:

- Windows - Win 10 and newer, 64 bits
- Linux - 64 bits
- Mac OS X - Version 10.14: "Mojave" or newer, 64 bits

To get started, follow these steps:

1. Visit and download the IDE for your OS version.

Arduino IDE 2.0.0

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

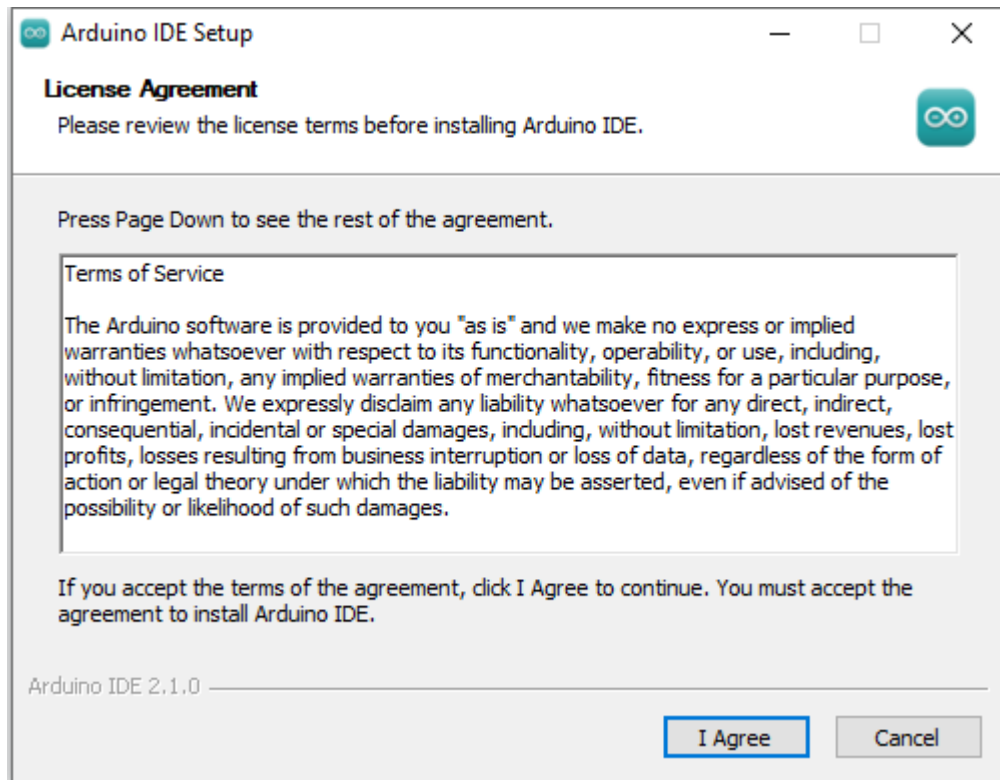
DOWNLOAD OPTIONS

- Windows** Win 10 and newer, 64 bits
- Windows** MSI installer
- Windows** ZIP file
- Linux** AppImage 64 bits (X86-64)
- Linux** ZIP file 64 bits (X86-64)
- macOS** 10.14: "Mojave" or newer, 64 bits

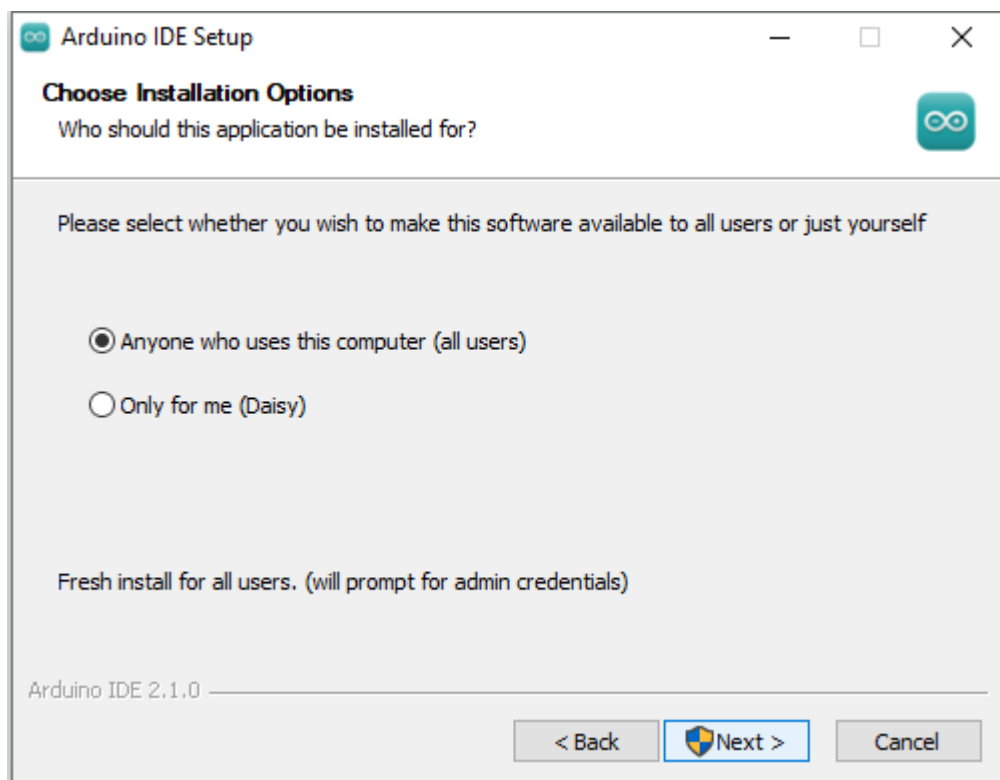
For Windows users:

1. Once you've downloaded the file (it will be called something like `arduino-ide_XXXX.exe`), double-click it to start the installation process.

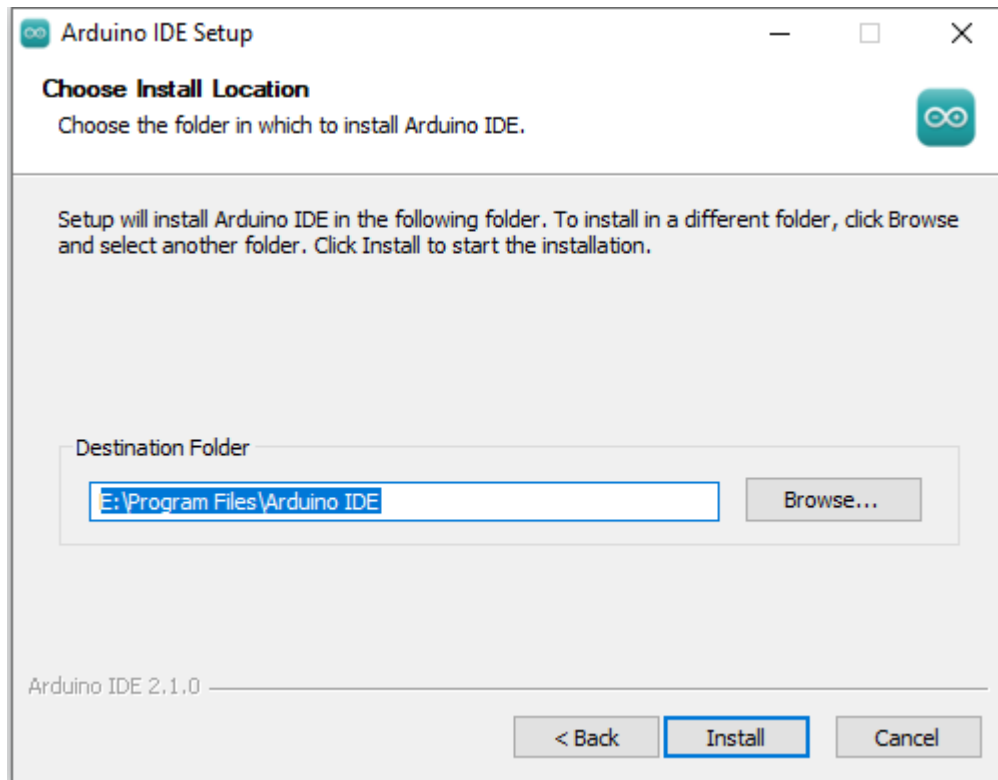
2. You'll be shown the **License Agreement**. Take a moment to read through this, and if you agree to the terms, click "I Agree".



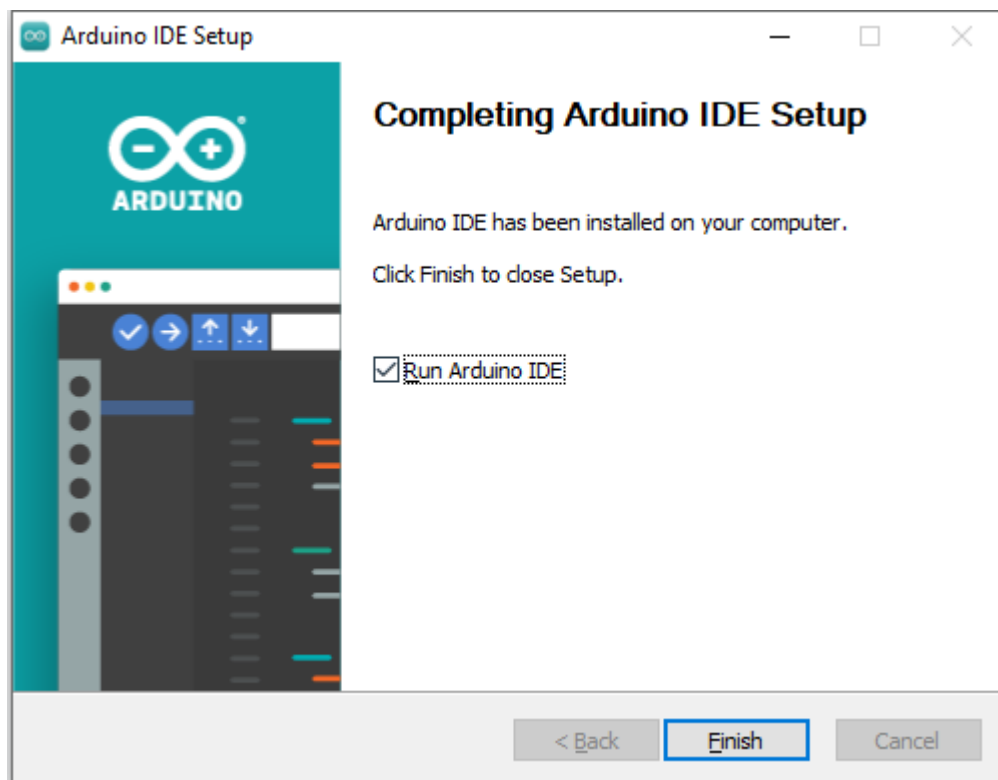
3. Next, you'll be asked to choose installation options. Leave these as they are and click "Next".



4. Choose where you want to install the software. It's generally best to install it on a different drive than the one your system uses.

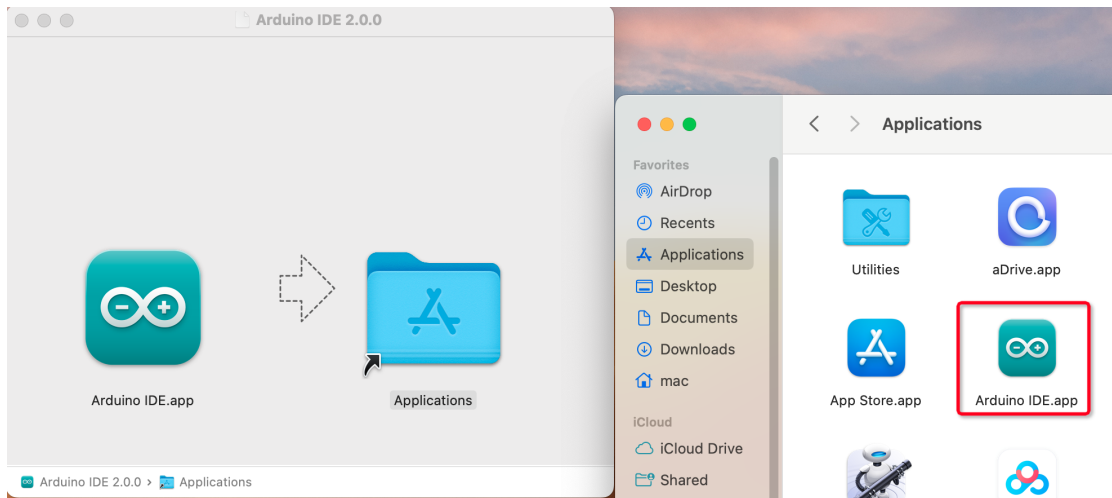


5. Click "Install" to start the installation. Once it's done, click "Finish".



For macOS users:

Double-click the downloaded file (which will be called something like `arduino_ide_XXXX.dmg`). Follow the on-screen instructions to drag the **Arduino IDE** app into the **Applications** folder. After a few seconds, the Arduino IDE will be successfully installed.

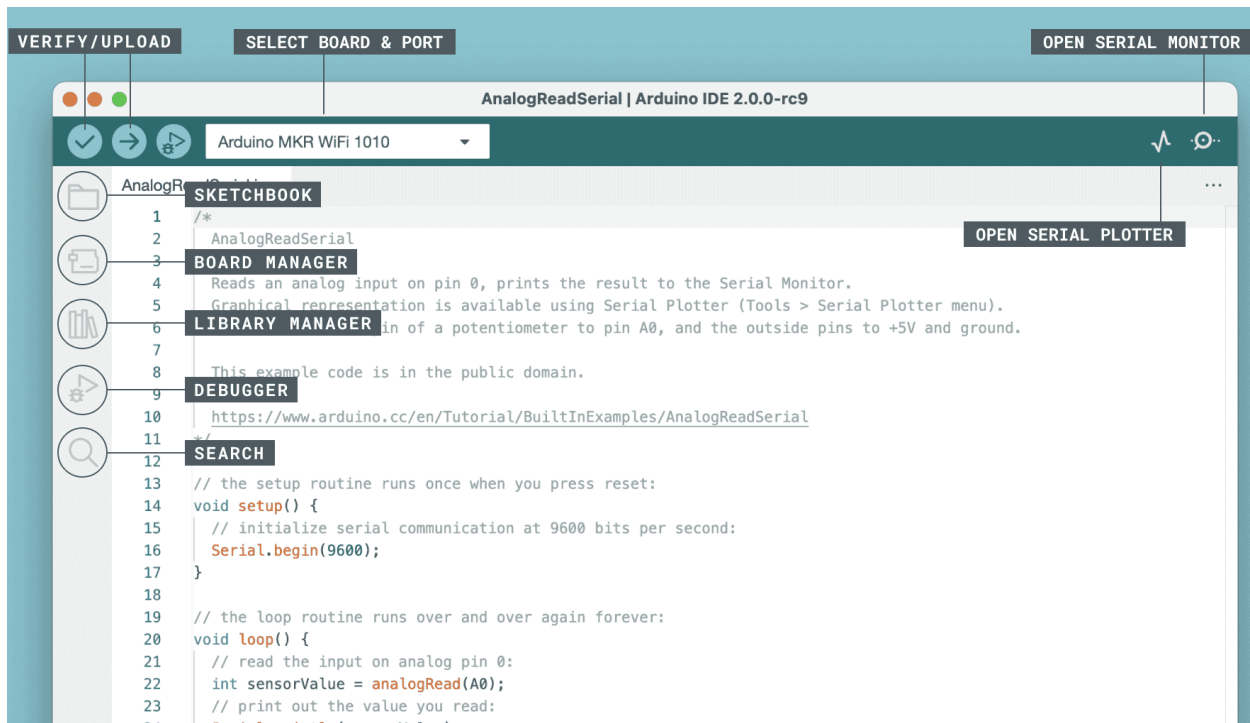


For Linux users:

You can find a detailed tutorial on installing the Arduino IDE 2.0 on a Linux system here: .

Step 4: Discovering the Arduino Playground (IDE)

Let's imagine together that the Arduino IDE is a magical playground filled with tools and gadgets waiting for us to explore and play with. Up next, I will guide you to understand every corner of this playground.



Here's what you'll find in your playground:

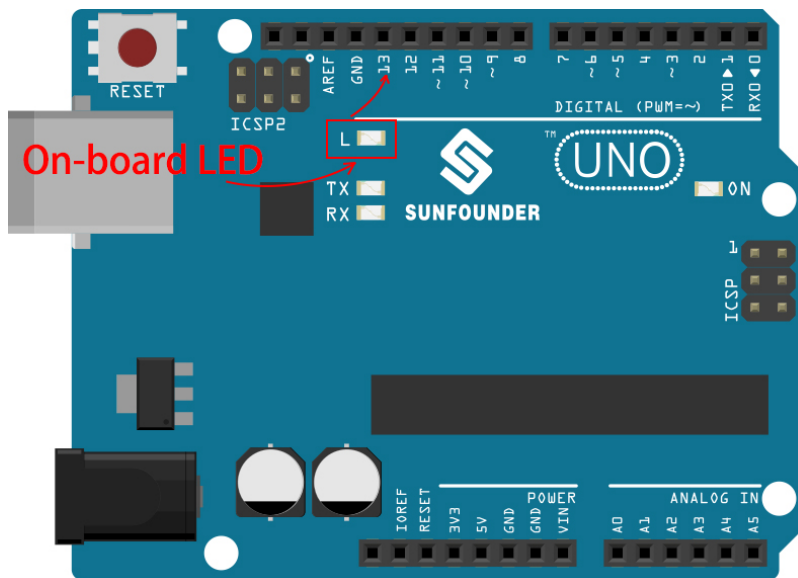
- **Verify / Upload** - Imagine this as your magic elevator. It takes the code you've written and whisks it up into your Arduino board.
- **Select Board & Port** - This is your treasure map. It automatically shows the Arduino boards you've plugged into your computer, and tells you their port number.
- **Sketchbook** - This is your personal library. It's where all your sketches (programs) are stored on your computer. Plus, it can connect to the Arduino Cloud, so you can fetch your sketches from the online world too.
- **Boards Manager** - Think of this as your toolkit. It's where you can find and install different packages for your Arduino.
- **Library Manager** - This is your endless treasure chest. Thousands of libraries made by Arduino and its community are waiting for you here. Need a tool or material for your code? Dive in and find it!
- **Debugger** - Imagine you had a superpower that let you test and debug your code in real time, finding and fixing problems as they happen. That's what this is!
- **Search** - Think of this as your magnifying glass. It helps you search for keywords in your code.
- **Open Serial Monitor** - This is like your communicator device. It opens a new tab that lets your computer and Arduino board send messages back and forth.

Now that we've gotten a glimpse of the playground, it's time to dive in and start creating!

Step 5: Upload Your First Sketch

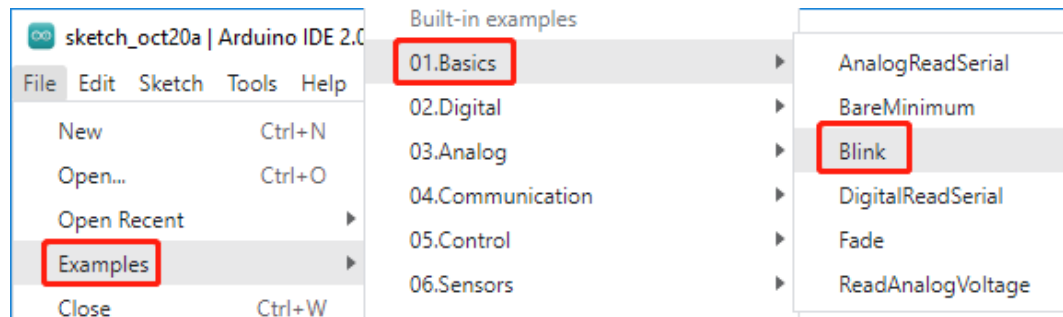
Alright, it's time to have some fun! We're going to make an LED blink - it's like saying "Hello, World!" in the world of Arduino.

Most Arduino boards have a built-in LED on pin 13, which makes this a good first experiment.

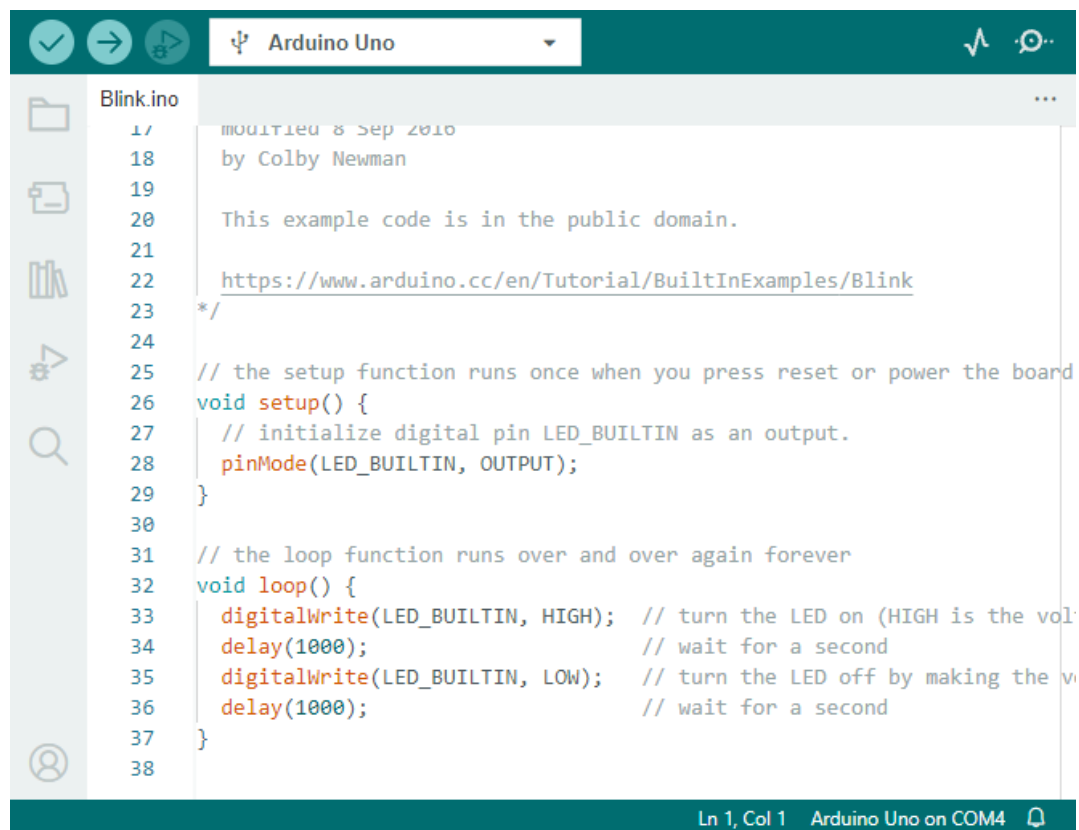


Let's break it down:

1. **Plug it in:** Connect your SunFounder R3 Board to your computer using a USB cable. This is how we're going to give our board power and send our program (also called a "sketch") to it. You might feel like you're just plugging in a computer gadget, but believe me, you're connecting to a world of possibilities!
2. **Find the example sketch:** On the Arduino IDE, go to **File -> Examples -> Basic -> Blink**. What you see that pops up is a ready-to-use program that we're going to modify. It's like getting a ready-made cake that we're about to decorate!



3. **Understand the sketch:** Look at the code in this new window. It tells Arduino to turn on the built-in LED (which is on pin 13) for one second, then turn it off for one second, and then repeat. It's like sending Morse code, but with light!



4. **Upload the sketch:** Once you've selected the correct board and port, just click the upload button. It's as easy as sending a letter; you're delivering your instructions to the Arduino board! Most of the time, the system will automatically detect the board and port for you.
5. **Watch it work:** If all goes well, you'll see the LED on your Arduino board start to blink on and off. It's like your Arduino is winking at you!

You've done a great job! You've just run your first Arduino program, making you a bona fide programmer! So what's next? We're just scratching the surface of what Arduino can do. Ready for the next challenge?

Step 6: Some Fun Arduino Programming Facts

Time to uncover some cool secrets about Arduino programming!

- Code Magic: `setup()` and `loop()`

An Arduino sketch, or a piece of code, is like a two-act play:

- `setup()`: This is Act 1, the opening scene. It only happens once, when your Arduino board first wakes up. It's used to set the stage by preparing things like pin modes and libraries.
- `loop()`: After Act 1, we move onto Act 2 which repeats on a loop until the final curtain (which only happens if we turn off the power or hit the reset button!). This part of the code is like the main part of our play, where the action really happens.

But remember, even if there's no magic (code) in the `setup()` or `loop()`, we still need to keep them. They're like the stage - even an empty stage is still a stage.

```
void setup() {
    // initialize digital pin LED_BUILTIN as an output.
    pinMode(LED_BUILTIN, OUTPUT);

    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the
    ↪voltage level)
    delay(1000);                     // wait for a second
    digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the
    ↪voltage LOW
    delay(1000);                     // wait for a second
}

// the loop function runs over and over again forever
void loop() {
}
```

- Punctuation Marks in Coding

Just like in a storybook, Arduino uses special punctuation marks to make sense of the code:

- Semicolons (;): These are like the full stops in a story. They tell the Arduino “Okay, I’m done with this action. What’s next?”
- Curly Braces {}: These are like the beginning and the end of a chapter. They wrap up pieces of code together, marking where a section starts and ends.

If you happen to forget some of these punctuation marks, don’t worry! The Arduino is like a friendly teacher who will check your work, point out where the mistakes are, and show you how to fix them. It’s all part of the learning adventure!

- About the Functions

Imagine these functions as magical spells. Each spell has a specific effect in our Arduino adventure:

- `pinMode()`: This spell decides whether a pin is an INPUT or an OUTPUT. It’s like deciding if a character in our story speaks (OUTPUT) or listens (INPUT).
- `digitalWrite()`: This spell can turn a pin HIGH (on) or LOW (off), like switching a magic light on and off.
- `delay()`: This spell makes the Arduino pause for a certain amount of time, like taking a short nap in the middle of our story.

Just like a spell book, you can find all these spells and many more in the . The more spells you know, the more exciting your Arduino adventures can be!

- **Comments: Our Secret Messages**

We also have a secret language in coding, called `comments`. These are messages that we can write in our code using `//` or `/* */`. The magic part? The Arduino completely ignores them! It's a great place to leave notes for yourself or others to explain what the tricky parts of the code are doing.

- **Code Readability: Making Code Friendly**

While you can write your code in any manner you want (for example, placing semicolons on a separate line won't cause any errors), it's important to keep in mind the readability of the code.

Just like writing a good story, the way we write code can make it either fun and easy or boring and difficult to read. Here are some ways to make your code more friendly:

- Use proper indentation to arrange your sentences into neat paragraphs. It helps the reader understand where one section ends and another begins.
- Use variable names that make sense. It's like calling a character by a fitting name in a story.
- Keep your functions small and simple, like short and sweet chapters in a book.
- Leave comments for the tricky parts. It's like leaving a footnote to explain a difficult word.

Remember, we're not only coding for machines but also for humans, so let's make sure our code tells a clear and understandable story!

Step 7: Reflect and Improve

Taking a moment to reflect on our journey can provide us with insights that we might miss in the flurry of exploration. Ask yourself:

- What was the most interesting part of this Arduino adventure?
- Were there any challenges along the way? How did you overcome them?
- Could you explain to a friend what Arduino is, what the Arduino IDE does, or how to run Arduino code?
- How would you describe your first Arduino programming experience?
- What more do you want to learn about Arduino?

By thinking about these questions, you are deepening your understanding and preparing yourself for future explorations. Always remember, there's no "wrong" answer in reflection – it's your personal journey after all!

3.4 Lesson 4: Mastering the TT Motor

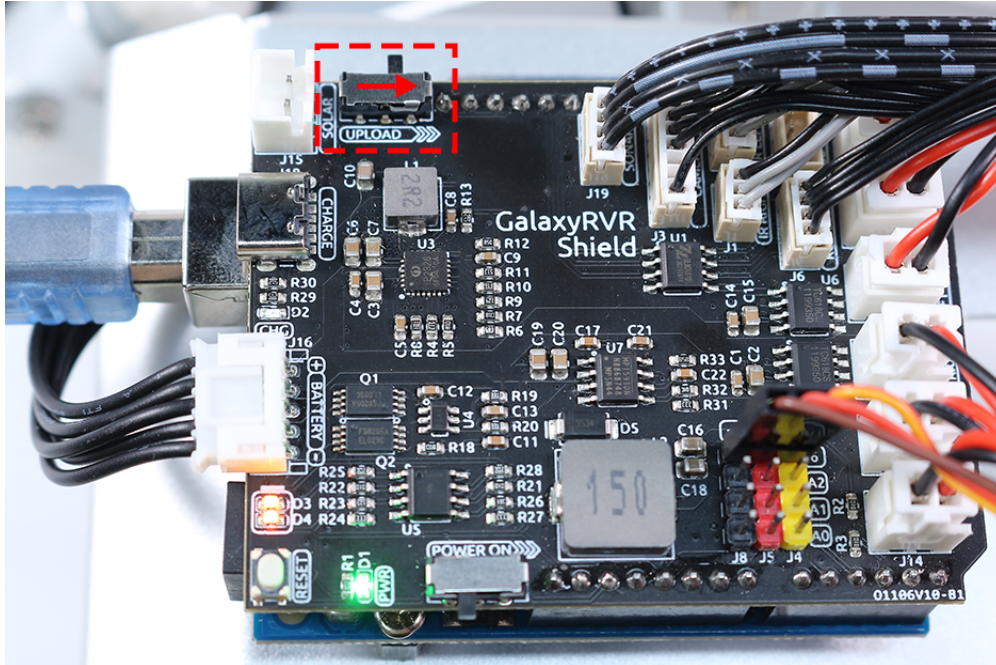
In the previous lessons, we explored Mars rovers, their suspension systems, and delved into knowledge about Arduino.

In this exciting course, we'll explore the workings of motors, a key component that drives Mars rovers. We'll understand the principles that power these motors and learn to control them using SunFounder R3 board and a GalaxyRVR Shield.

By the end of this course, you'll have a solid understanding of motor operation and hands-on experience in motor control.

Let's dive in!

Note: If you are learning this course after fully assembling the GalaxyRVR, you need to move this switch to the right before uploading the code.



3.4.1 Course Objectives

- Understand the basic principles of motors and the characteristics of the TT motor.
- Learn how to control the direction and speed of the TT motor.
- Understand how the GalaxyRVR Shield controls six motors.

3.4.2 Course Materials

- SunFounder R3 Board
- TT Motor
- GalaxyRVR Shield
- Battery
- USB Cable
- Arduino IDE
- Computer

3.4.3 Course Steps

Step 1: What is a Motor?

Motors play an integral part in our daily lives. They're everywhere! From the electric fans that cool us on hot days, the mixers that help us make delicious cakes, to the electric cars that whizz by on the streets – motors make things move!



A motor is like the heart of a machine. It converts electrical energy into mechanical energy, making our toys, appliances, and even big vehicles come to life!

The magic behind a motor isn't magic at all - it's science, specifically the principle of electromagnetic induction. Here's how it works: when electricity is supplied to a motor, it generates a magnetic field. This magnetic field then interacts with other magnets within the motor, causing the motor to spin. This spin, like spinning a top, can then be used to move wheels, propellers, or any other moving parts of a machine.

The type of motor we're focusing on in our GalaxyRVR is a specific kind called a TT Gear Motor.



This is essentially a regular motor combined with a series of gears, all encased within a plastic shell.

As the motor spins, the gears translate this spin to the wheels of our rover. The use of gears provides a crucial benefit - it increases torque, enabling the motor to move larger, heavier loads.

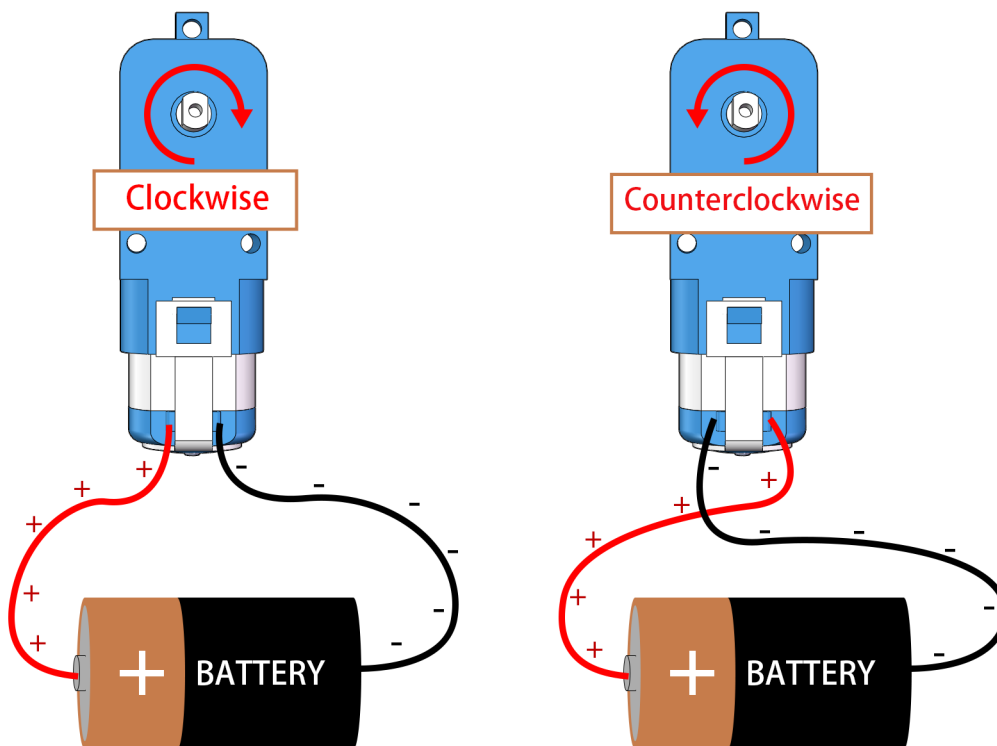
Isn't it fascinating to see how science and engineering principles come to life? Motors are a perfect example of these principles in action. By understanding how motors work, we can dream up and invent a wide array of machines. Let's dive deeper into the world of motors and unleash our creativity!

Step 2: Exploring Motor Functioning and Operation

Having understood what a motor is and its broad spectrum of applications, it's time we venture into the heart of motor operation.

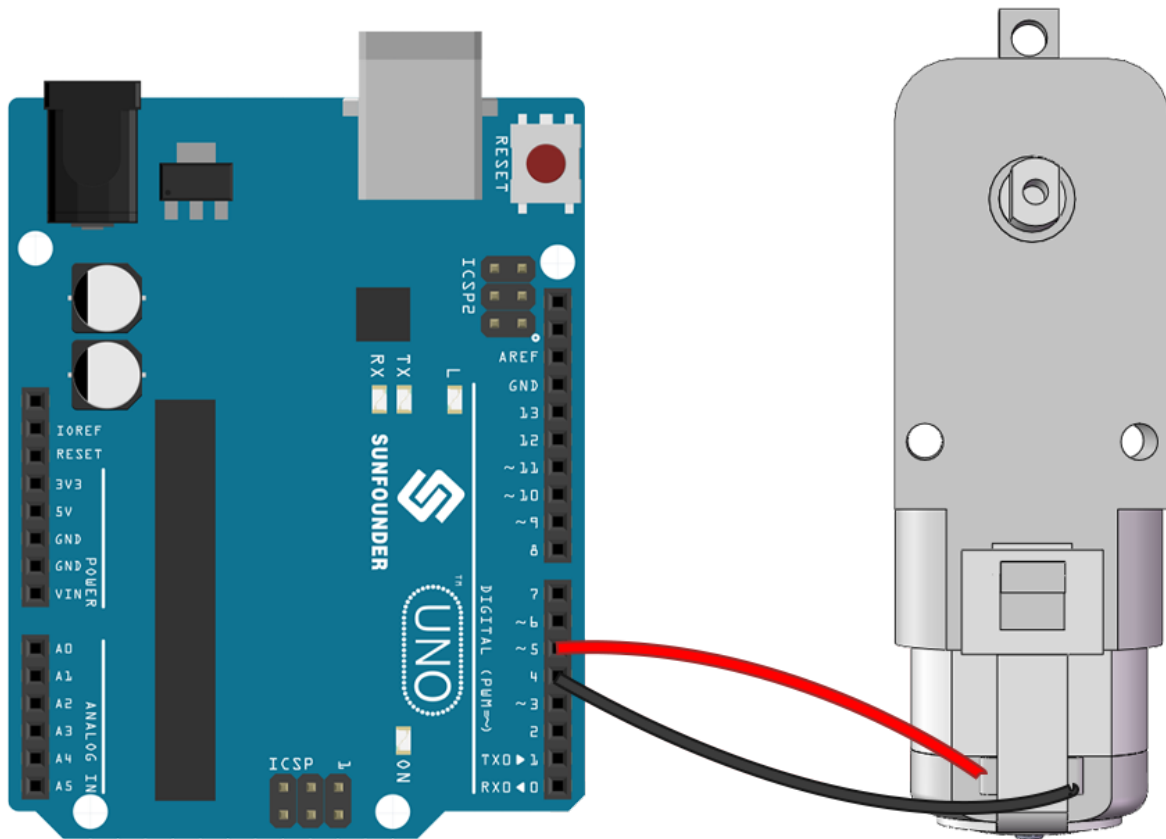
In essence, a motor works on the principle of electromagnetism. When an electric current passes through a wire, it generates a magnetic field around it. This magnetic field can interact with other magnetic fields, causing motion.

Consider a simple experiment where we connect a motor directly to a battery. The current from the battery flows into the motor, triggering the internal mechanism of the motor to start spinning. This spinning action is due to the magnetic forces inside the motor.



Interestingly, if you reverse the connections to the battery, the motor spins in the opposite direction! This happens because the direction of current flow changes, altering the direction of the magnetic field and consequently the direction of the motor's spin.

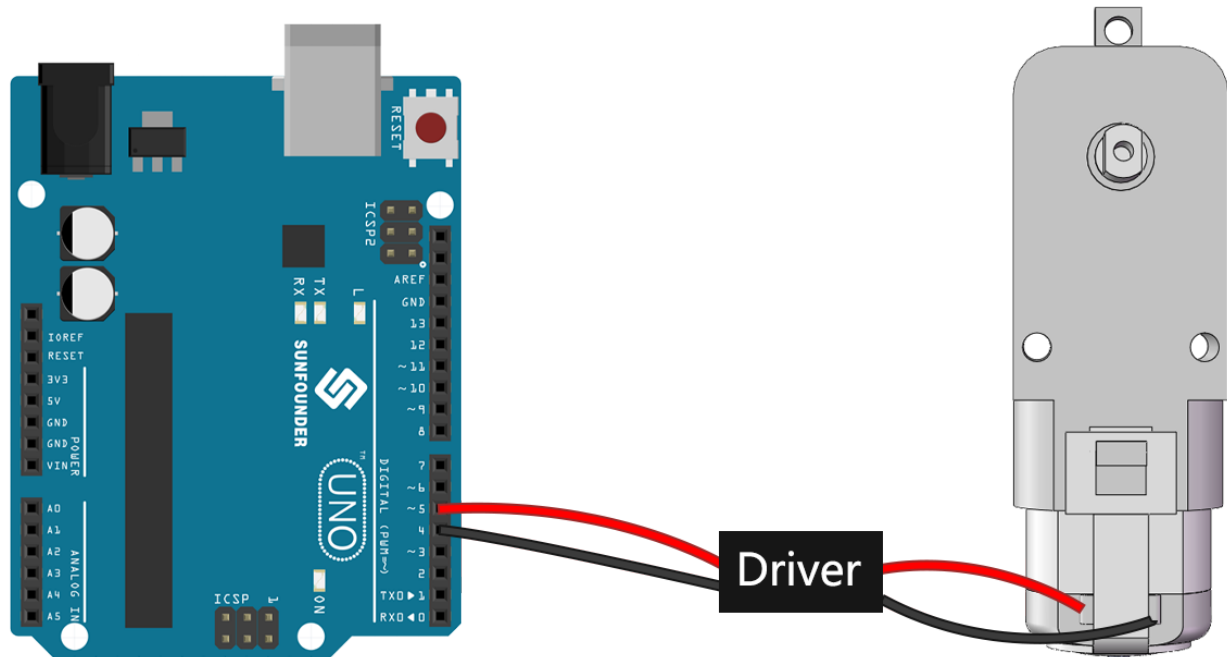
Now we know that connecting the motor directly to a battery can make it spin, but often we want to control its movement with code, so we include an Arduino board between them. But what would happen if we tried to connect the motor directly to the signal pins on the Arduino board?



If you guessed that the motor would not spin, you are correct! But why is that so?

The answer lies in the current output of the Arduino board. The signal pins on a typical Arduino board can output only about 20mA of current, which is insufficient to drive a motor.

So, how can we control motors using our Arduino? This is where a crucial component comes into the picture - a motor driver. Think of a motor driver as a bridge between the Arduino and the motor. It takes the low-current control signal from the Arduino, amplifies it, and sends it to the motor, thus enabling the motor to spin.

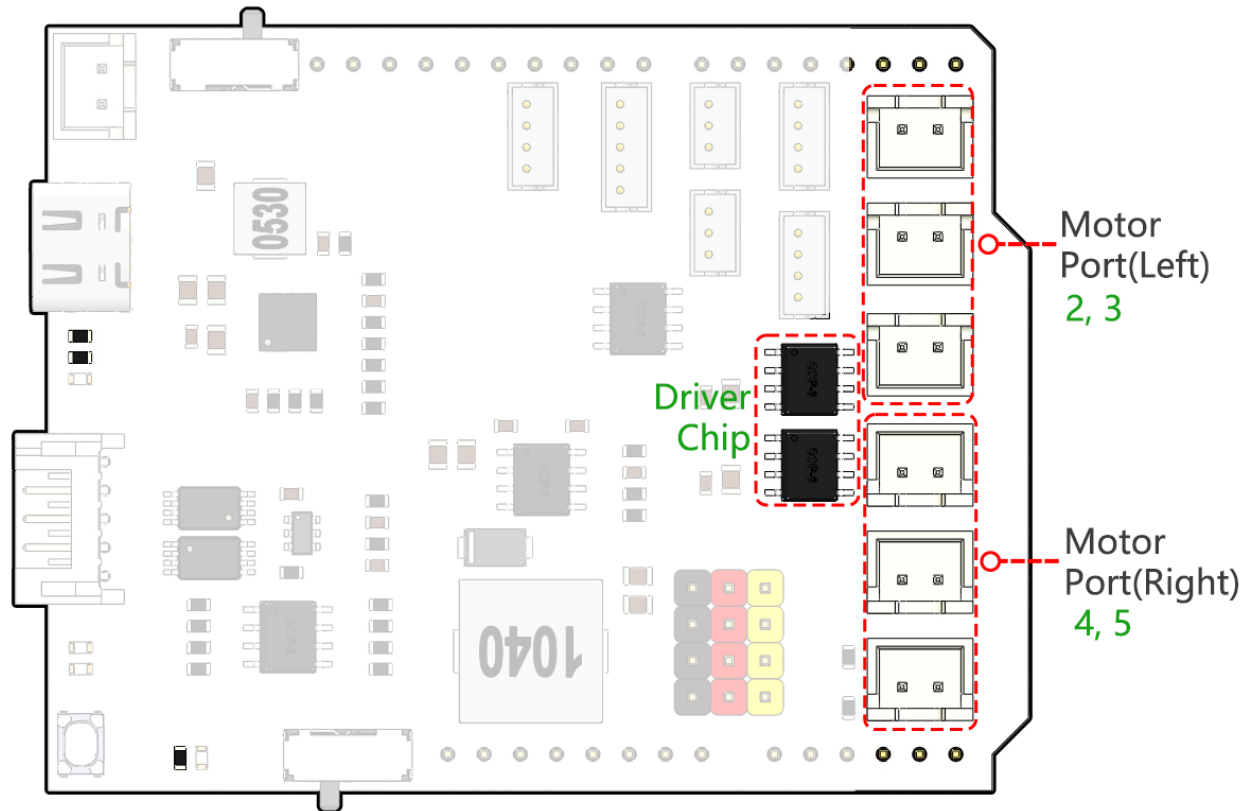


In our next step, we'll dive into the specifics of the motor driver and understand how we can effectively use it with our Arduino board to control a motor. Stay tuned for more exciting learning!

Step 3: How the Motor is controlled by the Motor Driver

Our GalaxyRVR Shield, included in the kit, serves as the control center for our Mars Rover. It is the hub where we connect all our sensors, motors, and power supply. It consists of several components that allow us to control and power our Rover effectively.

On the right side of the shield, you'll notice six motor ports. However, they are grouped into two sets, each controlled by a separate motor drive chip. Three ports marked "Left" are controlled by one chip, and the other three ports marked "Right" are controlled by another.



Let's learn how these two drive chips control the six motors through hands-on experience:

- **1. Connecting the Circuit**

1. Plug the GalaxyRVR Shield into the R3 board, connect a motor, and finally plug in the battery to provide power to the expansion board.
2. The first time you use, it is recommended that you plug in a Type-C USB cable to fully charge the battery first. Then turn the power on.

- **2. Writing and Uploading Code**

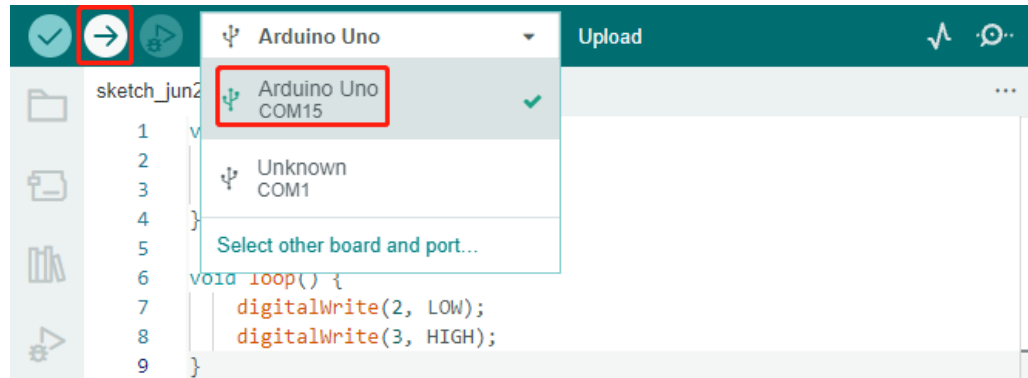
1. Open the Arduino IDE and input the following code:

```
void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
}

void loop() {
  digitalWrite(2, LOW);
  digitalWrite(3, HIGH);
}
```

- `pinMode()`: This function sets a pin as INPUT or OUTPUT, akin to deciding whether a character in our story speaks (OUTPUT) or listens (INPUT).

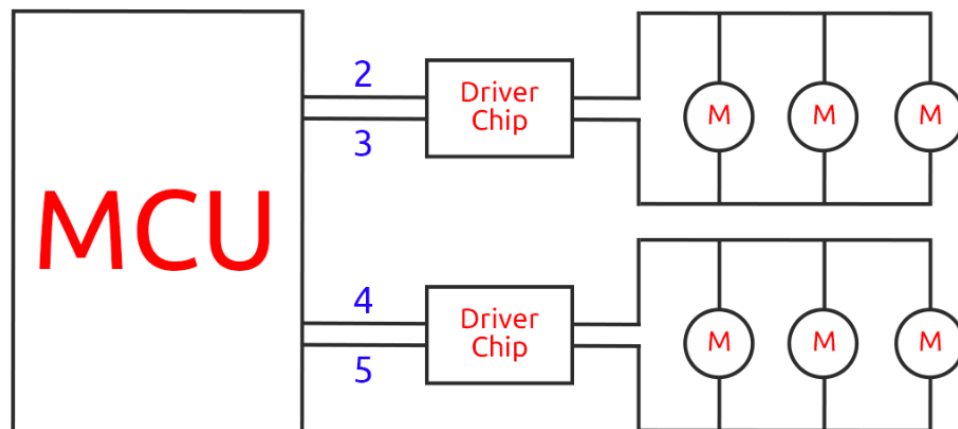
- `digitalWrite()`: This function can set a pin HIGH (on) or LOW (off), much like switching a magic light on and off.
- Once you've selected the correct board(Arduino Uno) and port, click on the **Upload** button. It's like putting a letter in a mailbox - you're sending your instructions off to Arduino!



- Once the code has been successfully uploaded, you will see the motor start to rotate clockwise.

• 3. About Circuit Internal Connection

- You can plug two more motors into the “Left” marked motor ports. You will see them rotate simultaneously.
- Now, let's understand the simple principle of how the two drive chips control the six motors. Pins 2 and 3 on the Arduino board output signals to the motor drive chip, and the other end of the chip is connected to three motors in parallel. Similarly, pins 4 and 5 output signals to another drive chip, which in turn is connected to another three motors in parallel.



- If you want to test another drive chip, you just need to change the pins to 4 and 5.

```
const int in3 = 4;
const int in4 = 5;

void setup() {
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
}
```

(continues on next page)

(continued from previous page)

```
void loop() {
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);
}
```

Here, we define two variables to represent pins 4 and 5. By using variables, we can easily manage and adjust our pin assignments throughout our code.

Think of it as if we're assigning a specific role or duty to each pin number. When we decide to reassign the roles, instead of going through the entire script and changing every instance, we just update the assignment at the beginning of the script (where the variable is initially defined).

• 4. About Drive Logic

1. In the previous tests, you would have noticed that the motors all spin in one direction. How do we make it spin in the opposite direction? Someone might suggest swapping the HIGH and LOW of the two pins. That's correct.

```
const int in3 = 4;
const int in4 = 5;

void setup() {
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
}

void loop() {
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);
}
```

Once you've written your code and uploaded it to your Arduino board, the motor will behave as instructed.

2. Let's now look at the internal driving logic of the drive chip.

INA	INB	Motor
L	L	Standby
L	H	Clockwise
H	L	Counterclockwise
H	H	Brake

3. Now, let's try to make the motor rotate clockwise for 2 seconds, counterclockwise for 2 seconds, and then stop.

```
const int in3 = 4;
const int in4 = 5;

void setup() {
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
}
```

(continues on next page)

(continued from previous page)

```
void loop() {  
  digitalWrite(in3, LOW);  
  digitalWrite(in4, HIGH);  
  delay(2000);  
  digitalWrite(in3, HIGH);  
  digitalWrite(in4, LOW);  
  delay(2000);  
  digitalWrite(in3, HIGH);  
  digitalWrite(in4, HIGH);  
  delay(5000);  
}
```

- Here we use the `delay()` function to make the Arduino pause for a certain amount of time, much like taking a short nap in the middle of our story.
- In the code, we use the “Brake” state to stop the motor, and you’ll notice that the motor stops abruptly. Try setting both pins to LOW to test the “Standby” state, and you’ll find that the motor gradually slows down to a stop.

Now that you should have a better understanding of how the motor driver chip controls the motors through the GalaxyRVR Shield and how we can use Arduino code to manipulate the motor’s movements. Isn’t it fascinating how a few lines of code can dictate the behavior of a physical object like our motor?

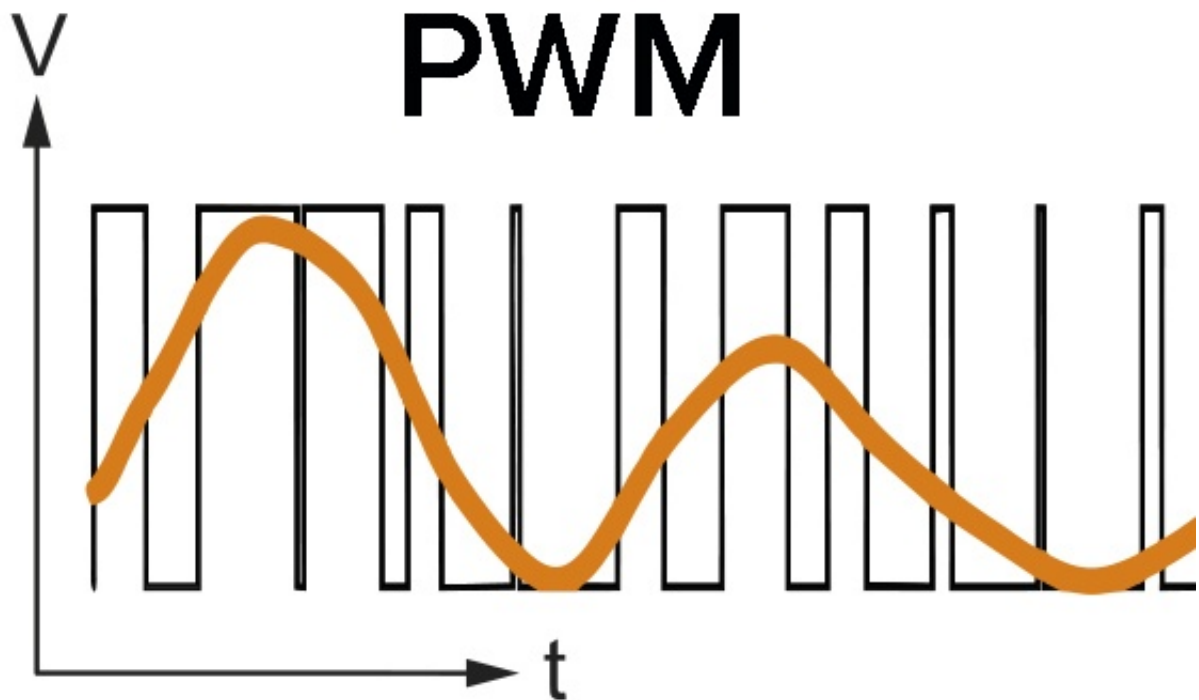
Consider the following questions as you move forward:

- If we move all the code from the `loop()` function into the `setup()` function, how would the behavior of the motor change?
- How would you modify the code to control six motors simultaneously?

Remember, the more you experiment and play around with your code, the more you learn. Feel free to tweak, modify, and optimize your code as you deem fit. Happy coding!

Step 4: Controlling Motor Speed

In the previous step, we controlled the motor’s direction by simply setting its pins HIGH or LOW. This is like giving the motor full power to drive it, similar to pressing the accelerator pedal to the floor in a car. But in many situations, we might want to adjust the motor speed to suit different scenarios, just like we adjust the speed of a car depending on whether we’re driving in a city or on a highway. This is where Pulse Width Modulation (PWM) comes in.



PWM is a technique used to create the effect of variable voltage output by rapidly switching the output between HIGH and LOW. With PWM, we can simulate the effect of an analogue signal while only actually outputting digital signals.

You might be finding this hard to understand, and that's okay! We'll be learning how to adjust motor speed using PWM through coding in the following sections.

Note that although the SunFounder R3 board has some pins with built-in PWM functionality, we can't use them for our motor because they're already serving other functions. Thus, we're connecting the driver chips to pins 2, 3, 4, and 5, and using the Arduino's SoftPWM library to enable PWM on these pins.

Here's what we'll do next:

1. Open Arduino IDE, search for `softpwm` in the **LIBRARY MANAGER** and install it.
2. Enter the following code into Arduino IDE. After uploading the code successfully, the motor will rotate clockwise.

```
#include <SoftPWM.h>

const int in1 = 2;
const int in2 = 3;

void setup() {
  SoftPWMBegin();
}

void loop() {
```

(continues on next page)

(continued from previous page)

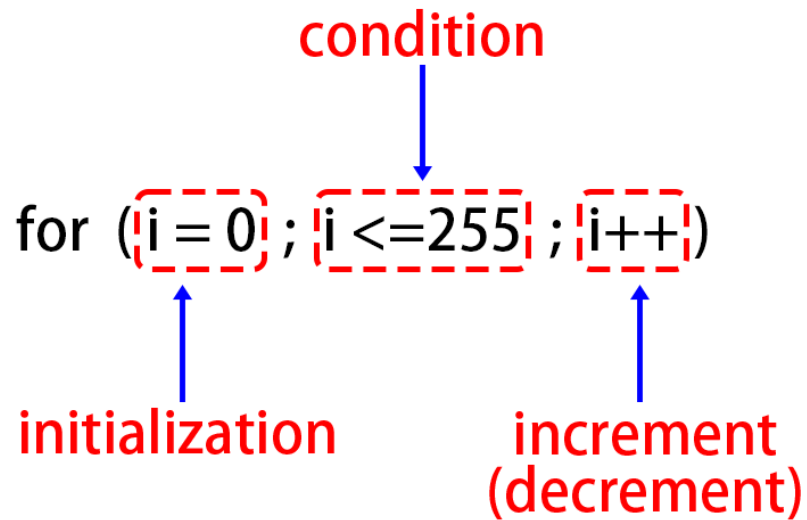
```
SoftPWMSet(in1, 0);  
SoftPWMSet(in2, 255);  
  
}
```

- In the code above, we first add `SoftPWM.h` to the top of the code, enabling us to use the functions in the `SoftPWM` library directly.
- Then, initialize the `SoftPWM` library with `SoftPWMBegin()` function.
- Finally, in the `loop()` function, we use `SoftPWMSet()` to assign different values to `in1` and `in2`, setting the motor in motion. You will notice the effect is similar to directly using `LOW` and `HIGH`, but here we use numerical values within a range of `0~255`.
- Remember, in the world of Arduino, speed is expressed as a value between 0 (like a car at a stop sign) and 255 (zooming down the highway!). So, when we say `SoftPWMSet(in2, 255)`, we're telling that motor to go full speed ahead!

3. Now, let's enter other values and observe any differences in motor speed.

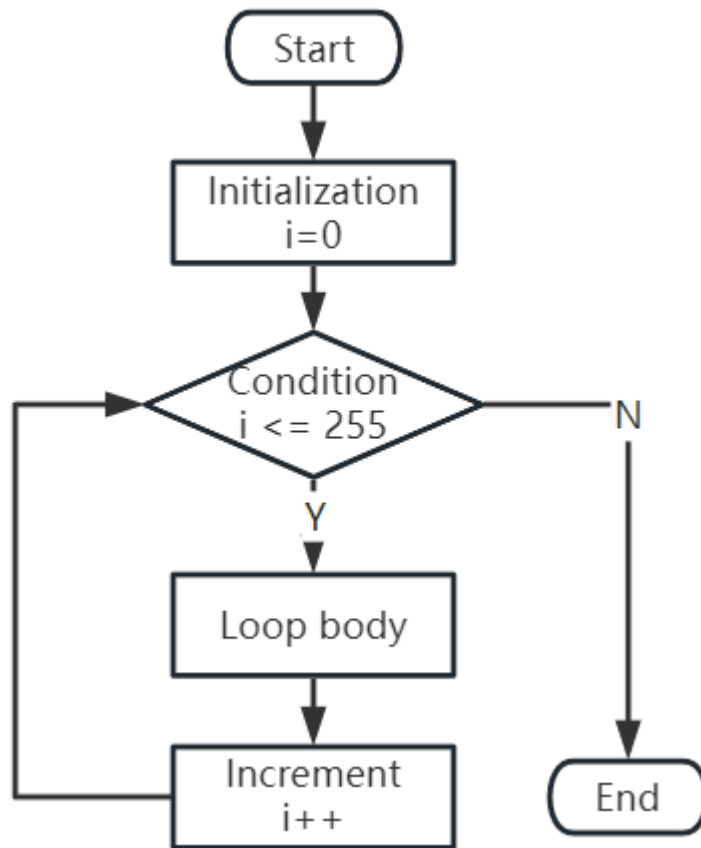
```
#include <SoftPWM.h>  
  
const int in1 = 2;  
const int in2 = 3;  
  
void setup() {  
    SoftPWMBegin();  
}  
  
void loop() {  
    SoftPWMSet(in1, 0);  
    for (int i = 0; i <= 255; i++) {  
        SoftPWMSet(in2, i);  
        delay(100);  
    }  
    delay(1000);  
}
```

In the code above, we use a `for` loop to increment a variable `i` up to 255. The `for` loop in C language is used to iterate over a part of the program several times. It consists of three parts:



- **Initialization:** This step is executed first and only once when we enter the loop for the first time. It allows us to declare and initialize any loop control variables.
- **Condition:** This is the next step after initialization. If it's true, the body of the loop is executed. If it's false, the body of the loop does not execute and the flow of control goes outside of the for loop.
- **Increment or Decrement:** After executing the Initialization and Condition steps and the loop body code, the Increment or Decrement step is executed. This statement allows us to update any loop control variables.

The flowchart for the for loop is shown below:



So, after running the above code, you will see the motor speed gradually increasing. It stops for a second, and then starts again from 0 and gradually increases.

In this step, we have learned about Pulse Width Modulation (PWM), a technique for controlling the speed of our motor. By using the Arduino's SoftPWM library, we can adjust the speed of the motor, allowing us to simulate analogue signals while only outputting digital signals. This provides us with finer control over our rover's movements, and prepares us for more complex maneuvers in the future.

Step 5: Reflect and Improve

Having completed this lesson, you should now be familiar with the working principles of motors, as well as how to control their direction and speed through programming.

Let's test your understanding with these challenges:

- How would you modify the for loop to gradually decrease the motor speed?
- How would you control the motor to accelerate or decelerate while rotating counterclockwise?

You can experiment with the provided code to answer these questions. Feel free to adjust the code according to your hypotheses and observe the changes in the motor's behavior.

Your hands-on experiments and reflections on these questions will deepen your understanding and enhance your problem-solving skills. It is through challenges like these that real learning occurs. Always remember, there is no "right" or "wrong" in your exploratory journey – this is all about learning and discovery!

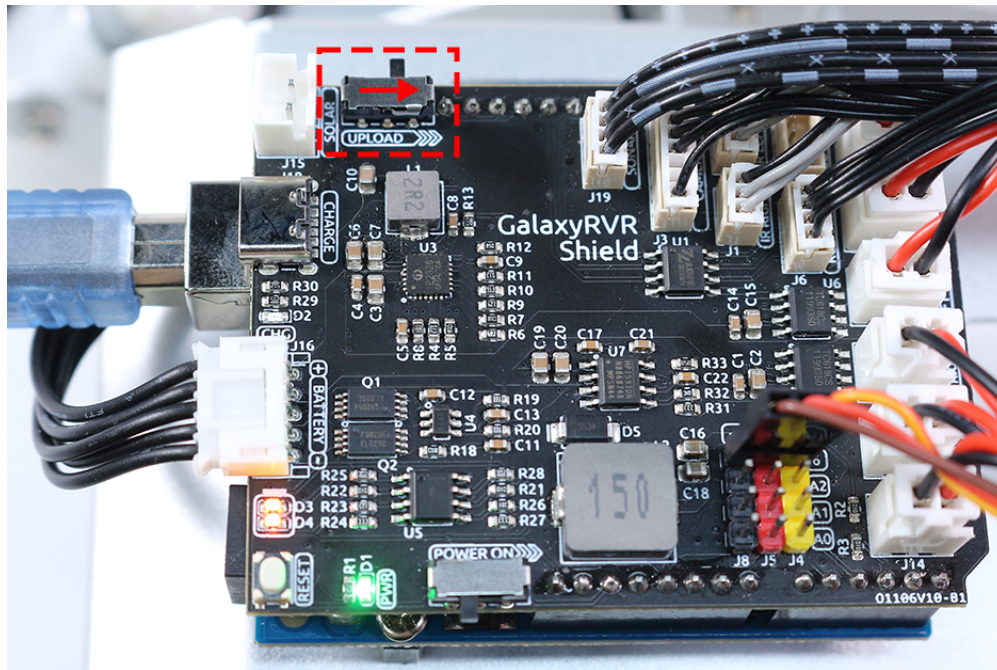
3.5 Lesson 5: Unleashing Mars Rover Mobility

Get ready, young explorers! We've mastered controlling motors, now we're taking those skills to the red planet. In this lesson, we're bringing the Mars Rover to life!

We'll learn how to assemble the motors into the Rocker-Bogie suspension system and then, using our coding skills, we'll guide our Rover across imagined Martian landscapes.

It's a Martian adventure in our classroom. Let's get started!

Note: If you are learning this course after fully assembling the GalaxyRVR, you need to move this switch to the right before uploading the code.



3.5.1 Learning Objectives

- Understand how to assemble the motors into the Mars Rover's Rocker-Bogie suspension system.
- Learn to use Arduino to control the motion of the Mars Rover.
- Practice writing a program to control the Mars Rover's motion on different terrains.

3.5.2 Materials needed

- SunFounder R3 Board
- TT Motors
- GalaxyRVR Shield
- Battery
- Mars Rover Model (Equipped with Rocker-Bogie System)
- Basic tools and accessories (e.g. screwdriver, screws, etc.)
- USB Cable
- Arduino IDE
- Computer

3.5.3 Steps

Step 1: Assembling the Rover Components

In this step, we will assemble the battery, R3 board, GalaxyRVR Shield, motors, and wheels onto the pre-assembled rocker-bogie system. This will bring the GalaxyRVR to a runnable state.

Congratulations! We've successfully built our very own rover, and it's ready to start exploring. Let's get moving!

Step 2: Set the Rover in Motion

Now it's time to breathe life into our creation and send it off on its maiden voyage. But how do we communicate with our rover? How do we tell it where to go and what to do? That's where our coding skills come into play!

In the real world, if we want a car to move forward, we push the accelerator, and both wheels start spinning. The wheels on the right side turn clockwise, while those on the left side turn counterclockwise.

Imagine you're sitting in the driver's seat, the world whizzing past you as you cruise along the open road - that's exactly the experience we're going to give our rover.

Now, let's translate that experience into the language our rover understands - code!

```
#include <SoftPWM.h>

// Define the pins of motors
const int in1 = 2;
const int in2 = 3;
const int in3 = 4;
const int in4 = 5;

void setup() {
  // Initialize SoftPWM
  SoftPWMBegin();
}

void loop() {
  // Set the left motors rotate counterclockwise
  SoftPWMSet(in1, 255); // Full speed
```

(continues on next page)

(continued from previous page)

```
SoftPWMSet(in2, 0);    // Stop

// Set the right motors rotate clockwise
SoftPWMSet(in3, 0);    // Stop
SoftPWMSet(in4, 255);  // Full speed

}
```

In this code, we're speaking to our rover, telling it exactly what to do. With the `SoftPWMSet()` function, we're acting like the car's accelerator and brakes, controlling the speed and direction of each motor. We tell the left motors to spin counterclockwise and the right motors to spin clockwise, and just like that, our rover moves forward!

Absolutely, the concept of reversing the rover is straightforward once you understand how to move it forward. To make the rover move backward, we just need to reverse the direction of rotation of each motor.

Here's how we'd do that in code, we do just the opposite. The right wheels should now rotate counterclockwise, and the left wheels should rotate clockwise.

```
#include <SoftPWM.h>

// Define the pins of motors
const int in1 = 2;
const int in2 = 3;
const int in3 = 4;
const int in4 = 5;

void setup() {
    // Initialize SoftPWM
    SoftPWMBegin();
}

void loop() {
    // Set the left motors to rotate clockwise
    SoftPWMSet(in1, 0);    // Stop
    SoftPWMSet(in2, 255);  // Full speed

    // Set the right motors to rotate counterclockwise
    SoftPWMSet(in3, 255);  // Full speed
    SoftPWMSet(in4, 0);    // Stop
}
```

In this code, we use `SoftPWMSet()` to tell the left motors to rotate clockwise and the right motors to rotate counterclockwise.

Isn't it fascinating that we can control our rover's journey simply with code? The next time you're in a car, take a moment to think about the journey of your rover, exploring the world one rotation at a time. Stay tuned, because our rover's journey is just beginning!

Step 3: Making the Rover Move in Other Directions

Now that we know how to move our Mars Rover forward and backward, what if we want it to turn left or right?

Just as in real life driving, there are two main ways for a car to turn left.

- The first way is by having the wheels on the left side rotate slower than the ones on the right. This difference in

speed will make the rover turn towards the left.

- The second way is by making both left and right motors rotate in the same direction (clockwise in this case), which will make the rover spin to the left on its axis.

Let's see how we can implement both ways in code:

Method 1: Different speeds on each side

```
#include <SoftPWM.h>

// Define the pins of motors
const int in1 = 2;
const int in2 = 3;
const int in3 = 4;
const int in4 = 5;

void setup() {
    // Initialize SoftPWM
    SoftPWMBegin();
}

void loop() {
    // Set the left motors rotate counterclockwise in low speed
    SoftPWMSet(in1, 40);
    SoftPWMSet(in2, 0);

    // Set the right motors rotate clockwise in higher speed
    SoftPWMSet(in3, 0);
    SoftPWMSet(in4, 200);

    delay(2000); // Last for 2 seconds
}
```

In this code, we have decreased the speed of the left motors while keeping the right motors at a higher speed. This will make the rover turn towards the left.

Method 2: Rotating all motors in the same direction

```
#include <SoftPWM.h>

// Define the motor pins
const int in1 = 2;
const int in2 = 3;
const int in3 = 4;
const int in4 = 5;

void setup() {
    // Initialize SoftPWM
    SoftPWMBegin();
}

void loop() {
    // Set all motors to rotate clockwise
    SoftPWMSet(in1, 0);
    SoftPWMSet(in2, 255);
```

(continues on next page)

(continued from previous page)

```
SoftPWMSet(in3, 0);  
SoftPWMSet(in4, 255);  
}
```

In this code, we set all motors to rotate clockwise. The rover will spin around its own axis and the direction will change to the left.

For making the rover turn right, the concepts are the same but the directions are reversed. Can you figure out how to do it?

Step 4: Moving in All Directions

As we develop more functionalities for our Mars Rover, our code could become quite long and messy. In programming, it's a good practice to keep your code organized and maintainable. One way to achieve this is by creating separate functions for separate tasks.

In this case, we can create separate functions for each direction the rover can move. This not only makes our code easier to understand but also allows us to re-use these functions anywhere in our program without having to re-write the same lines of code.

Let's see how we can do this:

With this structure, our main loop remains simple and easy to follow. We can clearly see that the rover moves forward, then backward, then turns left and right, and finally stops.

The details of how each of these actions is achieved are hidden away in separate functions. This is a key aspect of good programming practices known as abstraction.

Step 5: Sharing and Reflection

Congratulations on completing this exploration journey with the Mars Rover! This is a practical application of your knowledge of science and engineering, and you did a fantastic job!

Now, you can share how your rover overcomes various terrain obstacles. You could film your rover navigating different terrains, or show off your rover to your friends.

Reflecting on your learning process is also very important. What have you learned during this process? Did any new ideas or creative thoughts emerge? What have you gained in controlling motors and programming?

You can continue to explore, discover new knowledge, and constantly improve yourself. Keep it up, future scientists and engineers!

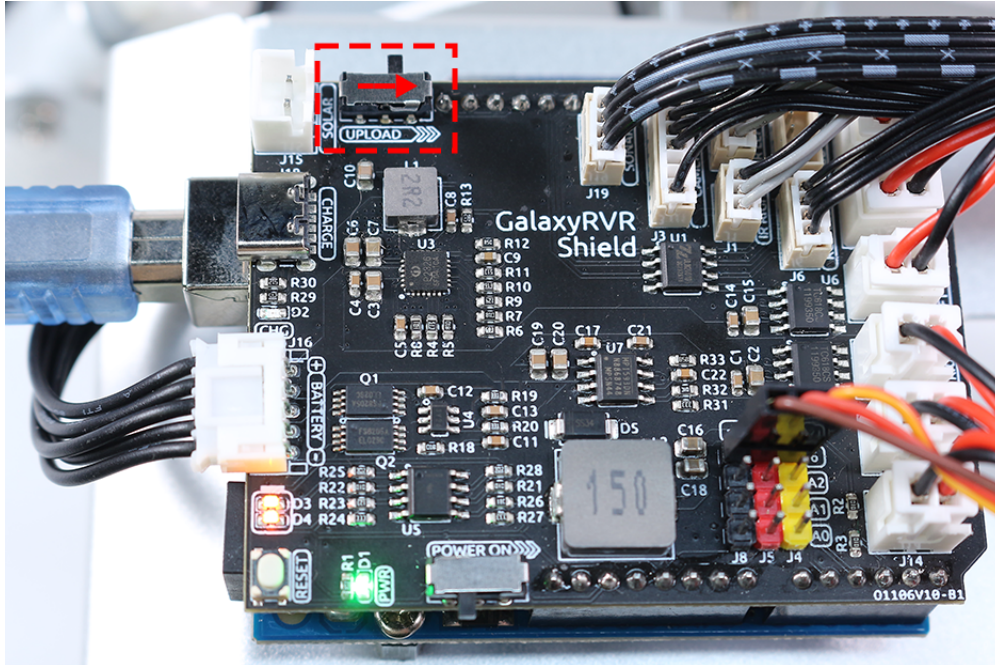
3.6 Lesson 6: Exploring the Obstacle Avoidance Module

We're diving into the world of the Infrared Obstacle Avoidance Module. Tucked at the sides of our Mars Rover, these sensors act as the rover's "eyes," helping it dodge side obstacles and safely navigate the Martian landscape.

We'll learn how to integrate these modules with our rover, unravel the magic behind their functioning, and develop code to make our rover smartly sidestep any hurdles it comes across.

Get ready to gear up our rover with some Martian obstacle-dodging intelligence! Let's get rolling!

Note: If you are learning this course after fully assembling the GalaxyRVR, you need to move this switch to the right before uploading the code.



3.6.1 Learning Objectives

- Understand the working principle and application of the infrared obstacle avoidance module.
- Learn to use Arduino to control the infrared obstacle avoidance module.
- Practice designing and building an automatic obstacle avoidance system based on infrared obstacle avoidance.

3.6.2 Materials Needed

- Obstacle Avoidance Modules
- Basic tools and accessories (e.g. screwdriver, screws, wires etc.)
- Mars Rover Model (Equipped with rocker-bogie system, main boards, motors)
- USB Cable
- Arduino IDE
- Computer

3.6.3 Steps

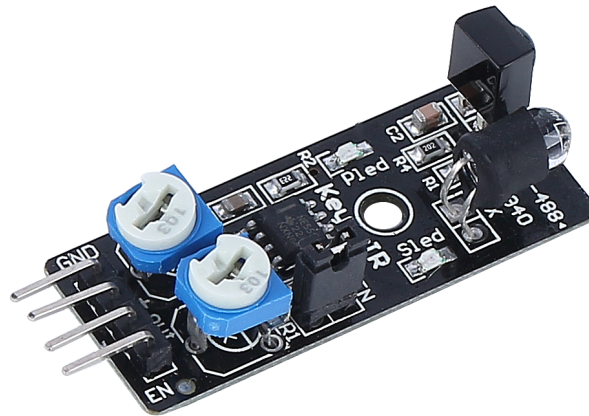
Step 1: Install the Obstacle Avoidance Module

Now we will install the two obstacle avoidance modules onto the rover.

The assembly steps were simple, weren't they? In the following steps, we will learn about the working principle of these modules, and how they help our Mars Rover to avoid obstacles. Stay tuned!

Step 2: Demystifying the Module

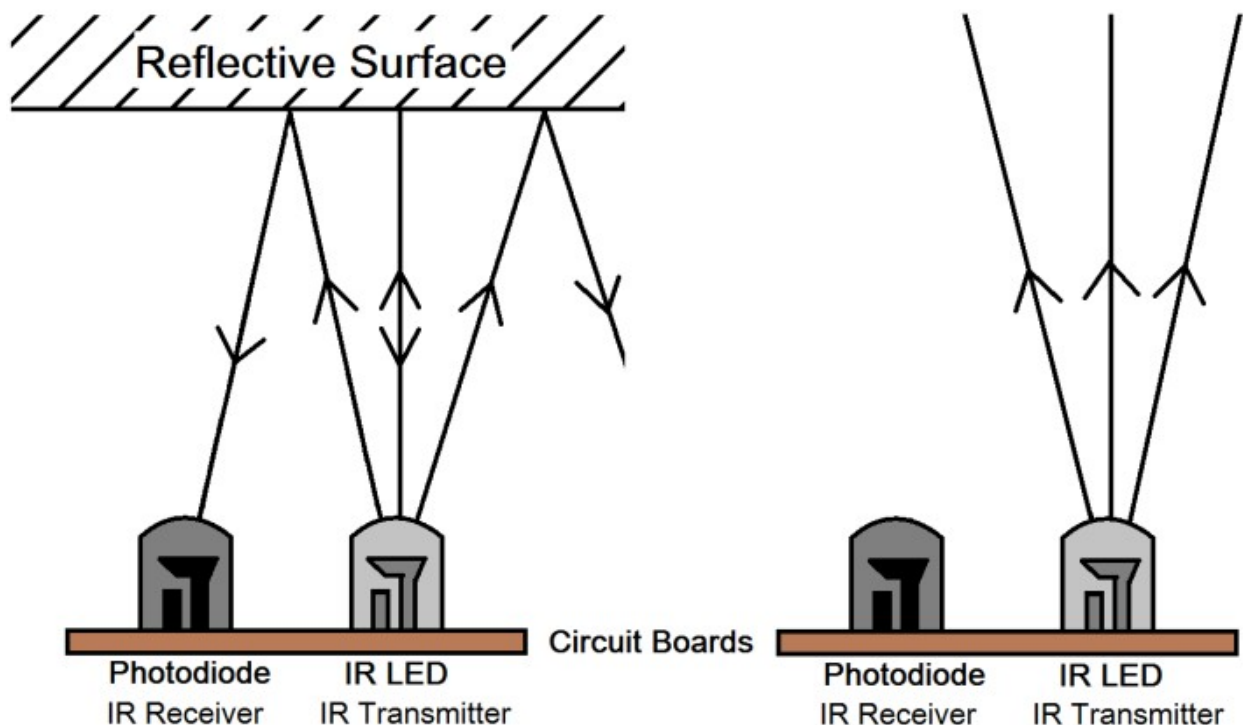
Meet the Infrared Obstacle Avoidance Module - our rover's smart sidekick. This little device is a pack of wonders. Let's take a tour:



Here are the pin definitions:

- **GND**: This is like the module's anchor, connecting it to the ground or common point in the circuit.
- **+**: Here's where the module gets its energy, needing a power supply of 3.3 to 5V DC.
- **Out**: This is the module's communicator. By default, it stays high and only goes low when it spots an obstacle.
- **EN**: Meet the module's controller. This **enable** pin decides when the module should work. By default, it is connected to GND, meaning the module is always on the job.

Curious about how this tiny module works? It's quite interesting! It uses a pair of IR components - a transmitter and a receiver. The transmitter is like the module's flashlight, emitting infrared light. When an obstacle appears, the infrared light bounces back and gets caught by the receiver. The module then gives a low signal, alerting our rover of the obstacle.



Our little module is quite robust, spotting obstacles within a range of 2-40cm and boasting excellent anti-interference

abilities. However, the color of objects does impact its sensing. Darker objects, especially black ones, are detected at a shorter range. Against a white wall, the sensor is most efficient, sensing within the 2-30cm range.

The **EN** pin's low-level state activates the module, with the jumper cap securing the **EN** pin to the **GND**. If you wish to control the **EN** pin via code, the jumper cap needs to be removed.



There are two potentiometers on the module, one for adjusting the transmitting power and one for adjusting the transmitting frequency, and by adjusting these two potentiometers you can adjust its effective distance.

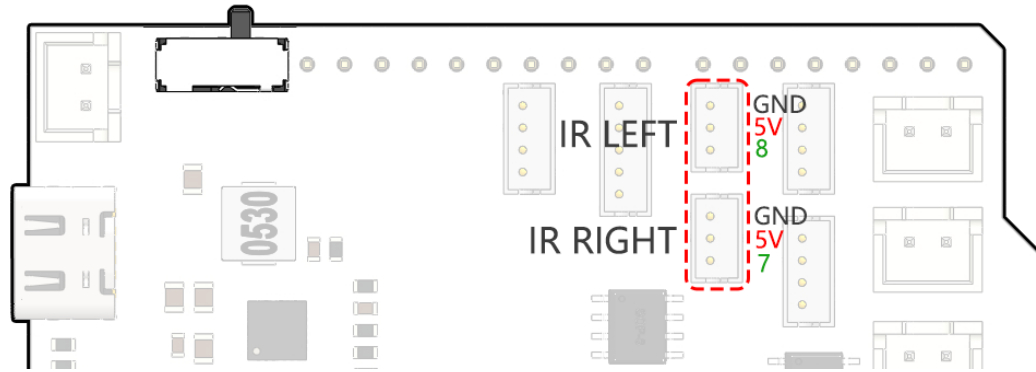


That's quite a bit about our little module. In the next step, we'll learn how to integrate it with our rover and control it using the Arduino. Stay tuned!

Step 3: Read from the 2 Modules

Just like curious space explorers, let's dive into the universe of codes and sensors!

1. Our Mars Rover is equipped with two special "Alien-Eye" sensors, perched neatly on pins 7 (right) and 8 (left). These "Alien-Eye" sensors are actually our infrared obstacle avoidance modules, always vigilant to dodge any "space rocks" (obstacles) in our Rover's interstellar journey!



2. Next, we'll need to communicate with our Rover using the universal language of Arduino code.

First things first, let's give a unique name to each eye of the Rover. Let's call them IR_RIGHT and IR_LEFT, this way we won't mix them up.

```
#define IR_RIGHT 7
#define IR_LEFT 8
```

Now, we let our Rover know that these are its special eyes - they will feed information from the world outside into the Rover's electronic brain.

```
pinMode(IR_RIGHT, INPUT);
pinMode(IR_LEFT, INPUT);
```

To make sure our Rover shares its findings with us, we establish a secret line of communication, like spies in a sci-fi movie. This next line kicks off a serial conversation at the speed of 9600 bits per second - that's lightning fast chatter!

```
Serial.begin(9600);
```

Now, our Rover scans its surroundings with its "Alien-Eyes" and relays the findings back to us. If it spots an obstacle, the value will be 0; if the path is clear, the value will be 1. It keeps sending these messages to us, keeping us in the loop.

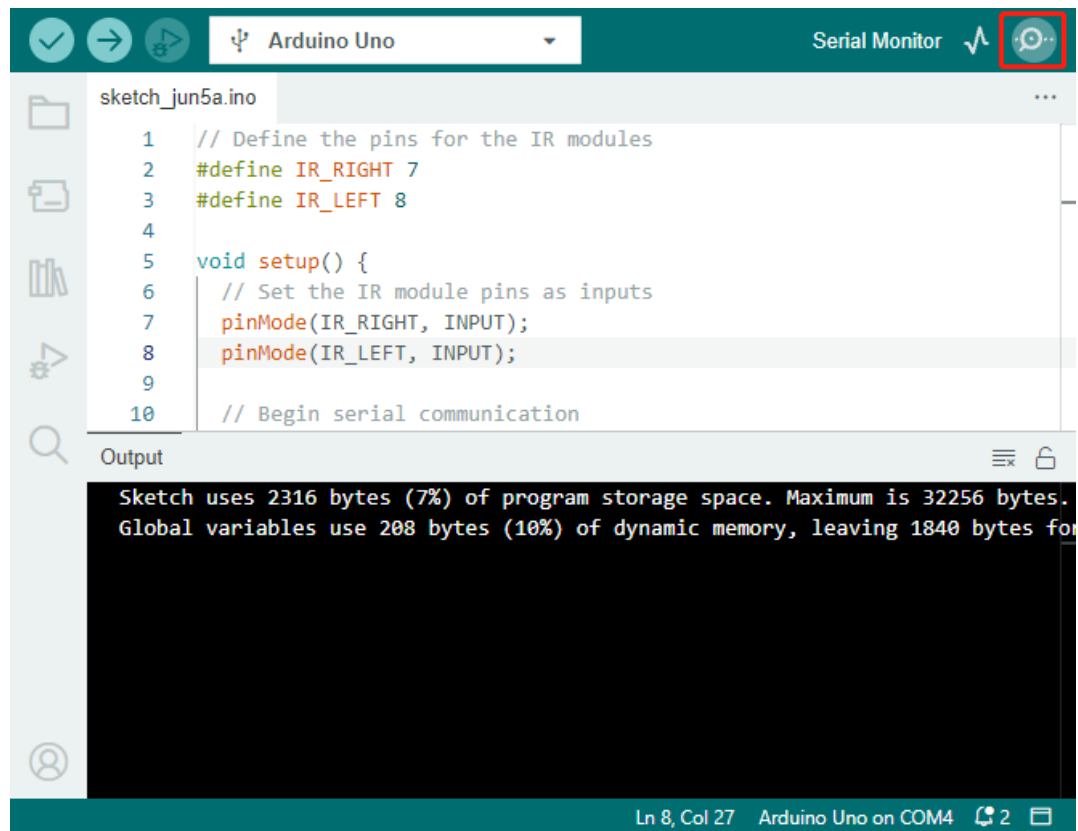
```
int rightValue = digitalRead(IR_RIGHT);
int leftValue = digitalRead(IR_LEFT);
Serial.print("Right IR: ");
Serial.println(rightValue);
Serial.print("Left IR: ");
Serial.println(leftValue);
```

Finally, the Rover pauses for a moment (about 200 milliseconds) after each transmission. This tiny break gives us the chance to interpret the Rover's message before it sends another one.

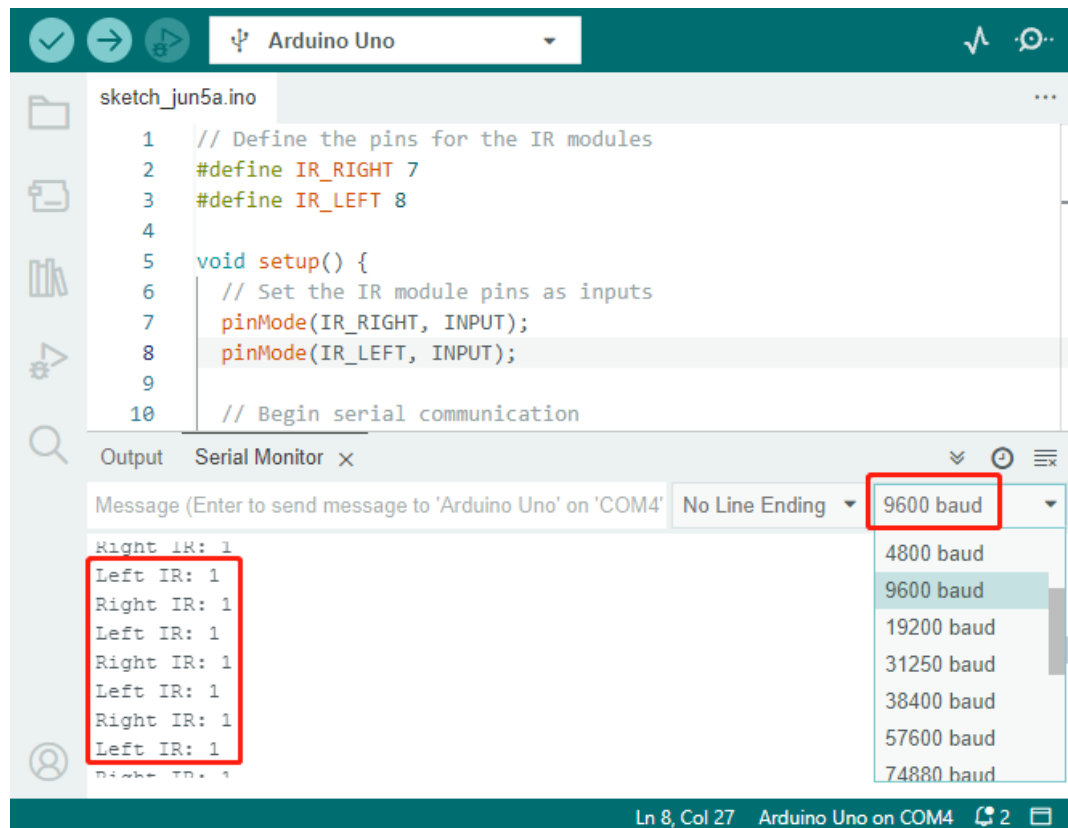
```
delay(200);
```

Here is the complete code:

3. Once your code is ready, select the correct board and port, and beam up the code to your Mars Rover. Then, tune into our secret communication line (the Serial Monitor) by clicking on its icon in the top right corner.



4. Before you start receiving the Rover's messages, make sure your secret communication line is tuned at the same speed (9600 baud) as your Rover. And there you have it - live updates from your Mars Rover!



5. To put our system to the test, wave a “space rock” (your hand) in front of one of the sensors. You’ll see the value flip to 0, and the corresponding LED on the module lights up. That’s the Rover saying, “Look out, space rock on my right!”

```

Right IR: 0
Left IR: 1
Right IR: 0
Left IR: 1
Right IR: 0
Left IR: 1

```

By now, you’ve not just journeyed through space but also deciphered Martian! Can’t wait to see what interstellar secrets we unveil in our next mission!

Step 4: Adjusting the Detection Distance

We have arrived at an essential step, which is to adjust the detection distances of our sensors based on our current environment. The factory settings may not be optimal.

If the detection distance of the two infrared modules is too short, the Mars Rover might collide with obstacles. If it’s too far, the Rover might start turning while still a significant distance from an obstacle, potentially impacting its movement.

Here’s how you can make adjustments:

1. Start by adjusting the right obstacle avoidance module. During transportation, collisions may cause the transmitter and receiver on the infrared module to tilt. Therefore, you need to manually straighten them.
2. Place an obstacle about 20 cm directly in front of the right module. The box in which our Rover kit came is a good choice for this! Now, turn the potentiometer on the module until the indicator light on the module just

lights up. Then, keep moving the obstacle back and forth to check if the indicator light comes on at the desired distance. If the light doesn't turn on at the correct distance or if it remains on without going out, you'll need to adjust the other potentiometer.

3. Repeat the same process for another module.

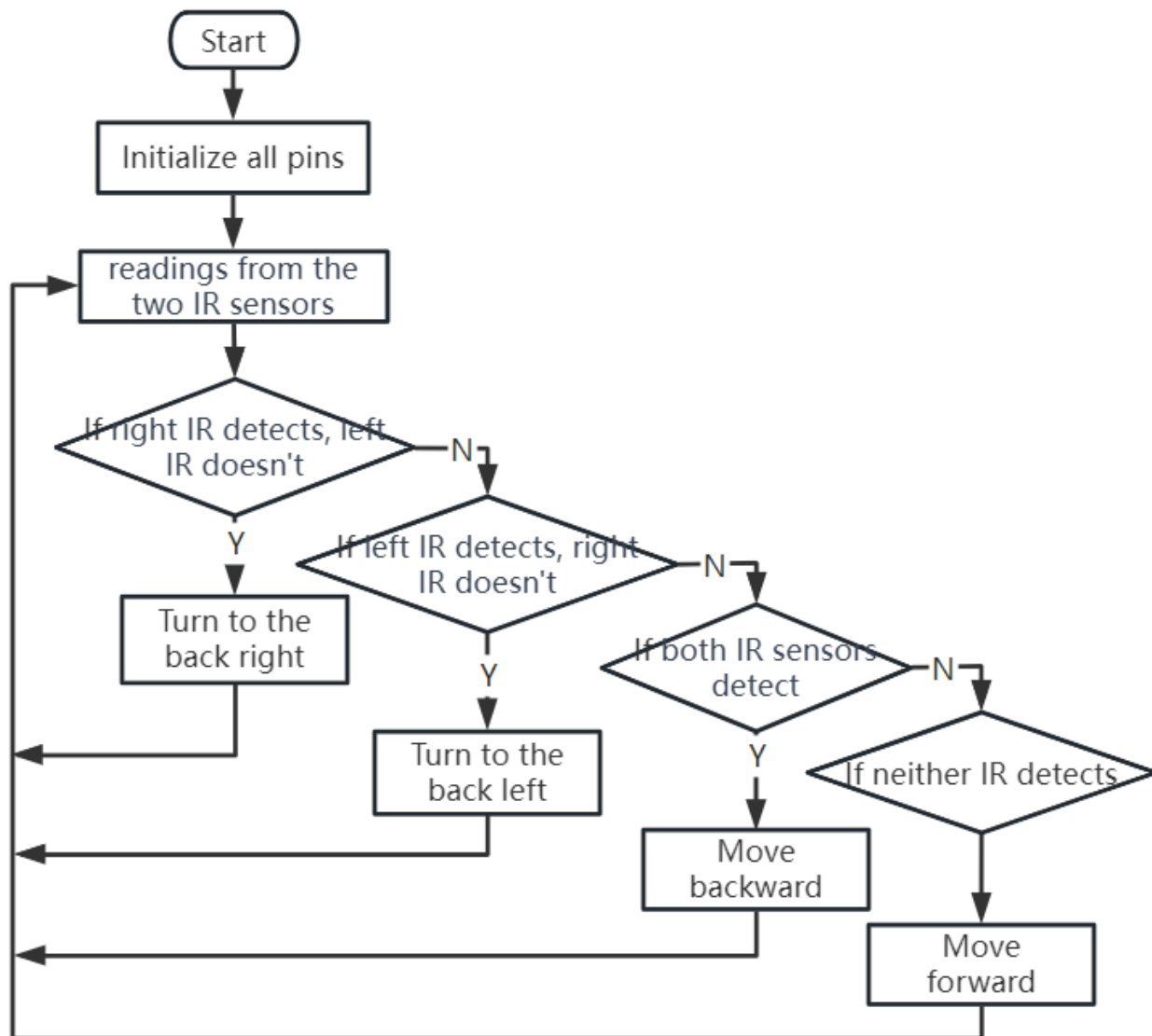
Now that our sensors are fully prepared, let's embark on the next journey!

Step 5: Design an Automatic Obstacle Avoidance System

Now, let's take a big leap in our space exploration and make use of these messages from the Rover. We're going to create an automatic obstacle avoidance system!

Here's our plan: If the right sensor detects an obstacle, the Rover will turn to the back right. If the left sensor detects an obstacle, the Rover will turn to the back left. If both sensors detect an obstacle, the Rover will move backward. If no obstacles are detected, the Rover will continue moving straight ahead.

Let's visualize this in a flowchart to make it even more clear. Flowcharts are a great way to logically outline a plan, especially when it comes to programming!



Let's whisper this plan to our Rover in its language(Arduino code):

In this code, we are using `if...else` statement in the `loop()` function.

The `if...else` statement is used to execute a block of code among two alternatives. However, when we need to choose among more than two alternatives, we use the `if...else if...else` statement.

The syntax of the `if...else if...else` statement is:

```
if (condition1) {  
  // code block 1  
}  
else if (condition2){  
  // code block 2  
}  
else if (condition3){  
  // code block 3  
}  
else {  
  // code block 4  
}
```

Here,

- If condition1 is true, code block 1 is executed.
- If condition1 is false, then condition2 is evaluated.
- If condition2 is true, code block 2 is executed.
- If condition2 is false, then condition3 is evaluated.
- If condition3 is true, code block 3 is executed.
- If condition3 is false, code block 4 is executed.

Now that we've designed our automatic obstacle avoidance system, it's time for the exciting part - putting it to the test!

- You can observe if the Rover moves as you expected.
- Or, place it in different lighting conditions to see how its movements change.

By integrating science into our engineering project, we're becoming space detectives, solving the mysteries of our Rover's behavior. This isn't just about correcting errors but optimizing performance, making our Rover the best it can be! Keep up the fantastic work, space detectives!

Step 6: Reflection and Summary

In the testing phase, you might have noticed an interesting behavior of our Mars Rover: while it expertly avoids obstacles to its left and right, it might struggle to detect smaller obstacles straight ahead.

How can we solve this challenge?

Stay tuned for the next lesson, where we'll continue our exploration into the fascinating world of coding, sensors, and obstacle detection.

Remember, every challenge is an opportunity for learning and innovation. And as we continue our space exploration journey, there's so much more to discover and learn!

3.7 Lesson 7: Enhancing Rover Navigation with Ultrasonic Module

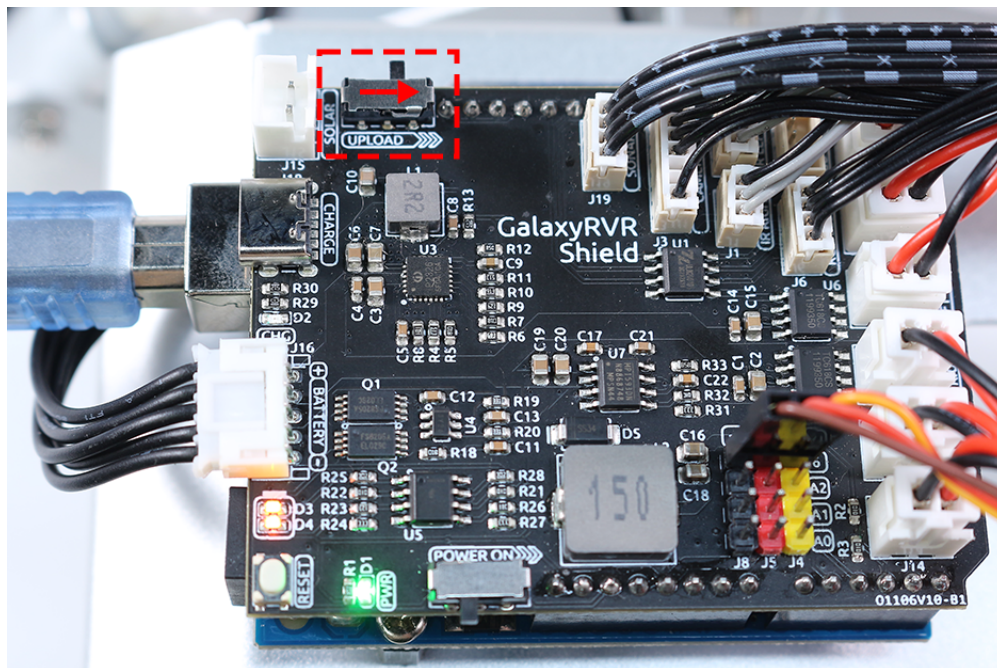
In our last adventure, we equipped our Mars Rover with “eyes” on its sides, creating a basic obstacle avoidance system. Yet, there’s a blind spot right in front – a challenge we’re ready to overcome!

Today, in this lesson, we’re going to give our Rover a new sense of “sight.” We’ll install an ultrasonic sensor module, acting as a pair of central eyes, which will help our rover detect obstacles directly ahead.

We will delve into the fascinating mechanics of ultrasonic waves and explore how they enhance our Rover’s ability to navigate complex terrains. With this addition, our Mars Rover will be more perceptive and agile, ready to embark on more ambitious exploratory missions.

Join us as we take a step further into this exciting STEAM journey, making our Mars Rover even more adept at exploring the uncharted territories!

Note: If you are learning this course after fully assembling the GalaxyRVR, you need to move this switch to the right before uploading the code.



3.7.1 Course Objectives

- Understand the principle of ultrasonic distance measurement.
- Learn how to use Arduino and ultrasonic module for distance measurement.
- Practice applying the ultrasonic module for obstacle avoidance on the Mars Rover model.

3.7.2 Materials Needed

- Ultrasonic Module
- Basic tools and accessories (e.g. screwdriver, screws, wires etc.)
- Mars Rover Model (Equipped with rocker-bogie system, main boards, motors, obstacle avoidance module)
- USB Cable
- Arduino IDE
- Computer

3.7.3 Course Steps

Step 1: Assemble the Ultrasonic Sensor Module

Now that we've got our eyes set on equipping our Mars Rover with a powerful new sense of "sight", it's time to put together the ultrasonic sensor module.

And there we have it! Our Mars Rover now has a fully-assembled ultrasonic sensor module, ready to help it navigate like never before. Are you excited to see how it changes our Rover's obstacle detection abilities? Let's dive right into it!

Step 2: Exploring the Ultrasonic Module

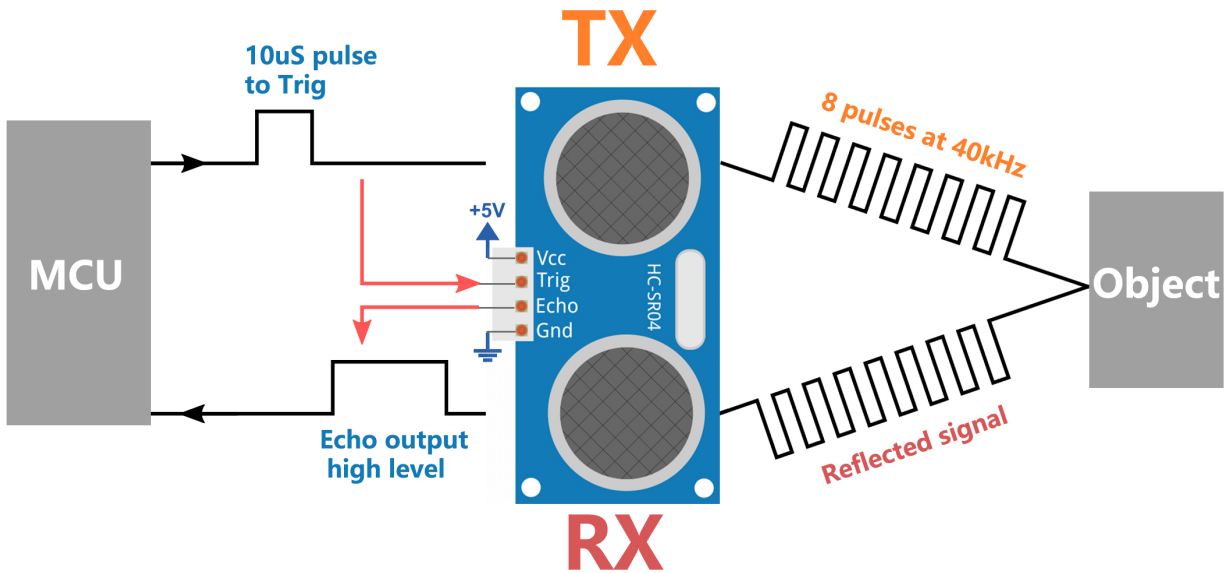
Let's get to know the HC-SR04, a powerful ultrasonic distance sensor. This tiny device can accurately measure distances from 2 cm up to 400 cm, all without touching a thing! Amazing, right? It's like having a superhero power! It can "see" the distances just by using sound waves, like how a bat navigates at night.

It uses four superpowers, or rather, four pins to do its magic:



- **TRIG (Trigger Pulse Input)** - It's the start button for our superhero. It tells our superhero, "Hey, it's time to send out a super sonic wave!"
- **ECHO (Echo Pulse Output)** - This is how our superhero listens to the echo of the sound wave it sent out.
- **VCC** - Even superheroes need some energy. We connect it to a 5V power supply.
- **GND** - It's the ground connection. Just like how superheroes need to stay connected to reality!

Imagine our superhero, the HC-SR04 Ultrasonic Sensor, playing a game of echo in the mountains.



- First, our superhero's brain, the MCU, says, "Ready, Set, Go!" by sending out a high-level signal for at least 10 microseconds to our superhero. This is like when we gather our energy before we yell into a valley.
- On hearing "Go!", our superhero shouts out loud 8 times very quickly. This super-sonic shout is sent out at a speed of 40 kHz. The superhero also starts a stopwatch and keeps an ear out for any returning shouts.
- If there is an obstacle in front, the shout will hit it and echo back. On hearing the echo, our superhero stops the stopwatch and notes the time. It also sends out a high-level signal to let the MCU know it heard an echo.
- Lastly, to find out how far away the obstacle is, our superhero uses a simple formula. It takes the time it recorded on the stopwatch, divides it by 2, and multiplies it by the speed of sound (340m/s). The result is the distance to the obstacle!

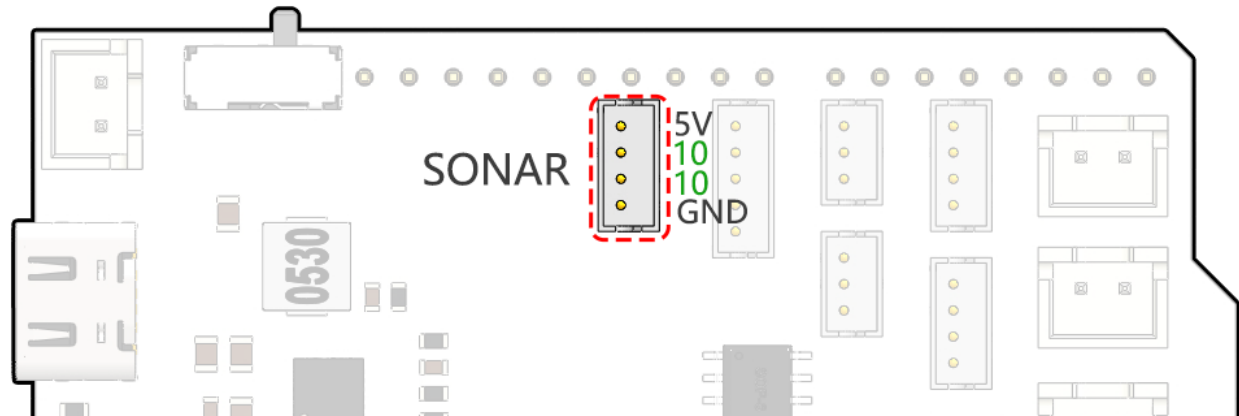
That's how our superhero sensor can figure out if there's an obstacle in its path and how far away it is. Amazing, isn't it? Next, we'll learn how we can use this superhero power in our Mars Rover!

Step 3: Coding Our Superhero Sensor

Having assembled our superhero sensor and understood how it uses its superpowers, it's time to put those powers into action! Let's write an Arduino sketch to allow our ultrasonic sensor to measure distances and then show us those measurements.

Here are the key steps that our superhero sensor will follow:

- We have connected both the TRIG and ECHO pins to pin 10 on the GalaxyRVR Shield. This allows us to control both signal transmission and reception of the ultrasonic module using a single Arduino pin.



```
// Define the pin for the ultrasonic module
#define ULTRASONIC_PIN 10
```

- **Preparation:** To start, we establish serial communication at a 9600 baud rate so we can see the superhero's measurements in our Serial Monitor.

```
void setup() {
  // Start the serial communication
  Serial.begin(9600);
}
```

- **Shout Out!:** We set the ULTRASONIC_PIN as an OUTPUT pin to send a 10-microsecond pulse, which is like commanding our ultrasonic superhero to emit a powerful shout (ultrasonic sound waves)!

```
// A 4ms delay is required, otherwise the reading may be 0
delay(4);

//Set to OUTPUT to send signal
pinMode(ULTRASONIC_PIN, OUTPUT);

// Clear the trigger pin
digitalWrite(ULTRASONIC_PIN, LOW);
delayMicroseconds(2);

// Trigger the sensor by sending a high pulse for 10us
digitalWrite(ULTRASONIC_PIN, HIGH);
delayMicroseconds(10);

// Set the trigger pin back to low
digitalWrite(ULTRASONIC_PIN, LOW);
```

- **Wait and Listen:** Set the ULTRASONIC_PIN as INPUT. This way, our superhero sensor is now ready to listen for the echo of its shout. If there is an obstacle in front, the shout will hit it and echo back. On hearing the echo, our superhero stops the stopwatch and notes the time. It also sends out a high-level signal to let the MCU know it heard an echo.

```
pinMode(ULTRASONIC_PIN, INPUT);
float duration = pulseIn(ULTRASONIC_PIN, HIGH);
```

- **Super Math:** With the echo returned, our sensor uses the speed of sound to calculate the distance to the object. We divide the total echo time by 2 because the time includes both the shout out and the wait for the echo.


```
float distance = duration * 0.034 / 2;
```

- **Report the Findings:** Our superhero sensor then reveals the result of its mission, printing the distance to the Serial Monitor for us to see.

```
// Print the distance to the serial monitor
Serial.print("The distance is: ");
Serial.print(distance);
Serial.println(" cm");
```

- **Rest & Ready:** Every superhero needs a rest, so our sensor takes a short pause before the next mission. This allows the sensor to “reset” before we ask it to start another measurement.

```
delay(200);
```

Here’s the complete code that turns our sensor into a superhero:

Step 4: Programming the Ultrasonic Module to Drive the Mars Rover

Now that we’ve equipped our Mars Rover with an ultrasonic sensor module, it’s time to program it to respond based on the sensor’s measurements.

- For easier reading, we have created a function called `readSensorData()`. This function encapsulates all the code required to read the distance from the ultrasonic sensor.

```
float readSensorData() {
    // A 4ms delay is required, otherwise the reading may be 0
    delay(4);

    //Set to OUTPUT to send signal
    pinMode(ULTRASONIC_PIN, OUTPUT);

    // Clear the trigger pin
    digitalWrite(ULTRASONIC_PIN, LOW);
    delayMicroseconds(2);

    // Trigger the sensor by sending a high pulse for 10us
    digitalWrite(ULTRASONIC_PIN, HIGH);
    delayMicroseconds(10);

    // Set the trigger pin back to low
    digitalWrite(ULTRASONIC_PIN, LOW);

    //Set to INPUT to read
    pinMode(ULTRASONIC_PIN, INPUT);

    // pulseIn returns the duration of the pulse on the pin
    float duration = pulseIn(ULTRASONIC_PIN, HIGH);

    // Calculate the distance (in cm) based on the speed of sound (340 m/s or 0.034_
    // cm/us)
    float distance = duration * 0.034 / 2;

    return distance;
}
```

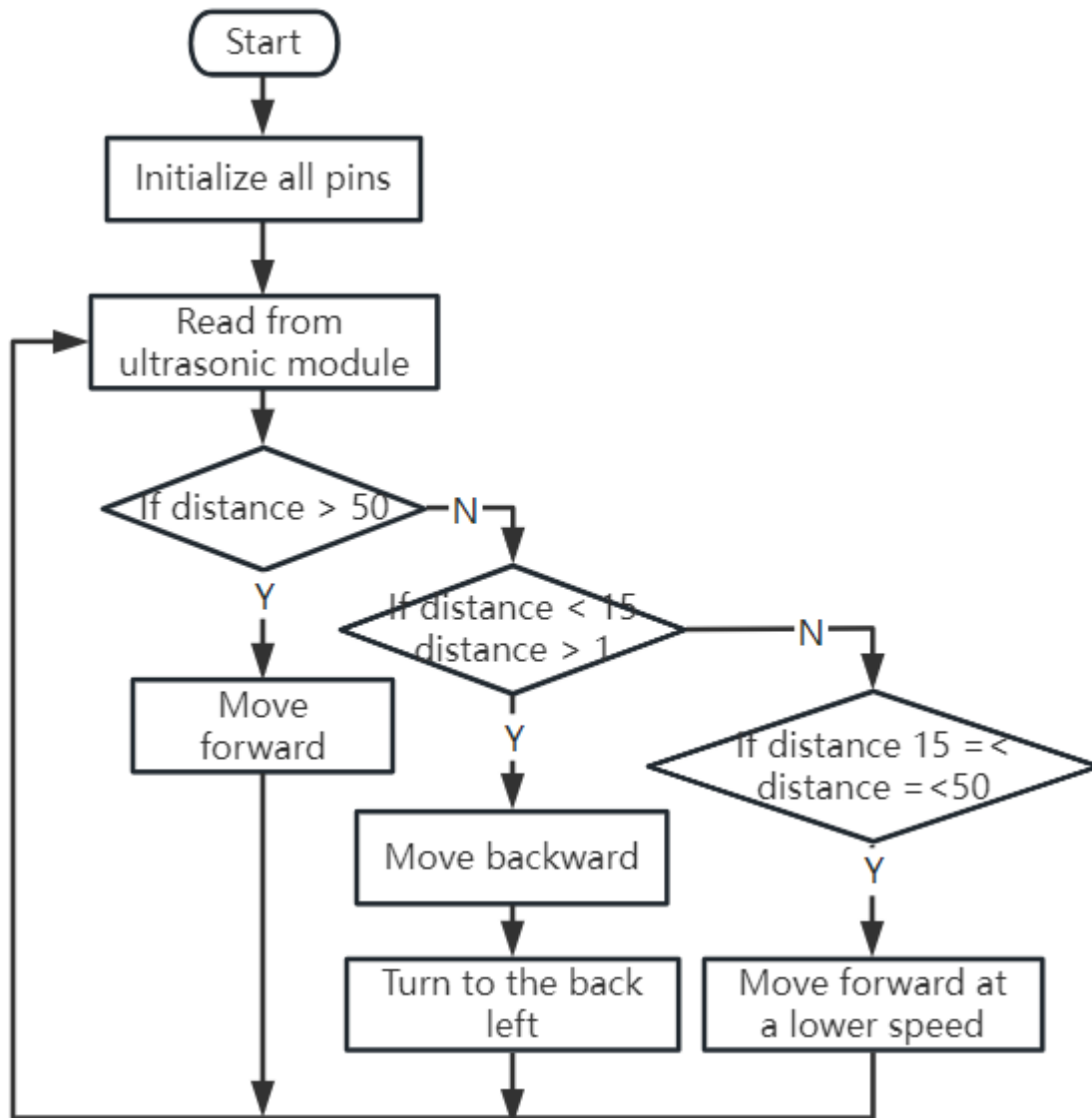

- In the loop() function, we call readSensorData() and stores its returned value in the distance variable.

```
float distance = readSensorData();
```

- Depending on this distance, the Rover will move forward, move backward, or stop.

```
// Control rover based on distance reading
if (distance > 50) { // If it's safe to move forward
  moveForward(200);
} else if (distance < 15) { // If there's an obstacle close
  moveBackward(200);
  delay(500); // Wait for a while before attempting to turn
  backLeft(150);
  delay(1000);
} else { // For distances in between, proceed with caution
  moveForward(150);
}
```

- If the path is clear (the obstacle is more than 50 cm away), our Rover boldly moves forward.
- And if an obstacle is getting close (less than 50 cm but more than 15 cm away), our Rover will move forward at a lower speed.
- If an obstacle is too close for comfort (less than 15 cm away), the Mars rover will move backward and then turn to the left.



Below is the complete code. You can upload this code to the R3 board and see if it achieves the desired effect. You can also modify the detection distance based on the actual environment to make this obstacle avoidance system more perfect.

By leveraging these enhanced capabilities, the Mars Rover would be better equipped to identify potential obstacles in its path, measure distances accurately, and make informed decisions to navigate around them. This would significantly reduce the likelihood of collisions or other hazards that could hinder the rover's exploration mission.

With its super-senses, the Mars Rover can operate with greater confidence and efficiency, enabling it to delve deeper into the mysteries of Mars and gather valuable scientific data for researchers back on Earth.

Step 5: Summary and Reflection

In this lesson, we delved into the workings of ultrasonic waves and how to translate their return time to the sensor into measurable distance via coding.

Subsequently, we leveraged ultrasonic waves to devise an obstacle-avoidance system. This particular system varies its responses based on the distance to the impending obstacle.

Now, let's prompt some introspection on this lesson through a handful of questions:

- How does an ultrasonic module detect distance? Can you elucidate the underlying concept?
- How does the obstacle-avoidance system of this lesson differ from that of the previous one? What are their respective advantages and drawbacks?
- Is it feasible to amalgamate these two obstacle-avoidance systems?

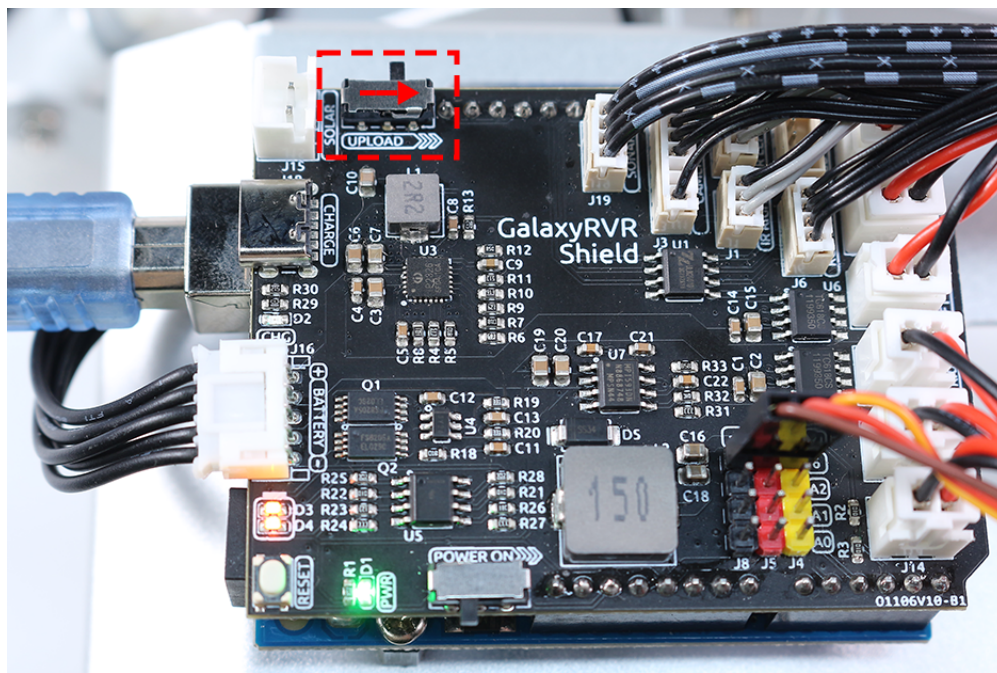
Reflecting upon these queries will aid in solidifying our comprehension and prompt us to contemplate the application of these concepts to other projects. Looking forward to our next venture!

3.8 Lesson 8 Advanced Obstacle Avoidance and Intelligent Following System

In today's lesson, we're going to push our STEAM skills a step further. We'll combine an obstacle avoidance module with an ultrasonic sensor to create an advanced obstacle avoidance system. We'll also implement an intelligent following system to our Rover.

By the end of this lesson, our Mars Rover will not only be able to avoid obstacles in its path but also follow moving objects. Imagine having a mini robotic pet following you around! Exciting, isn't it? So let's get started.

Note: If you are learning this course after fully assembling the GalaxyRVR, you need to move this switch to the right before uploading the code.



3.8.1 Course Objectives

- Learn how to combine obstacle avoidance modules with an ultrasonic module for improved navigation.
- Understand the principles and functionalities behind an advanced obstacle avoidance system.
- Learn how to implement an intelligent following system in the Mars Rover.

3.8.2 Course Materials

- Mars Rover model (the one we built in previous lessons)
- USB Cable
- Arduino IDE
- Computer
- And of course, your creative mind!

3.8.3 Course Steps

Step 1: Understanding the Concept

The obstacle avoidance module, as the name suggests, helps our Rover avoid obstacles. It detects obstacles by transmitting an infrared signal and then receiving the signal reflected back from the object. If there is an obstacle in front of the module, the infrared signal is reflected back, and the module detects it.

Now, adding an ultrasonic sensor to the mix improves this system. Ultrasonic sensors measure distance by sending out a sound wave at a specific frequency and listening for that sound wave to bounce back. By recording the elapsed time between the sound wave being generated and the sound wave bouncing back, it is possible to calculate the distance between the sensor and the object.

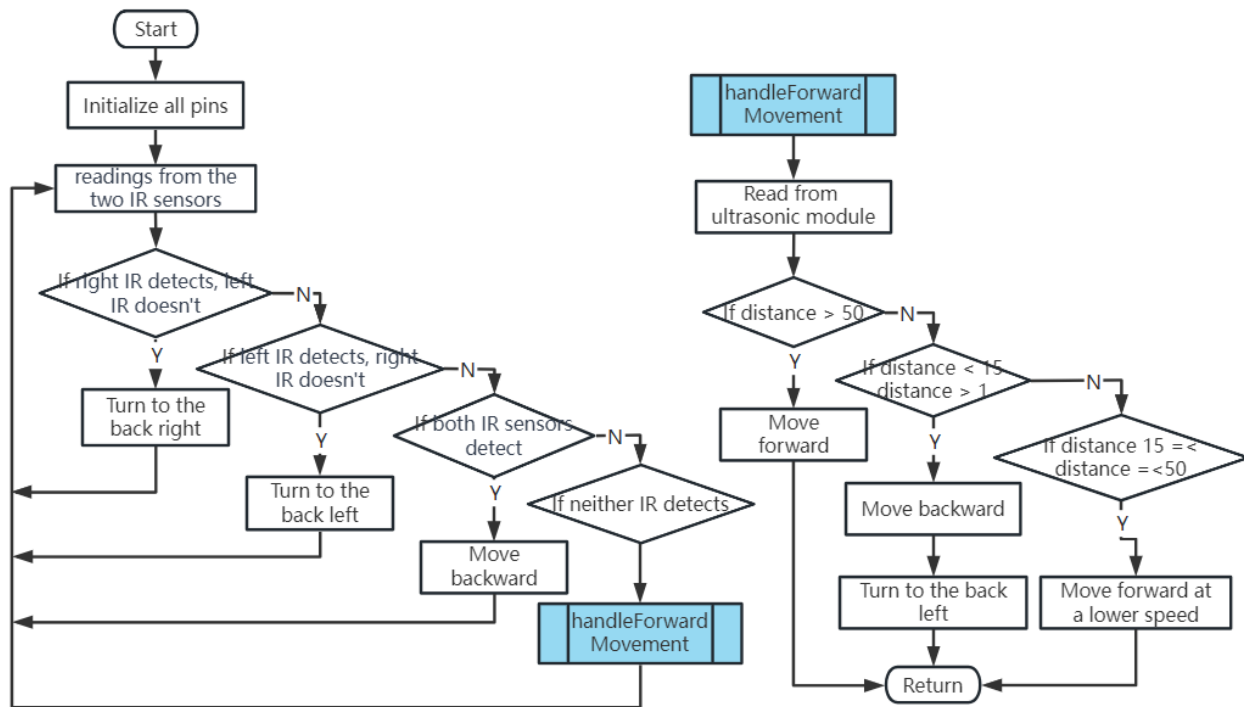
Combining these two gives us a reliable, efficient, and versatile obstacle avoidance system!

Step 2: Constructing Advanced Obstacle Avoidance Systems

In our previous lessons, we've learned the basics of obstacle avoidance using infrared sensors. We've also explored how an ultrasonic module works. Now, we are going to bring all these pieces together and build an advanced obstacle avoidance system!

Our enhanced Mars Rover will now use both ultrasonic and infrared sensors to navigate its surroundings.

Let's envision how the infrared and ultrasonic modules should work together. To help clarify our logic, let's use a flowchart. Learning how to create flowcharts is an invaluable step in our coding journey as it can help you clarify your thoughts and systematically outline your plan.



Now let's turn this flowchart into actual code to bring our Rover to life.

Note that the `handleForwardMovement()` function is where we've integrated the behavior of the ultrasonic sensor. We read the distance data from the sensor and based on this data, we decide the movement of the Rover.

After uploading the code to your R3 board, it's time to test the system. Make sure the Rover can detect and avoid obstacles efficiently. Remember, you may need to adjust the detection distance in the code based on your actual environment to perfect the system.

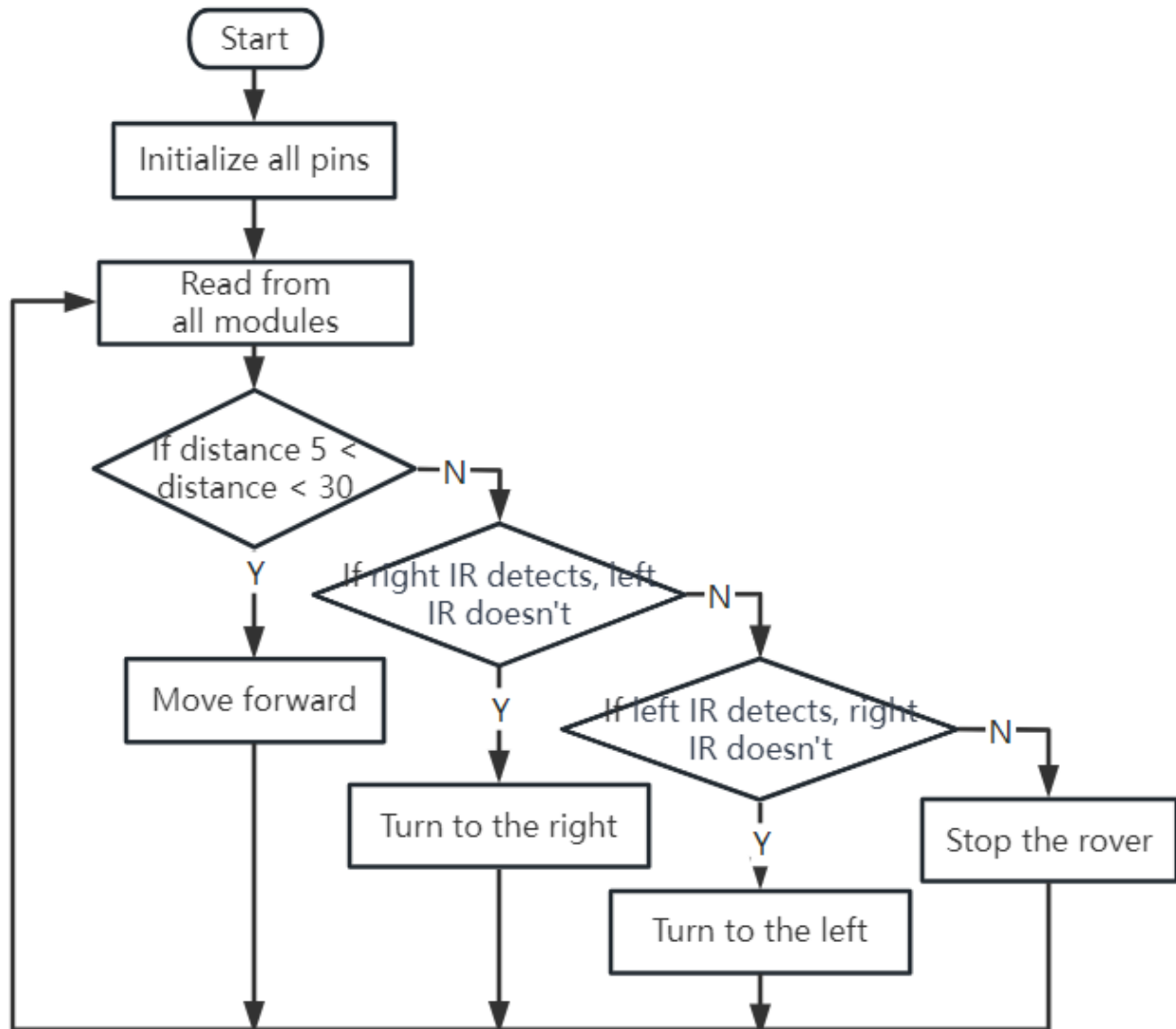
Step 3: Coding the Intelligent Following System

With our Rover now capable of avoiding obstacles, let's enhance it further by making it follow objects. Our goal is to modify our existing code to make the Rover move towards a moving object.

Ever wondered about the differences between a following system and an obstacle avoidance system?

The key here is that in a following system, we want our Rover to move in response to detected objects, while in an obstacle avoidance system, we're looking to steer clear of detected objects.

Let's visualize the desired workflow:



- If the ultrasonic sensor detects an object within 5-30 cm, our Rover should move towards it.
- If the left IR sensor detects an object, our Rover should take a left turn.
- If the right IR sensor detects an object, our Rover should take a right turn.
- In all other cases, our Rover should remain stationary.

Now, it's time for us to complete the code.

Once the code is completed, test if the Rover follows your movements.

As we did with the obstacle avoidance system, it will be crucial to test our following system and troubleshoot any issues that may arise. Ready to start?

Step 4: Summary and Reflection

Today, you've accomplished something amazing. You combined different modules and concepts to create an advanced obstacle avoidance and following system for your Mars Rover. Remember, learning does not end here - keep exploring, innovating, and applying your newfound skills to other projects.

Remember to always reflect on your learning process. Think about the following:

- Why do you think we prioritized the obstacle avoidance module before the ultrasonic sensor in our obstacle avoidance system, and vice versa in the following system?
- How would the outcome differ if we were to swap the order in which these modules are checked in the code?

Challenges and problems are an integral part of the STEAM learning process, offering valuable opportunities for improvement. Don't shy away from troubleshooting - it's a powerful learning tool in itself!

As you continue on your journey, know that every obstacle you overcome brings you one step closer to mastering your STEAM skills. Keep going and enjoy the journey!

3.9 Lesson 9: Lighting the Way with RGB LED Strips

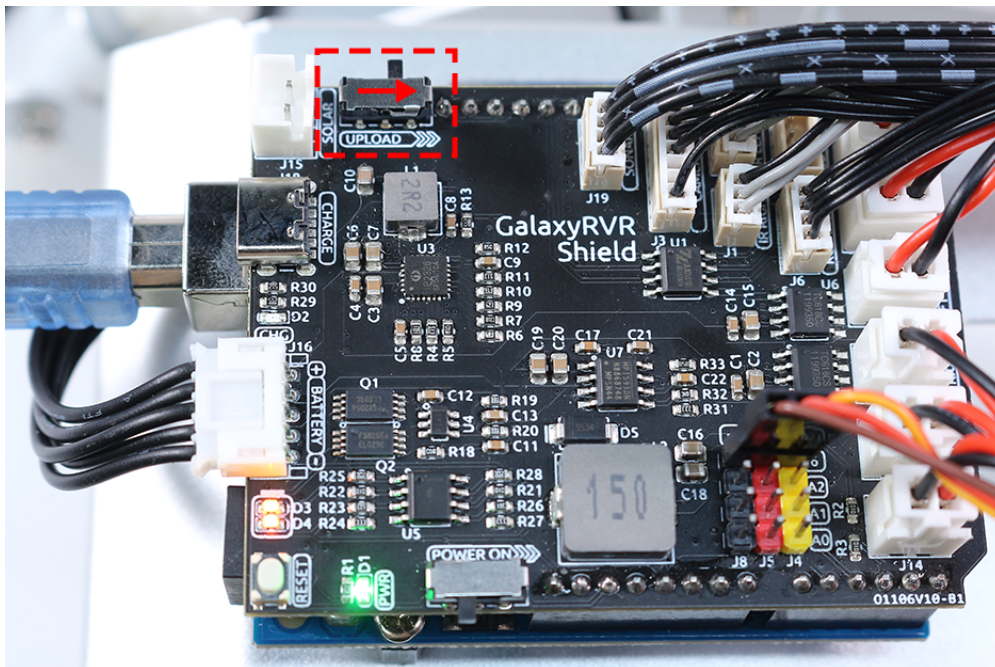
In our journey so far, we've transformed our Mars Rover into a smart explorer, capable of skilfully manoeuvring around obstacles. It's become quite adept at navigating the Mars-like terrains we've set up for it.

But, what if we could add a bit of flair to its practicality? Let's give our Rover the ability to express itself through a spectacle of colors and light. We're talking about incorporating RGB LED strips - a cool feature that would allow our Rover to illuminate its path, even in the darkest conditions.

Picture this - the Rover leaves a trail of color-coded signals, making it easier for us to understand its moves. A green glow when it's on the go, a stern red when it halts, or a flashy yellow during those swift turns. It could even light up in an array of colors just for the sheer fun of it!

Our goal in this lesson is to understand the principles of RGB LED strips, learn to control their color and brightness, and then synchronize this with the Rover's movements. By the end, our Mars Rover will be more than a machine. It'll be a luminous, color-changing entity, leading the way in the vast Martian landscape!

Note: If you are learning this course after fully assembling the GalaxyRVR, you need to move this switch to the right before uploading the code.



3.9.1 Objective

- Understand the working principles and applications of RGB LED strips.
- Learn how to use Arduino programming to control the color and brightness of RGB LED strips.
- Practice installing and using RGB LED strips on the Mars Rover model as indicators.

3.9.2 Materials Needed

- RGB LED Strips (each strip has 8 RGB LEDs, a total of two strips)
- Basic tools and accessories (e.g. screwdriver, screws, wires etc.)
- Mars Rover Model (Equipped with rocker-bogie system, main boards, motors, obstacle avoidance module, ultra-sonic module)
- USB Cable
- Arduino IDE
- Computer

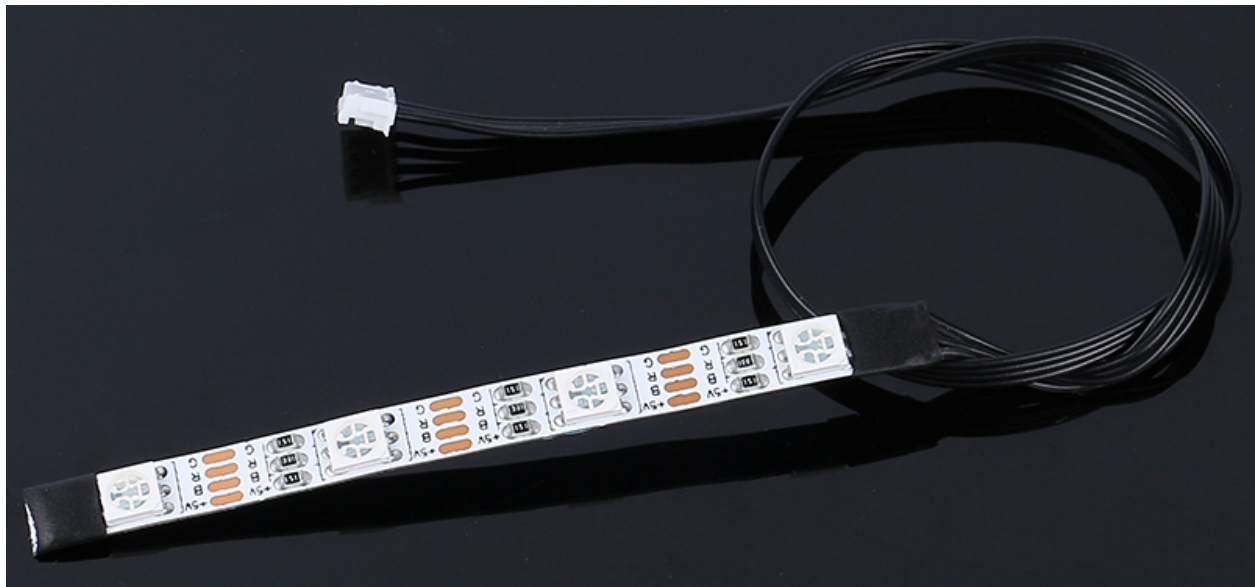
3.9.3 Course Steps

Step 1: Install the RGB LED Strips on the Mars Rover

Now, fix the two RGB light strips to the bottom sides of the car. They are controlled by a single set of pins, so there is no need to differentiate during the wiring process.

Step 2: Explore the Magic of Light with RGB LED Strips

Do you remember the last time you saw a rainbow? How it made the sky colorful with seven vibrant hues? How would you like to create your own rainbow, right here in our little Martian rover? Let's dive into the magic of light with RGB LED strips!

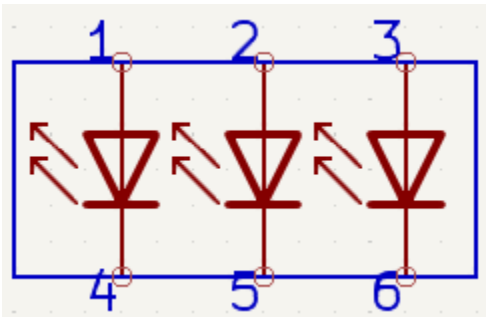


You might notice that our RGB LED Strip has four pins labeled as follows:

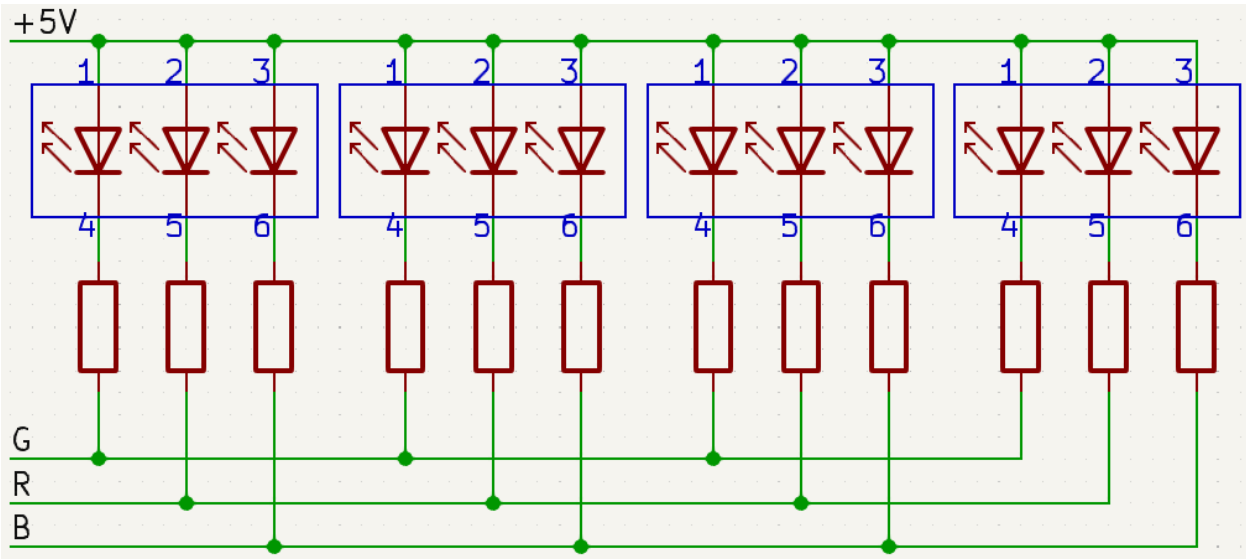
- **+5V**: This is the common “positive” end or the “anode” of the three tiny light bulbs (LEDs) inside our strip. It needs to connect to DC 5V, a kind of electric juice that powers our tiny bulbs!
- **B**: This is the “negative” end or the “cathode” of the blue LED.
- **R**: This is the “cathode” of the red LED.
- **G**: This is the “cathode” of the green LED.



Do you remember the three primary colors - Red, Blue, and Green - that we learned in our art class? Just like an artist mixes these colors on his palette to create new shades, our strip contains 4 “5050” LEDs that can mix these primary colors to create virtually any color! Each “5050” LED is like a tiny art studio that houses these three colored bulbs.



These tiny art studios are then connected in a smart way on a flexible circuit board - kind of like a mini electric highway! The “positive” ends of all LEDs (anodes) are connected together, while the “negative” ends (cathodes) are connected to their corresponding color lanes (G to G, R to R, B to B).



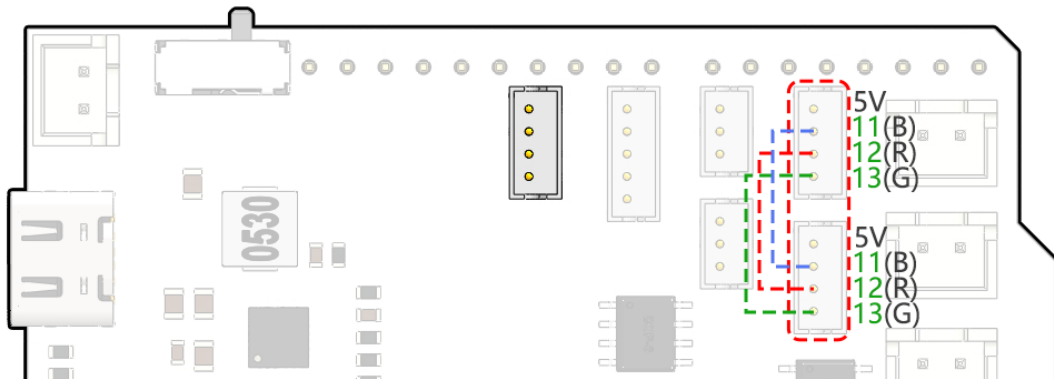
And the coolest part? With our command, all the LEDs on this strip can change their colors at once! It's like having our own light orchestra at the tip of our fingers!

So let's get ready to play some light music! In our next step, we'll learn how to control these LEDs to display the colors we want. It will be like conducting a symphony of light!

Step 3: Light Up the Show - Coding to Control the RGB LED Strips

We've stepped into the realm of colors, it's time to bring our Mars Rover to life. Brace yourself to paint the darkness with a spectrum of colors using RGB LED strips. Think of this as a chance to transform your Mars Rover into a mobile disco party!

- Before we dive into the fun part, let's understand that even though we have two LED strips, they are both controlled by the same set of pins. Think of it as having two dazzling dancers moving in perfect synchronization!



- It's time to summon our coding magic. We're going to initiate our pins with the Arduino code.

```
#include <SoftPWM.h>

// Define the pin numbers for the RGB strips
const int bluePin = 11;
const int redPin = 12;
const int greenPin = 13;
```

- With our pins in place, we'll now use the `SoftPWMSet()` function to control these pins. To make the RGB strip display red, we turn the red LED on and switch off the others.

```

void setup() {
  // Initialize software-based PWM on all pins
  SoftPWMBegin();
}

void loop() {
  // Set the color to red by turning the red LED on and the others off
  SoftPWMSet(redPin, 255); // 255 is the maximum brightness
  SoftPWMSet(greenPin, 0); // 0 is off
  SoftPWMSet(bluePin, 0); // 0 is off
  delay(1000); // Wait for 1 second
}

```

In the above code, we've only demonstrated how to display a single color.

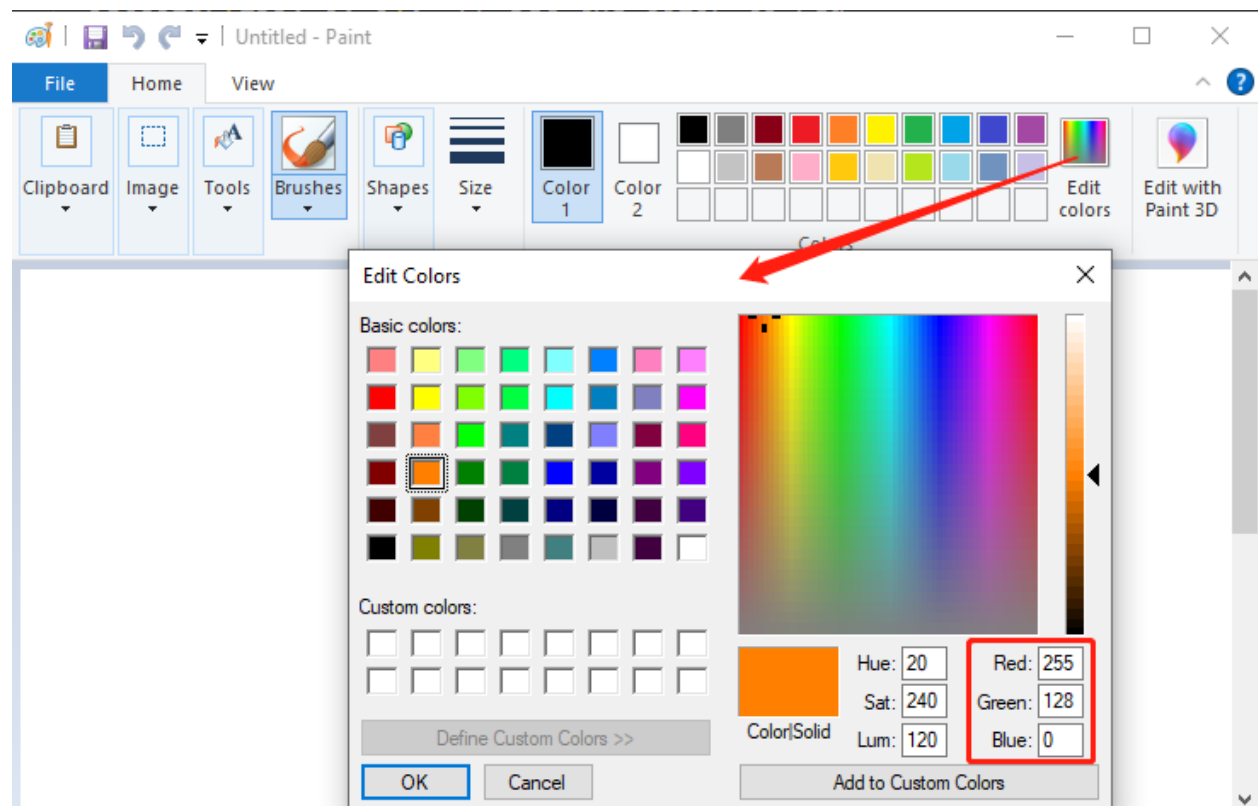
If we were to showcase a variety of colors using this method, the code could become quite cumbersome. Therefore, to make our code more concise and maintainable, we can create a function to assign PWM values to the three pins. Then, within the `loop()`, we can easily set a multitude of colors.

After uploading the code to your R3 board, you may find that the orange and yellow colors seem a bit off. This is because the red LED on the strip is relatively dim compared to the other two LEDs. Thus, you'll need to introduce offset values in your code to correct this color discrepancy.

Now, the RGB LED strip should be able to display the correct colors. If you still notice discrepancies, you can manually adjust the values of `R_OFFSET`, `G_OFFSET`, and `B_OFFSET`.

Feel free to experiment and display any color of your choosing on the LED strip. All you need to do is fill in the RGB values for the color you want.

Here's a tip: You can use the Paint tool on your computer to determine the RGB values of your desired color.



Now that we've mastered the art of color-setting, in the next step, we'll integrate these dazzling displays with the movements of the Mars Rover. Exciting times ahead!

Step 4: Move the Rover with Color Indication

Now, we'll add color indications to the movements of the Mars Rover. For instance, we can use green for forward, red for backward, and yellow for turning left or right.

To do this, we will add a control mechanism in our code that sets the color of the LED strip based on the Rover's movement. This will involve modifying our Rover control code to include our color control functions.

Let's see an example of how we can do this:

Within the `loop()` function, we commanded the Rover to perform a series of actions by calling different functions. Each action had its corresponding color display - green for moving forward, red for moving backward, and yellow for turning. This color display feature was brought to life using the `setColor()` function, which manipulated the brightness of each RGB color channel.

For the stop action, we introduced an engaging element - a breathing effect with a red and blue light. This was achieved by cyclically adjusting the brightness of the red and blue channels within the `stopMove()` function. As such, upon stopping, the LED strip transitioned colors between red and blue, creating a dynamic visual effect.

Now, our Mars Rover now possesses its own vibrant color effects, leaving behind a trail of color-coded signals, each representing a unique movement.

Through this project, we've discovered how STEAM subjects can amalgamate to breathe life into an otherwise ordinary machine, turning it into a vibrant, interactive, and fun learning tool.

Step 5: Summary and Reflection

In today's lesson, we delved into the world of RGB LED strips, exploring how to manipulate them to display a vivid array of colors. These brilliant hues breathed new life into our Mars Rover, transforming it from a mere machine into a vibrant spectacle.

Now, I invite you to ponder - If it was you in the driver's seat, how would you utilize these colors to enhance your Mars Rover? What unique effects would you want it to exhibit?

Moreover, through the process, I hope you had a hands-on understanding of how diverse STEAM concepts can be interwoven in an engaging project, providing you with a broader perspective of its practical applications.

See you in our next exciting adventure!

3.10 Lesson 10: Exploring the Mars Rover Visual System - Servo and Tilt Mechanism

Welcome back, young explorers! In today's adventure, we are going to delve into the fascinating world of the Mars Rover's visual system. Just like our eyes and neck work together to help us see and navigate our surroundings, our Rover too needs a similar system to navigate the treacherous Martian landscape. And that's exactly what we are going to build today!

The visual system of our Rover has two main parts: a camera that acts as its "eyes", and a tilt mechanism that acts like a "neck", allowing it to look up and down. By the end of this lesson, we'll give our Rover the ability to "see" and "nod"!

First, we'll build the tilt mechanism - a device that will hold our Rover's camera and let it rotate vertically. It's like giving our Rover a neck, so it can nod its "head" or camera up and down!

Next, we'll learn about the servo, the tiny yet powerful "muscle" that moves our tilt mechanism. We'll understand how it works and how we can control it using Arduino programming.

Just as our neck muscles move our head so our eyes can get a better view, the servo will move the tilt mechanism so the Rover's camera can better survey the Martian landscape.

So, buckle up, explorers, let's start our mission to equip our Rover with its very own visual system!

3.10.1 Objective

- Practice installing and operating the tilt mechanism on the Mars Rover model.
- Understand the principles of operation and application of servo.
- Learn how to control servo movement through Arduino programming.

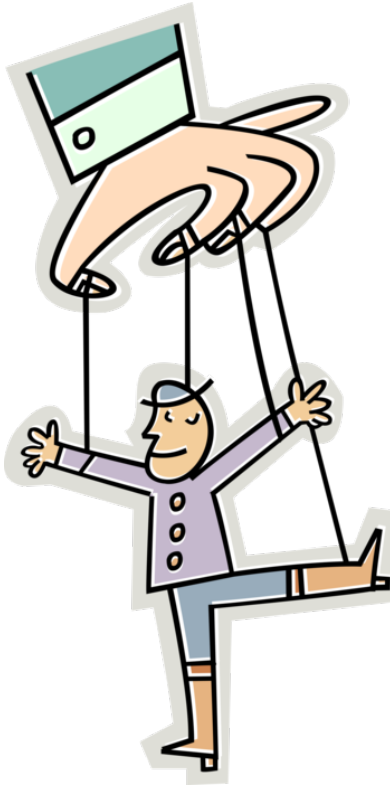
3.10.2 Materials

- Arduino UNO development board
- Servo
- Gimbal and camera
- Mars Rover model (already equipped with TT motor, suspension system, ultrasonic and infrared obstacle avoidance modules, RGB LED strip)
- Arduino IDE
- Computer

3.10.3 Steps

Step 1: What is a Servo?

Have you ever watched a puppet show? If you have, you might have marveled at how the puppeteer can make the puppet's arms, legs, and head move so smoothly, just by pulling on some strings. In a way, servo motors are like our puppeteers.



Servo motors are special type of motors that don't just spin around and around like a wheel. Instead, they can move to a specific position and hold that position. Imagine if you're playing a game of Simon says, and Simon says, "Raise your arm to a 90-degree angle!" You can do it, right? That's because, like a servo, you can control exactly how much to move your arm.



- Brown Line: GND
- Orange Line: Signal pin, connect to the PWM pin of main board.
- Red wire: VCC

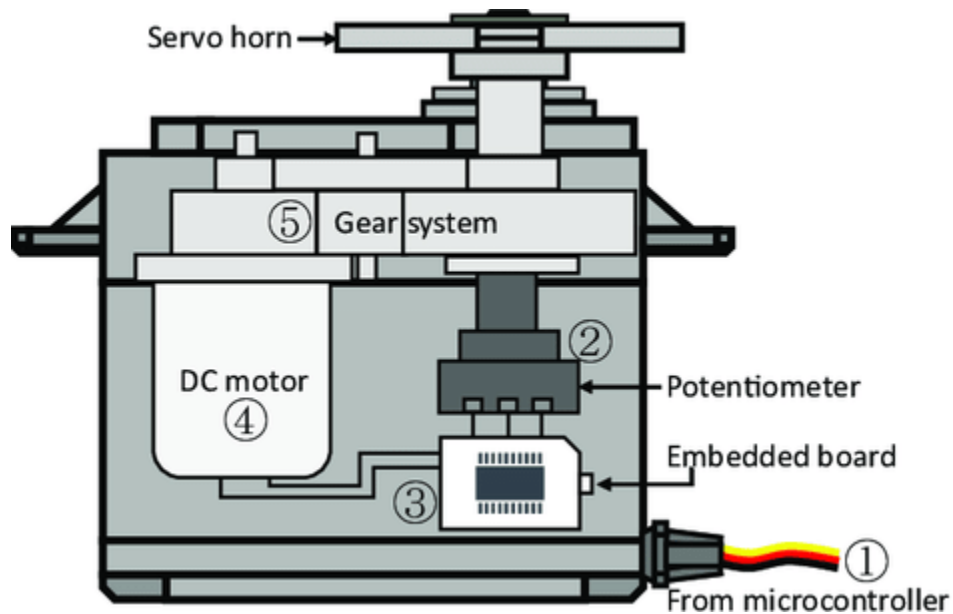
Just like you can control your arms to move to specific positions, we can use servo motors to control the exact position of objects in our projects. In our Mars Rover, we will use a servo to control the up and down movement of our tilt mechanism, just like how you can nod your head up and down.

In the next step, we will go on a fascinating journey inside a servo motor to understand how it works. Excited? Let's go!

Step 2: How does a Servo Work?

So how does a servo work its magic? Let's go on an exciting journey inside a servo!

If we were to peek inside a servo, we would see a few parts. At the heart of a servo is a regular motor, very similar to the motors that spin our Mars Rover's wheels. Wrapped around the motor, there is a big gear that is connected to a smaller gear on the motor shaft. This is how the motor's fast, circular motion gets transformed into slower but stronger motion.



But that's not what makes a servo special. The magic happens in a tiny piece of electronics called a "potentiometer" and the "control circuitry". Here's how it works: when the servo moves, the potentiometer turns and changes its resistance. The control circuitry measures this change in resistance and knows exactly what position the servo is in. Clever, isn't it?

To control a servo, we send it a special kind of signal called a "pulse-width modulation" signal or PWM. By changing the width of these pulses, we can control exactly how much the servo moves and hold it in that position.

In the next step, we'll learn how to control a servo using an Arduino. Ready for some magic spells in the form of code? Let's go!

Step 3: Controlling a Servo using Arduino

Alright, explorers, now that we know how a servo works, let's learn how to control it using our magic wand, the Arduino!

Controlling a servo is like giving it directions. Remember the pulse-width modulation (PWM) signals we mentioned earlier? We are going to use those to tell the servo where to move.

Luckily, Arduino makes this task easy for us with a built-in library called `Servo`. With this library, we can create a `Servo` object, attach a pin to it (the pin that our servo is connected to), and then use a simple command, `write()`, to set the angle.

Here's a snippet of what the code looks like:

```
#include <Servo.h>
```

(continues on next page)

(continued from previous page)

```
Servo myServo; // create a servo object

void setup() {
  myServo.attach(6); // attaches the servo on pin 6
}

void loop() {
  myServo.write(90); // tell servo to go to 90 degrees
}
```

In this code, myServo is our Servo object, attach(6) tells the Arduino that our servo is connected to pin 6, and write(90) tells the servo to move to 90 degrees.

Great job, explorers! You've just learned how to control a servo motor with Arduino. You can experiment with different angles too!

Step 4: Assemble the Visual System

You're now ready to assemble the visual system of our Rover.

Note:

- When inserting the ESP32 CAM into the Camera Adapter, be aware of its orientation. It should align properly with the ESP32 Adapter.



Step 5: Understanding the Limits of the Tilt Mechanism

Even though servo is designed to rotate between 0 and 180 degrees, you may notice that it stops responding beyond a certain point (let's say after 150 degrees). Have you ever wondered why this happens? Let's explore this mystery together in our next adventure!

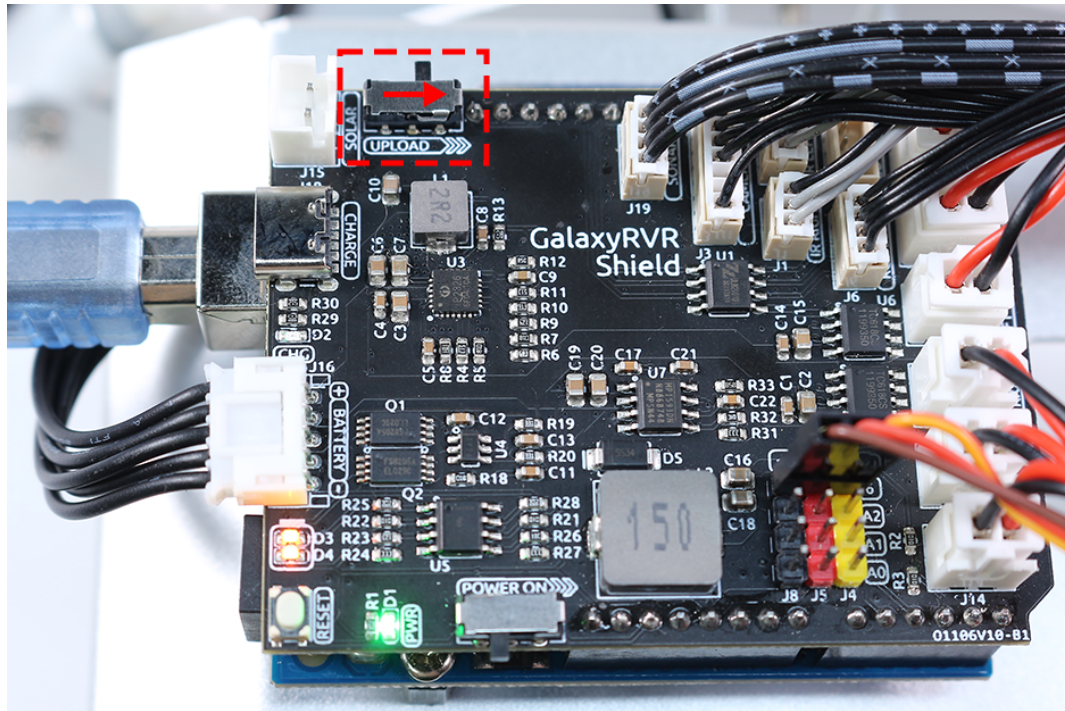
Can you imagine a bird trying to bend its neck too much that it hits its own body and can't move any further? Our Rover's tilt mechanism faces a similar situation. As the servo moves the mechanism downwards, it can bump into the body of our Rover and can't go beyond a certain angle.

If we try to force it to move beyond this point by writing an unreachable angle in our code, our little servo birdie can

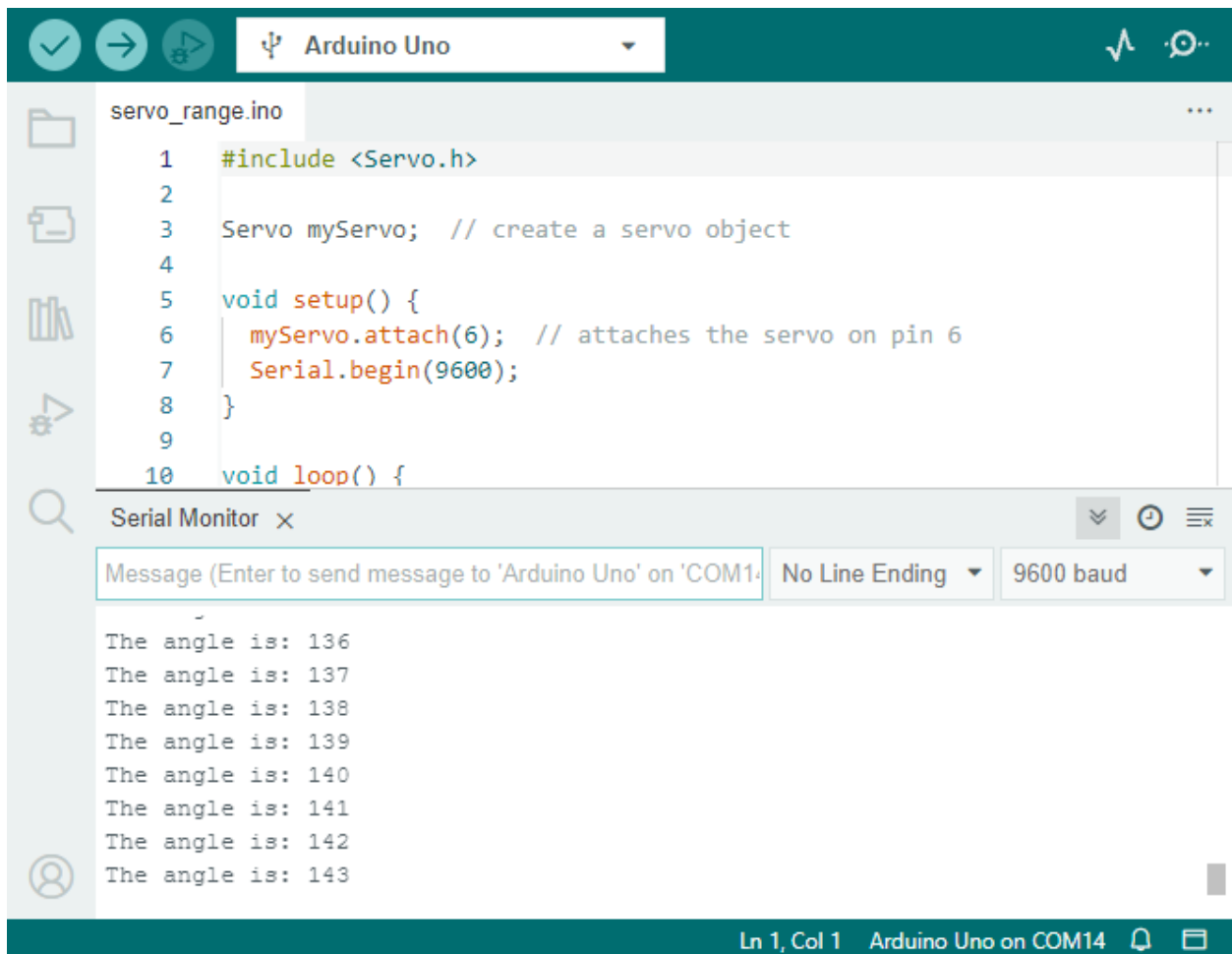
get stuck and even damage itself! We don't want that to happen, do we? So, let's understand its movement limitations with a little experiment.

We use a for loop to rotate the servo from 0 to 180 degrees while keeping a note of the angle in the Serial Monitor.

- The ESP32-CAM and the Arduino board share the same RX (receive) and TX (transmit) pins. So, before uploading the code, you'll need to first release the ESP32-CAM by slide this switch to right side to avoid any conflicts or potential issues.



- After we upload this code, open the **Serial Monitor**. If no information appears, press the **Reset button** on the GalaxyRVR shield to run the code again.
- You will see the servo rotate, and the Serial Monitor will display the angle.



On my Rover, the tilt mechanism could go up to around 140° before it hit the body of the Rover and couldn't go any further.

So, explorers, always remember to respect the limits of your rover to keep it safe and functioning!

Step 6: Sharing and Reflection

Well done, explorers! Today, you've not only built a tilt mechanism for your Rover but also understood how to control a servo to move it around. That's a big step forward in our Mars Rover mission.

Now, let's share our experiences and reflect on what we've learned.

Did you encounter any challenges while setting up the tilt mechanism or programming the servo? How did you overcome them?

Remember, every challenge we overcome makes us smarter and our Rover better. So don't hesitate to share your stories, ideas, and solutions. You never know, your innovative solution might help a fellow explorer in their journey!

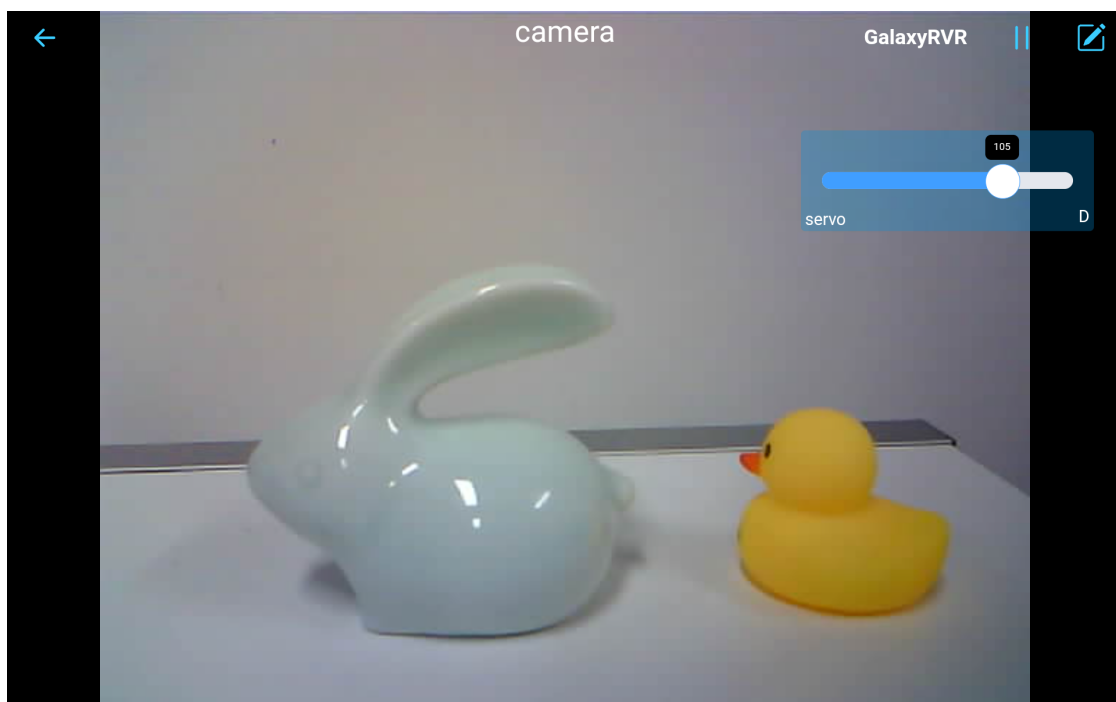
3.11 Lesson 11: Exploring the Mars Rover Visual System - Camera and Real-time Control

Welcome back, young explorers! In the last lesson, we equipped our Mars Rover with the ability to “nod” using a tilt mechanism. Now, it’s time to give our Rover “eyes” - the camera!

In this thrilling journey, we’ll dive into the setup of the Rover’s camera system. You’ll learn how to relay the visuals captured by the Rover’s camera to a web page, so you can see exactly what the Rover sees, in real time. Imagine the excitement of experiencing the Martian landscape from the Rover’s perspective!

The excitement continues as we also introduce the SunFounder Controller app. This application allows us to get a live feed of the Rover’s view as it navigates around, and we can control the tilt mechanism directly from our smartphones or tablets. It’s like having a remote control with a built-in screen!

This offers an even more interactive and engaging experience with our Rover. Stay tuned for more adventures!



3.11.1 Learning Goals

- Understand how to establish a WiFi connection with the ESP32 CAM.
- Learn how to see exactly what the Rover sees, in real time.
- Learn how to use the SunFounder Controller app to create a virtual remote and control the Mars Rover.

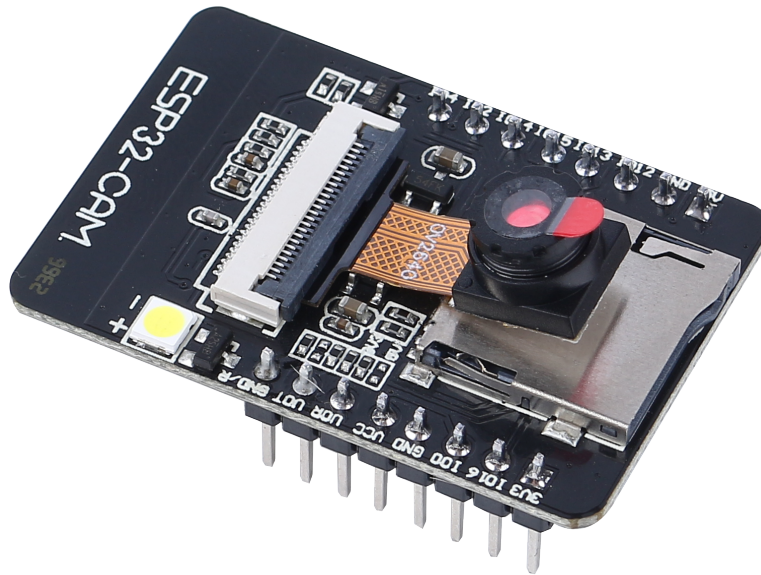
3.11.2 Materials needed

- Mars Rover model (equipped with all components)
- Arduino IDE
- Computer
- Tablet or smartphone with SunFounder Controller app installed

3.11.3 Course Steps

Step 1: Introduction to ESP32 CAM

In our previous adventure, we have equipped our Mars Rover with a pair of “eyes” by integrating the ESP32 CAM. Today, we’re going to learn more about it and actually make it “see”.



The ESP32 CAM, acting like the eyes of our Rover, is a small yet powerful module. Not only does it integrate Wi-Fi and Bluetooth functionalities, it also comes with a compact camera. This camera helps our Rover capture images of its surroundings.

Just like we use our eyes to observe our environment, the ESP32 CAM can “see” what lies ahead for the Rover, then send these visual data to our smartphone or computer. This allows us to see everything the Rover sees in real-time!

It’s as if we’re piloting the Rover directly, observing not just the Rover itself, but also the world it explores! Incredible, isn’t it? So, let’s dive deeper into it...

Step 2: Programming the Rover’s Camera and Viewing the Feed

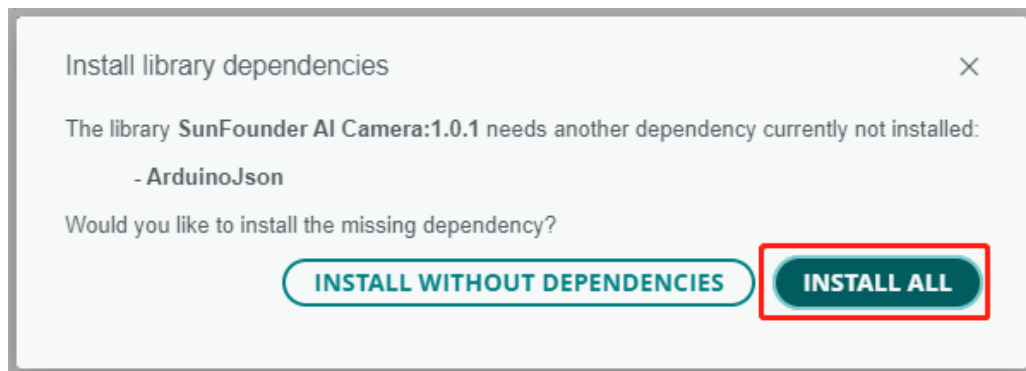
After fitting the ESP32-CAM to our Rover, we now need to breathe life into it. To do so, we will use the Arduino IDE to write a program that will control the camera, allow it to connect to WiFi, and stream the visuals it captures.

Here's how we can do it:

1. Install the SunFounder AI Camera library.
 - Open the Arduino IDE's **Library Manager**, search for "SunFounder Camera", and click **INSTALL**.



- A pop-up window will appear for the installation of library dependencies. Click **INSTALL ALL** and wait for the process to complete.



2. In the Arduino IDE, input the following code.

Regarding the variables NAME, TYPE, and PORT in the code, let's not delve into them at this point. They will come into play in our next step. Just keep in mind that these variables will be important in our upcoming journey to establish a real-time video feed from our Mars Rover.

Notice we have two connection modes in the code - **AP** mode and **STA** mode. You can decide which one to use based on your specific needs.

- **AP Mode:** In this mode, the Rover creates a hotspot (named as GalaxyRVR in our code). This allows any device like a mobile phone, tablet, or laptop to connect to this network. This is especially useful when you want to control the Rover remotely under any circumstances. However, note that this would make your device temporarily unable to connect to the Internet.

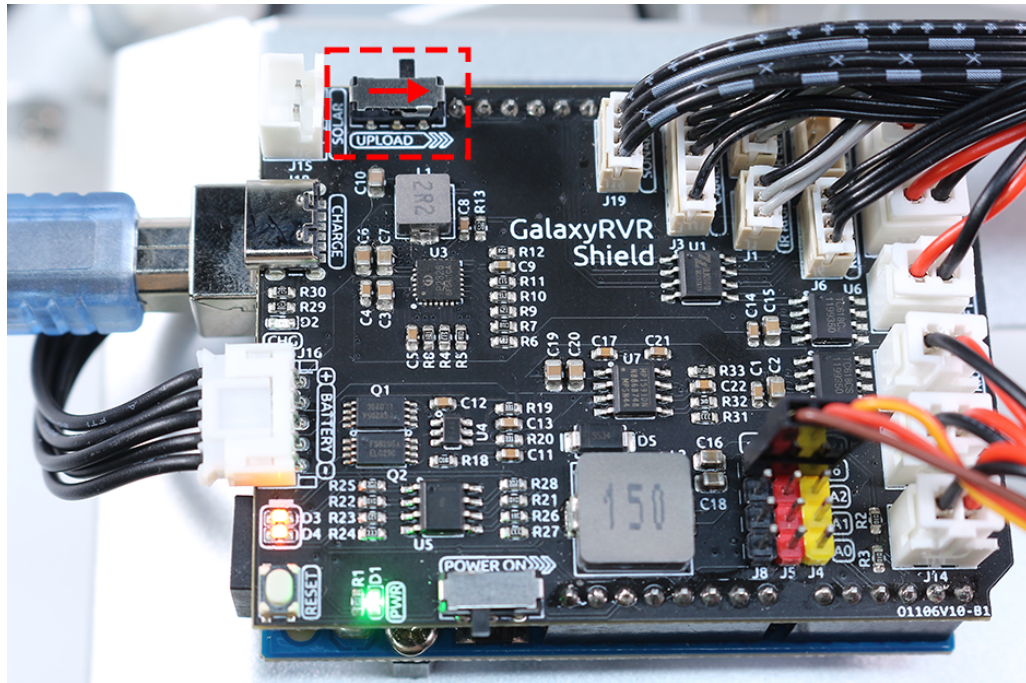

```
// AP Mode #define WIFI_MODE WIFI_MODE_AP #define SSID "GalaxyRVR" #define
PASSWORD "12345678"
```

- **STA Mode:** In this mode, the Rover connects to your home WiFi network. Remember that your controlling device (like a mobile phone or tablet) should also be connected to the same WiFi network. This mode allows your device to keep its regular internet access while controlling the Rover, but limits the Rover's operational range to your WiFi coverage area.

```
// STA Mode
#define WIFI_MODE WIFI_MODE_STA
#define SSID "YOUR SSID"
#define PASSWORD "YOUR PASSWORD"
```

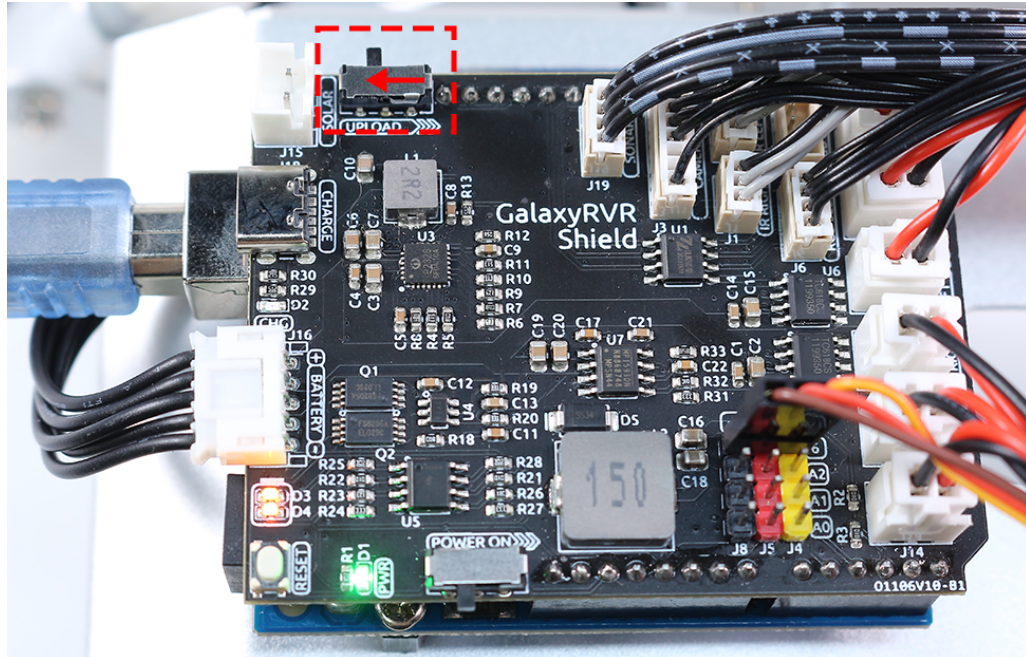
3. Upload the code to our Rover and bring our ESP32 CAM to life!

- The ESP32-CAM and the Arduino board share the same RX (receive) and TX (transmit) pins. So, before uploading the code, you'll need to first release the ESP32-CAM by slide this switch to right side to avoid any conflicts or potential issues.

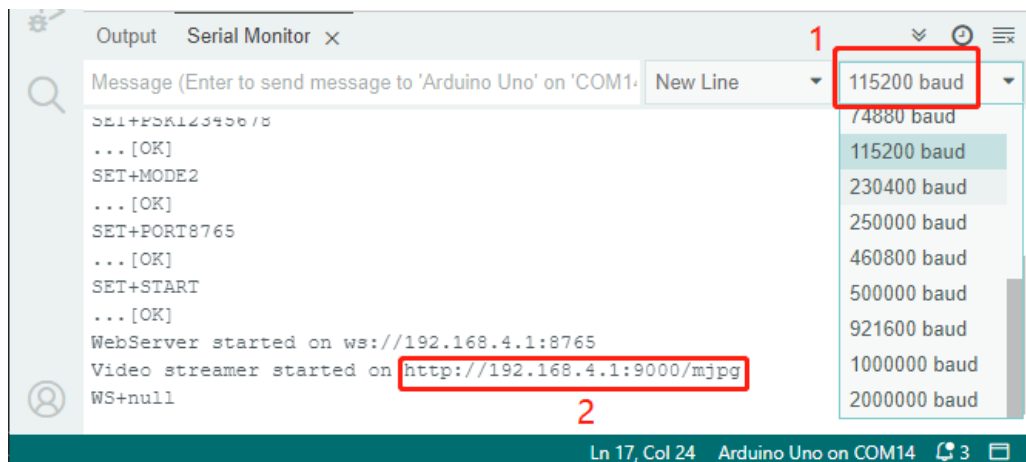


- Once the code has been uploaded successfully, switch it back to the left side to start the ESP32 CAM.

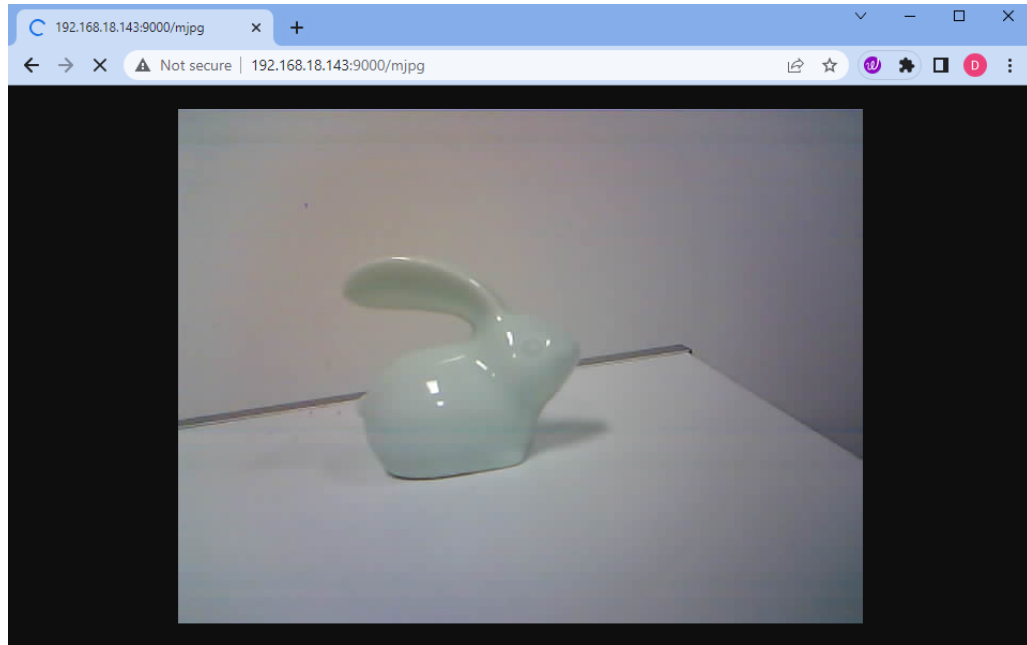
Note: This step and the previous one are required every time you re-upload the code.



- Open the **Serial Monitor** and set the baud rate to 115200. If no information appears, press the **Reset** button on the GalaxyRVR shield to run the code again. You should see an IP address in the serial monitor output. This is the address your Rover's camera is broadcasting to.



- Now, it's time to actually see what our Rover sees! Open up a web browser - we recommend Google Chrome - and enter the URL you see in the Serial Monitor, in the format `http://ip:9000/mjpg`.



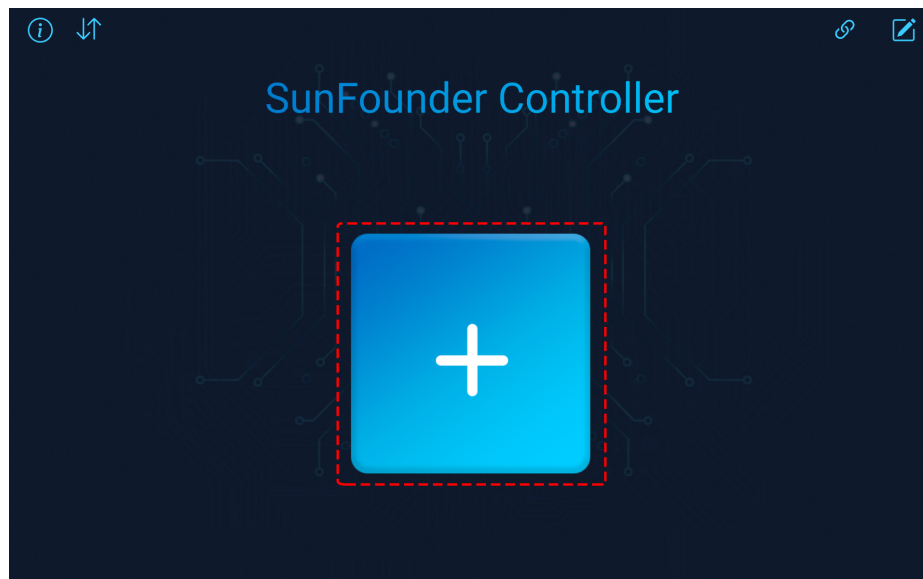
And voila! You should now be able to see the live feed from your Rover's camera. Isn't it amazing to think that you are viewing Mars (or maybe just your living room) from the Rover's perspective? Just like a real Mars Rover scientist!

Remember, this is just the beginning. There is so much more to explore and learn. In our next step, we will explore how to control our Rover while viewing the live camera feed. Exciting, isn't it? Onwards, explorers!

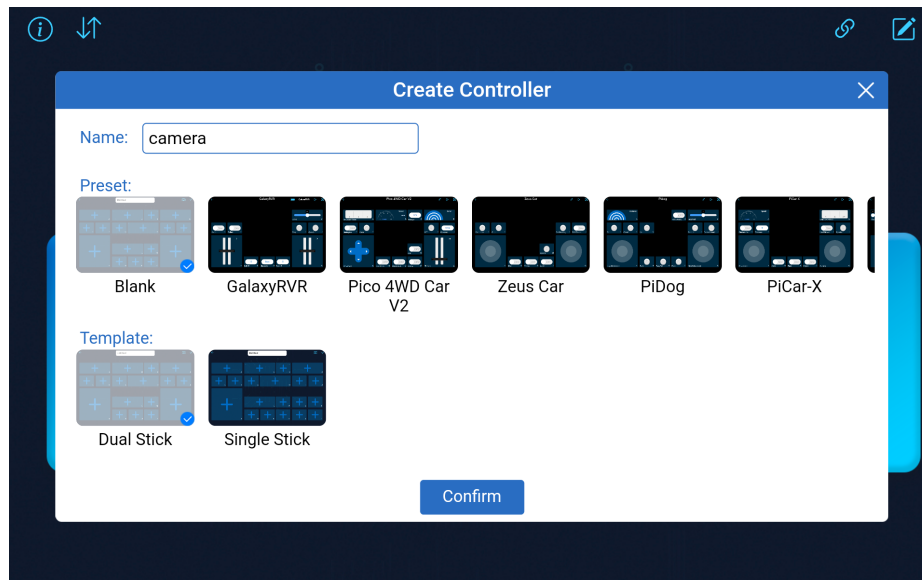
Step 3: Controlling and Viewing the Camera Feed Using the App

Ever wished you could view the Mars Rover's visual feed right on your smartphone while also being able to control its tilt mechanism? Now you can! With the help of the SunFounder Controller app, you'll be able to do exactly that. Follow the steps below:

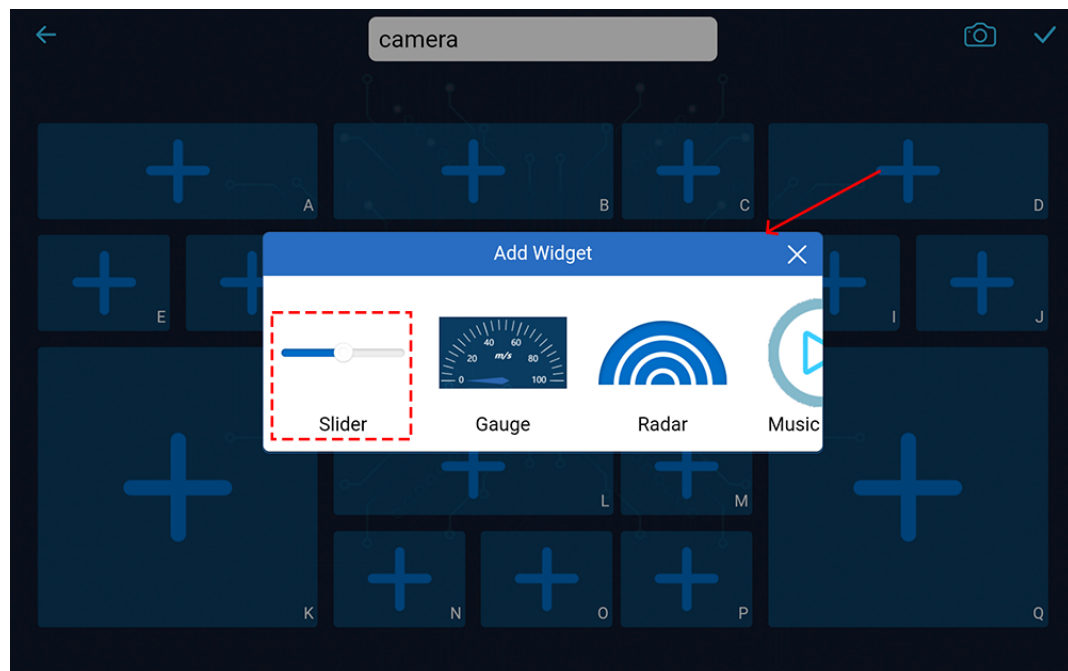
1. Install from **APP Store(iOS)** or **Google Play(Android)**.
2. Create a controller.
 - To add a controller on SunFounder Controller, click the + icon.



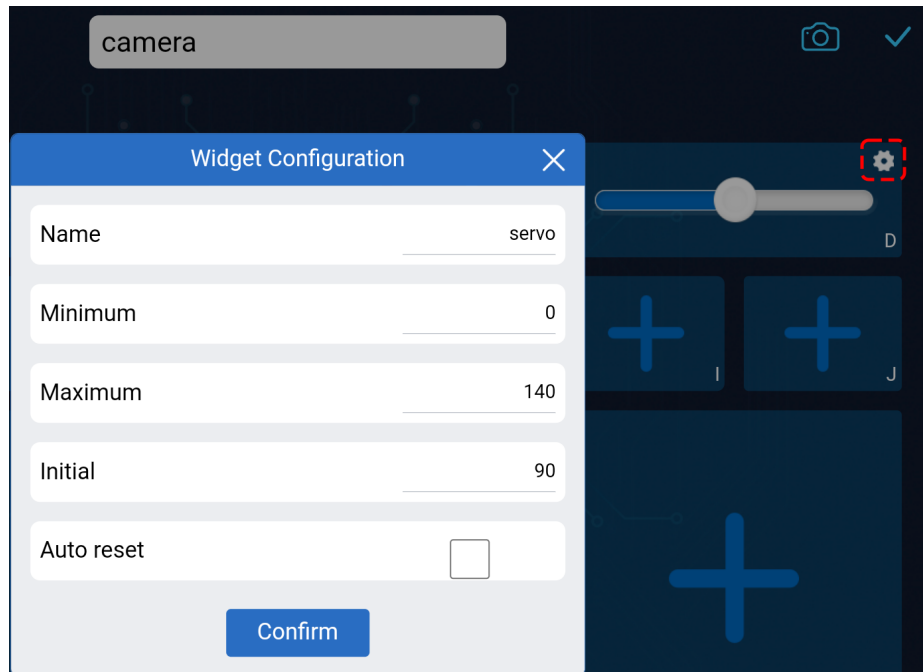
- Choose the **Blank** preset, select either **Dual** or **Single Stick** according to your preference. Give your new controller a name and click **Confirm**.




- You are now inside the controller. Click the + icon in the D section, then select Slider from the popup menu.



- This Slider widget is designed to control the tilt mechanism. As we learned in the previous lesson, its range is from 0 to 140. Therefore, we will set these as the minimum and maximum values for our Slider widget.



- Click the  button in the upper right corner to save this controller.

3. Let's write a code to capture the value of the slider:

- Based on the previous code, let's switch to AP mode, where you can set the SSID and PASSWORD to whatever you prefer.

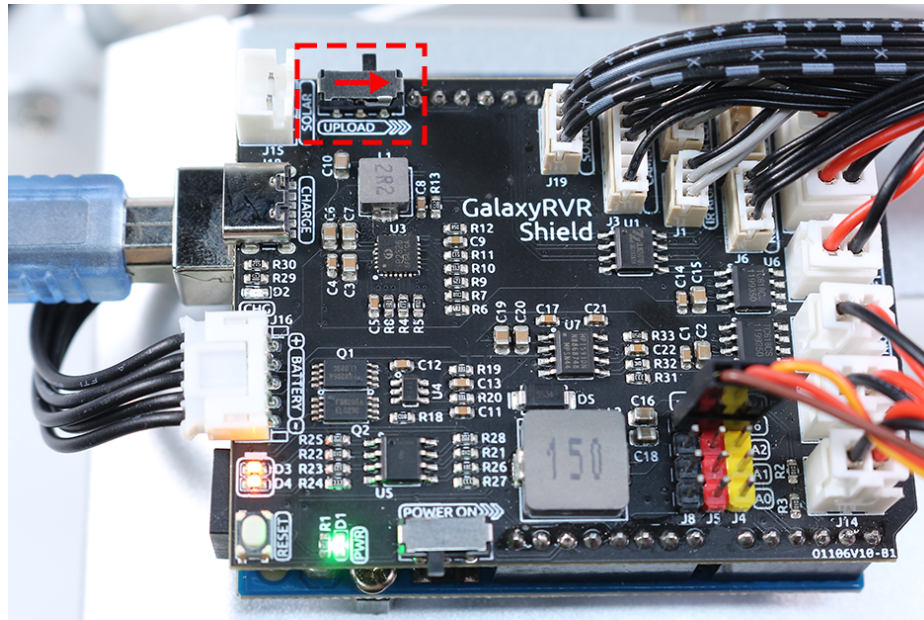
```
// AP Mode
#define WIFI_MODE WIFI_MODE_AP
#define SSID "GalaxyRVR"
#define PASSWORD "12345678"
```

- Next, we add an `onReceive()` function to receive values from the SunFounder Controller and print these values in the Serial Monitor. We use the `getSlider()` function to get the value of the **slider** widget. I added a **slider** widget in Region D, but if you added it in a different region, you need to change `REGION_D` to your region.

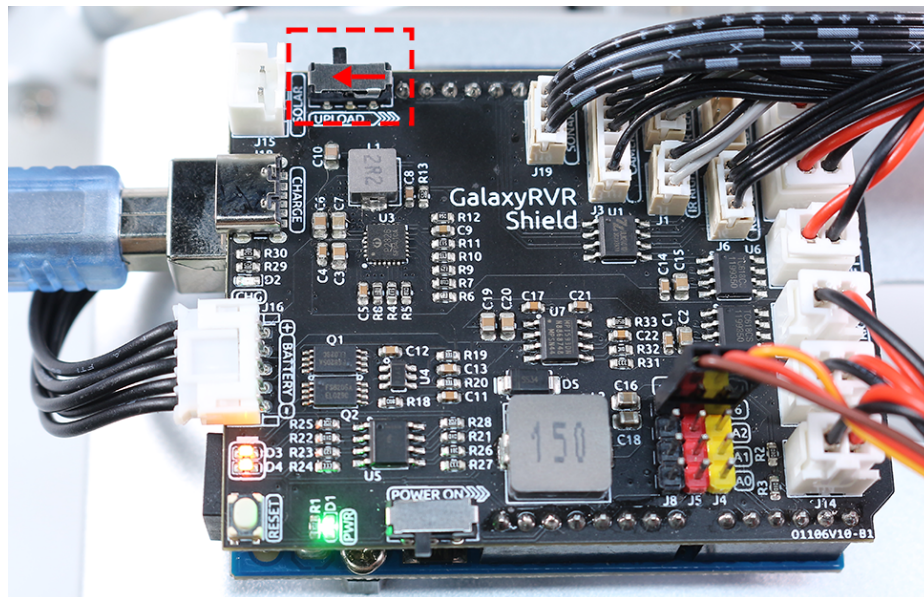
```
void onReceive() {
    int16_t sliderD = aiCam.getSlider(REGION_D);
    Serial.print("Slider D: ");
    Serial.println(sliderD);
}

void setup() {
    ...
    // Set the function to execute when data is received
    aiCam.setOnReceived(onReceive);
    ...
}
```

- Here is the complete code:
- Before uploading the code, make sure the switch is turned to the right.



- After the code is successfully uploaded, move the switch to the left to start the ESP32 CAM.



- When you see the following information in the Serial Monitor, you can move on to the next step.

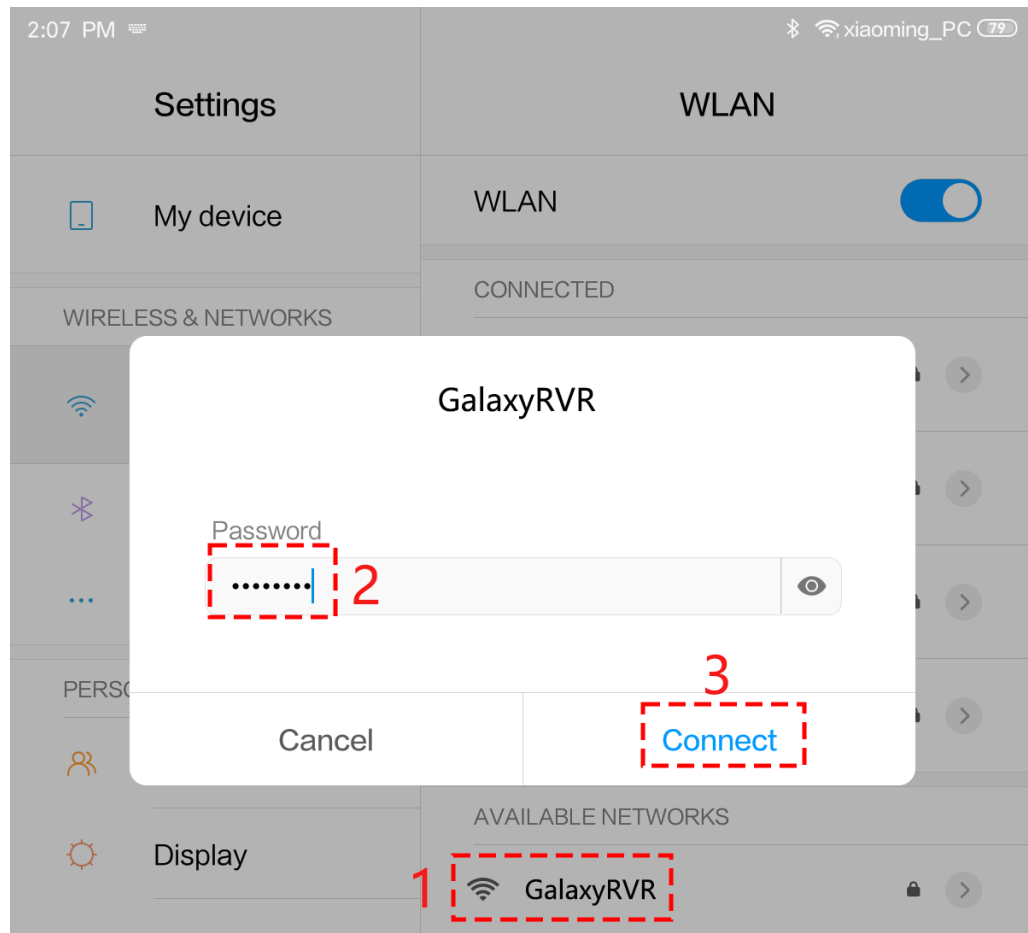
```
...[OK]  
SET+PORT8765  
...[OK]  
SET+START  
...[OK]  
WebServer started on ws://192.168.4.1:8765  
Video streamer started on http://192.168.4.1:9000/mjpg  
WS+null
```

4. Connect to the GalaxyRVR Network.

At this point, you should connect your mobile device to the local area network (LAN) provided by the

GalaxyRVR. By doing this, both your mobile device and the Rover will be on the same network, enabling smooth communication between the applications on your mobile device and the Rover.

- Find GalaxyRVR on the list of available networks on your mobile device (tablet or smartphone), enter the password 12345678, and connect to it.



- The default connection mode is **AP mode**. After you connect, there may be a prompt warning you that there is no Internet access on this WLAN network, please choose to continue the connection.


Confirm

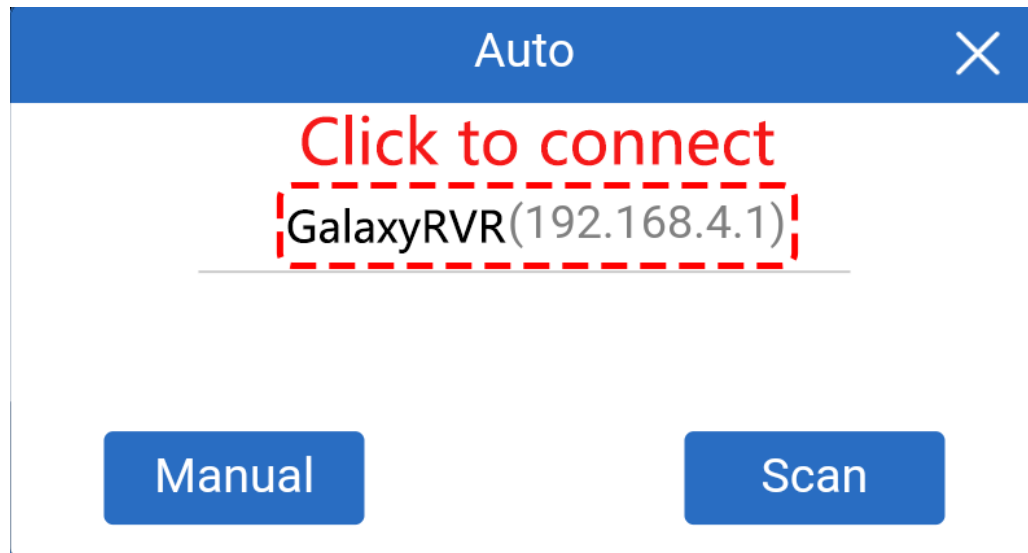
This WLAN network has no access to the internet. Other WLAN networks are available. Switch networks?

Stay connected


Switch networks

5. Connect and Activate the Controller.

- Now, return to the controller you created earlier (in my case, it's named "camera"). Use the  button to link the SunFounder Controller to the Rover and establish a line of communication. After a brief wait, GalaxyRVR(IP) (the name you assigned in the code with `#define NAME "GalaxyRVR"`) will appear. Click on it to establish a connection.



Note: Please verify that your Wi-Fi is connected to GalaxyRVR if you don't see the above message after some time.

- Once you see the "Connected Successfully" message, press the  button. This will bring up the camera's live footage on the app.



- Now, move the slider and open Arduino IDE's serial monitor simultaneously. You should see similar data like below.

```
Slider D: 105
WS+null
Slider D: 105
WS+null
Slider D: 105
WS+null
```

6. Let the Slider control the tilt mechanism.

Now that we know the values transmitted by the slider widget, we can directly use these values to rotate the servo.

Therefore, based on the previous code, add the following lines to initialize the servo and write the slider's value to the servo.

```
...
#include <Servo.h>

Servo myServo; // create a servo object
myServo.write(int(slidebar)); // control the servo to move to the current
↪angle

...

void onReceive() {
    ...
    myServo.write(int(slidebar)); // control the servo to move to the
↪current angle
}

void setup() {
    ...
    myServo.attach(6); // attaches the servo on pin 6
    ...
}
```

Here is the complete code:

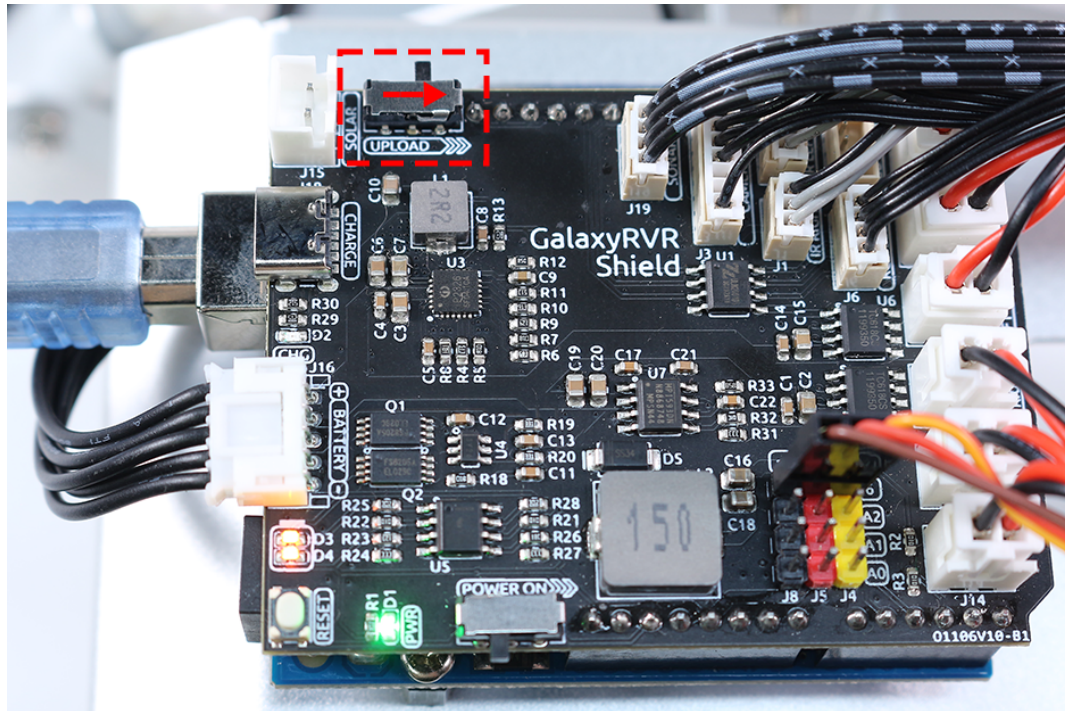
Upload the above code to the GalaxyRVR, repeat steps 4 and 5 above, reconnect to the GalaxyRVR LAN and re-run in the SunFounder Controller, then you can slide the slider to control the rover's tilt mechanism.

Now you've successfully learned to implement the SunFounder Controller and how to use the slider widget to control servo movements. This process will allow you to interact with your GalaxyRVR in a more intuitive and direct way.

Step 4: Reflection and Summary

Using the SunFounder Controller to operate your Mars Rover may seem a bit complicated at first. Every time you modify your code, you'll need to repeat the following steps:

- Prior to uploading the code, ensure the switch is turned to the right.



- Once the code has been successfully uploaded, switch to the left to initiate the ESP32 CAM.
- Connect to the GalaxyRVR Network.
- Connect and run the controller.

Though these steps might seem tedious, they are crucial for the process. After repeating them a few times, you'll become more familiar and comfortable with the procedure.

Now that we've finished this lesson, let's reflect on what we've learned through some questions:

- In the process of creating a new controller, you've encountered many different types of blocks. Have you considered what their individual functions might be?
- Is it possible to use other widgets to control the tilt mechanism?
- Or even directly control the Mars Rover's movements?

Let's anticipate our exploration of these questions in the next lesson!

3.12 Lesson 12: Driving the Rover with the App

In our last adventure, we mastered the art of using the SunFounder Controller to not only explore the world through the rover's eyes but also to adjust its gaze by controlling the tilt mechanism. It was like giving our Mars rover a sense of sight!

Now, imagine if you could steer this Martian explorer as well, directing its path at your whim. In this lesson, that's exactly what we're going to do! We're going to take our understanding of the SunFounder Controller to the next level and learn to maneuver our rover, giving us the thrilling experience of being a Mars rover driver!

With our vivid and lively STEAM course, children will not just learn; they'll experience the excitement and wonder of space exploration! Strap in and get ready for an exciting journey!

3.12.1 Learning Goals

- Gain a deeper understanding of the SunFounder Controller.
- Learn how to drive the Mars rover using the mobile app.

3.12.2 Materials needed

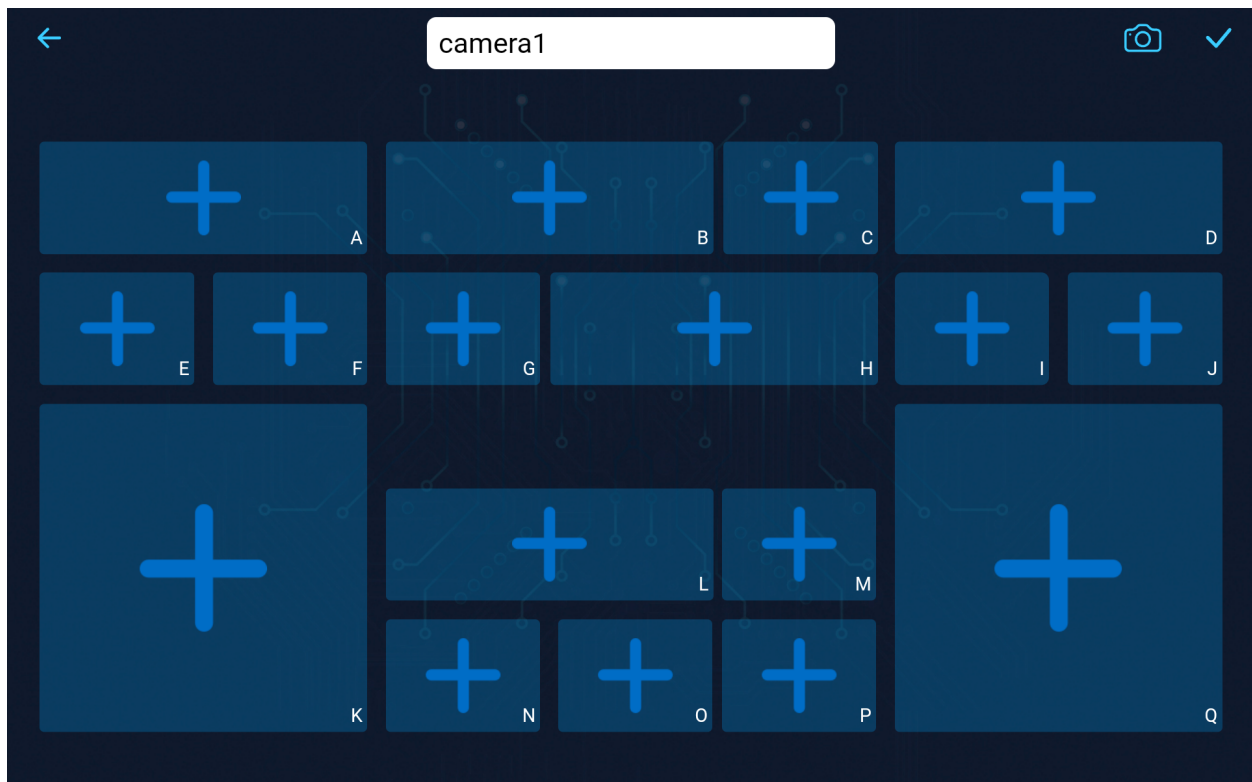
- Mars Rover model (equipped with all components, except for solar panel and bottom plate)
- Arduino IDE
- Computer
- Tablet or smartphone with SunFounder Controller app installed

3.12.3 Course Steps

Step 1: Dive Deeper into the SunFounder Controller

In our previous lesson, we got our first taste of the SunFounder Controller, its basic operations and uses. But surely, you're left with some burning questions, right? Time to quench that curiosity and delve deeper into this tech marvel.

On creating a new controller, you'll be met with a screen that might look like an enigma at first.



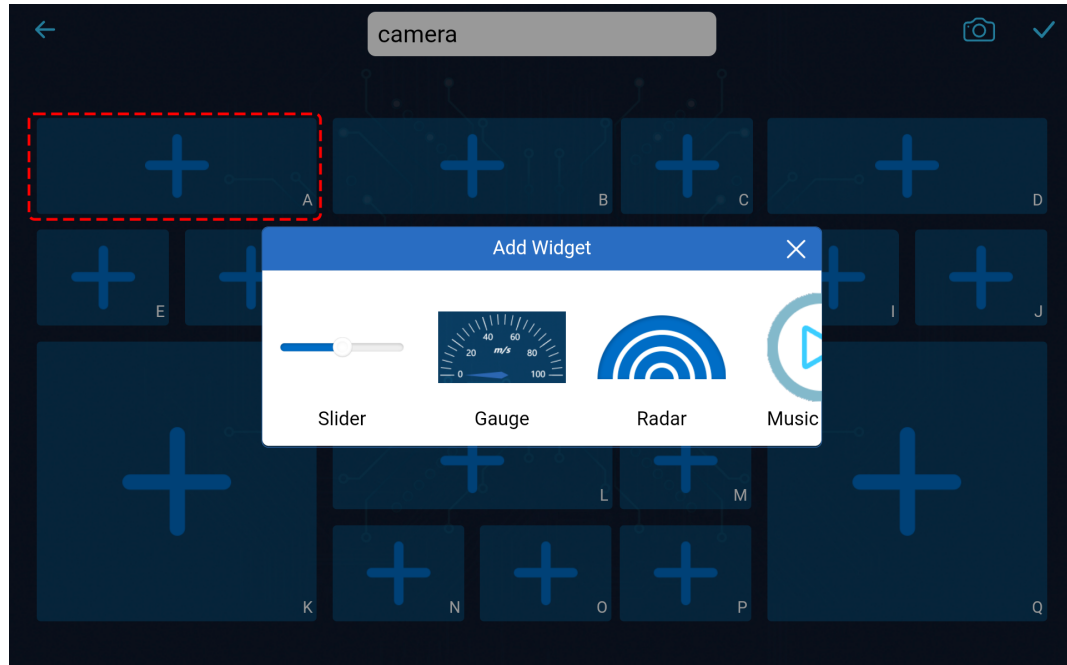
You'll see a kaleidoscope of shapes: long rectangles, short rectangles, and large squares, each uniquely tagged with identifiers from A to Q.

Ever wondered why such diversity? What do the identifiers A to Q imply?

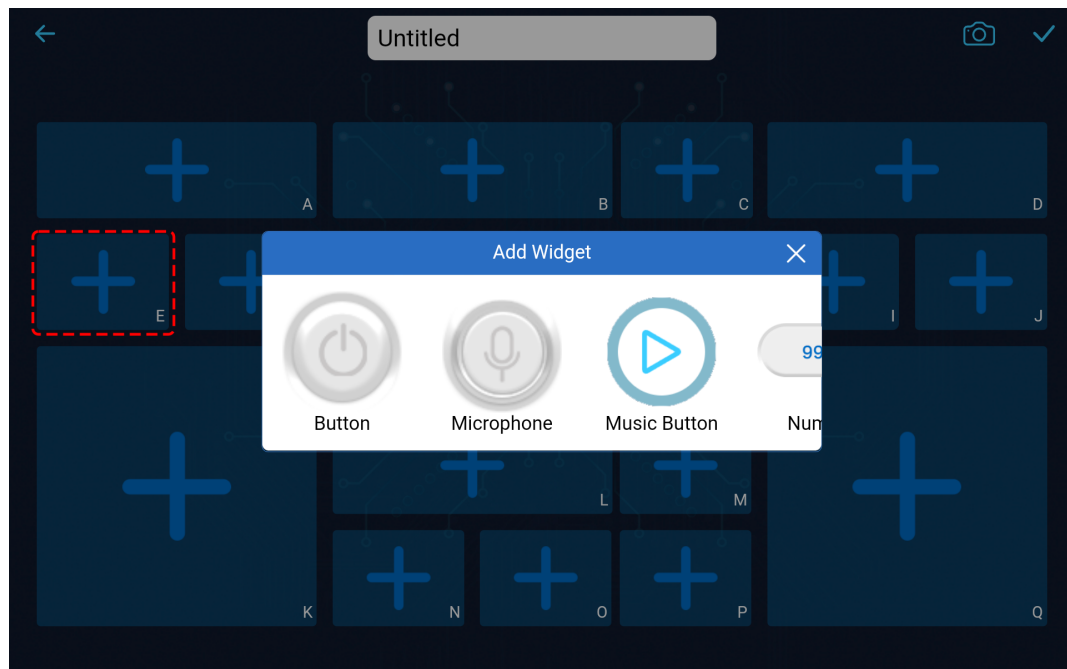
Let's embark on this exploratory journey!

- **Deciphering the Shapes**

For instance, tap on a long rectangular area. Like opening a treasure chest, you'll unveil several widgets. Remember the **Slider** from our previous class? Swiping it left or right controlled the Mars rover's camera tilt. Then there's the **Gauge**, can be used as your rover's personal speedometer. And a plethora of other widgets!

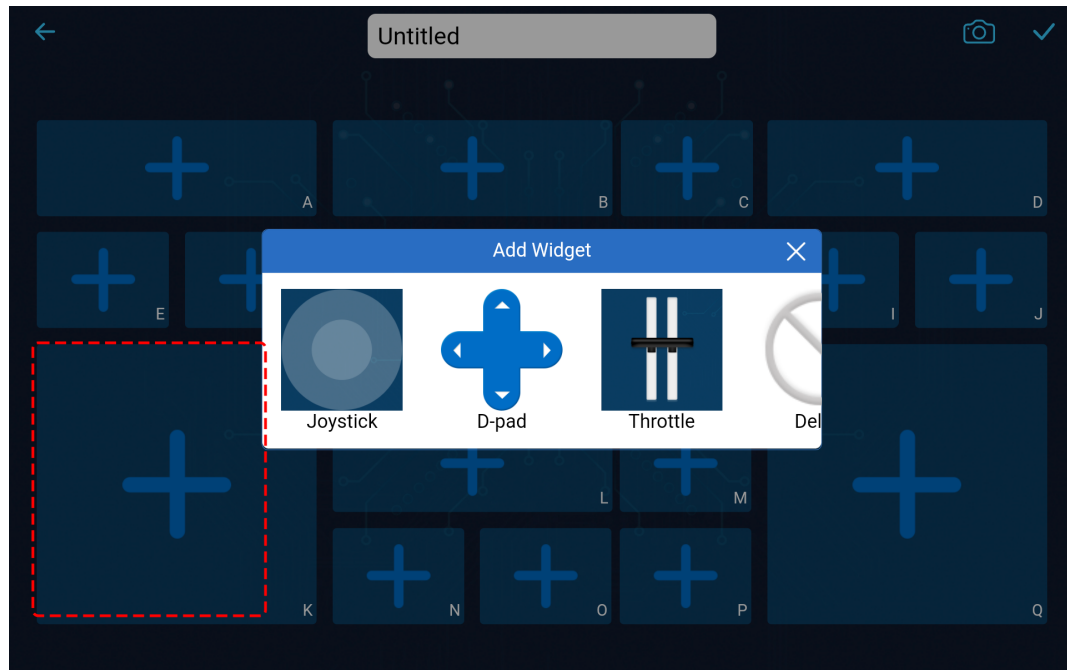


Tap on the shorter rectangle, and it reveals a different set of widgets. The **Button** widget, like a binary switch, issues singular commands, while the **Microphone** widget can listen to your voice commands to control the rover, among other widgets.



What about the square block, you ask? It houses the **Joystick** widget, the D-pad widget, and the

Throttle widget. And yes, there's more!



At this point, don't fret about each widget's function or usage. Familiarize yourself with what widgets each shape houses.

• Harnessing the Widgets

Through the content above, we know that there are many different widgets. So what can these widgets be used for?

We categorize these widgets into two types: control widgets and show widgets.

- **Control widgets** primarily let you manipulate the app, and then the Mars rover receives these control commands and performs corresponding actions.
- **Show widgets** allow you to use them to show some values in the app, such as sensor values, etc.

For a detailed description of these widgets, please refer to: .

Also, for the functions, parameters, etc. related to these widgets, please refer to: .

• The Role of Identifiers

Every shape carries an identifier on the bottom right corner. Why, you ask? Let's understand this with an example.

In our last class, we added a **Slider** widget in the **D** region. The code to get its value looked something like this:

```
int16_t sliderD = aiCam.getSlider(REGION_D);
```

What if we added a **Slider** widget in the **B** region? How would the code change? As simple as replacing REGION_D with REGION_B.

```
int16_t sliderD = aiCam.getSlider(REGION_B);
```



Easy, right? These identifiers help discern which widget you've added in which area.

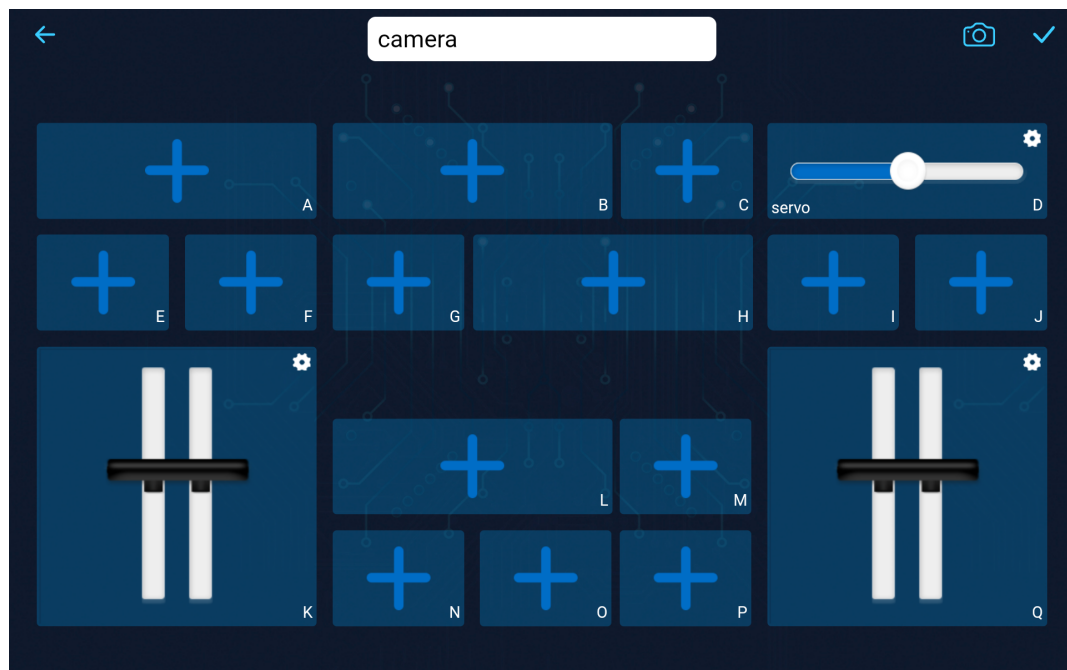
Note:

- Detailed instructions on creating and adding widgets, and connecting and operating the SunFounder Controller, are available in Step 3 of Lesson 11.
- For more in-depth operations, please visit .

Step 2: Control the Mars Rover with Dual Throttles

Now that we've delved into the ins and outs of the SunFounder Controller widgets, let's elevate our game by incorporating two Throttle widgets to commandeer the movement of the Mars Rover.

1. Place a **Throttle** widget each in the **K** and **Q** regions. You will need to hit the  button at the top right to switch to editing mode, and once you're done setting things up, click  to store your changes.



2. Since we plan to utilize two **Throttle** widgets to govern the Rover's mobility, let's tweak the function that dictates the rover's movements accordingly:

```
// Function to set the power of the motors
void carSetMotors(int8_t power_L, int8_t power_R) {
    // Set power for the left motor
    if (power_L >= 0) {
        SoftPWMSet(in1, map(power_L, 0, 100, 0, 255));
        SoftPWMSet(in2, 0);
    } else {
        SoftPWMSet(in1, 0);
        SoftPWMSet(in2, map(power_L, 0, -100, 0, 255));
    }

    // Set power for the right motor
    if (power_R >= 0) {
        SoftPWMSet(in3, 0);
    }
}
```

(continues on next page)

(continued from previous page)

```

        SoftPWMSet(in4, map(power_R, 0, 100, 0, 255));
    } else {
        SoftPWMSet(in3, map(power_R, 0, -100, 0, 255));
        SoftPWMSet(in4, 0);
    }
}

```

Sure, let's break down the `carSetMotors()` function. This function accepts two arguments, `power_L` and `power_R`, which are likely the power settings for the left and right motors respectively. The values of these arguments are presumably from -100 to 100, where negative values indicate reverse motion, 0 indicates stop, and positive values indicate forward motion.

- Set power for the left motor:
 - If `power_L` is greater than or equal to 0, the left motor is set to move forward.
 - `SoftPWMSet(in1, map(power_L, 0, 100, 0, 255))` uses the Arduino `map` function to map the input range (0 to 100) to the output range (0 to 255) - this is because PWM values in Arduino are between 0 (0% duty cycle) and 255 (100% duty cycle). This mapped value is then passed to the `SoftPWMSet` function along with `in1`.
 - If `power_L` is less than 0, the left motor is set to move in reverse and the input range for the `map` function is now 0 to -100.
- Set power for the right motor:
 - This follows the same logic as setting the power for the left motor, but uses `in3` and `in4` instead, and the input power values are `power_R` instead of `power_L`.

Overall, this function takes two motor power values, converts them into the appropriate PWM values, and sets the PWM values on the correct motor control pins to achieve the desired motion.

3. Within the `onReceive()` function, retrieve the values from the two **Throttle** widgets and employ them as the power for the left and right motors of the Mars Rover.

```

void onReceive() {
    // Get the value of the slider in region D
    int16_t sliderD = aiCam.getSlider(REGION_D);

    // Move the servo to the angle indicated by the slider
    myServo.write(int(sliderD));

    // Get the throttle values for the left and right
    int throttle_L = aiCam.getThrottle(REGION_K);
    int throttle_R = aiCam.getThrottle(REGION_Q);

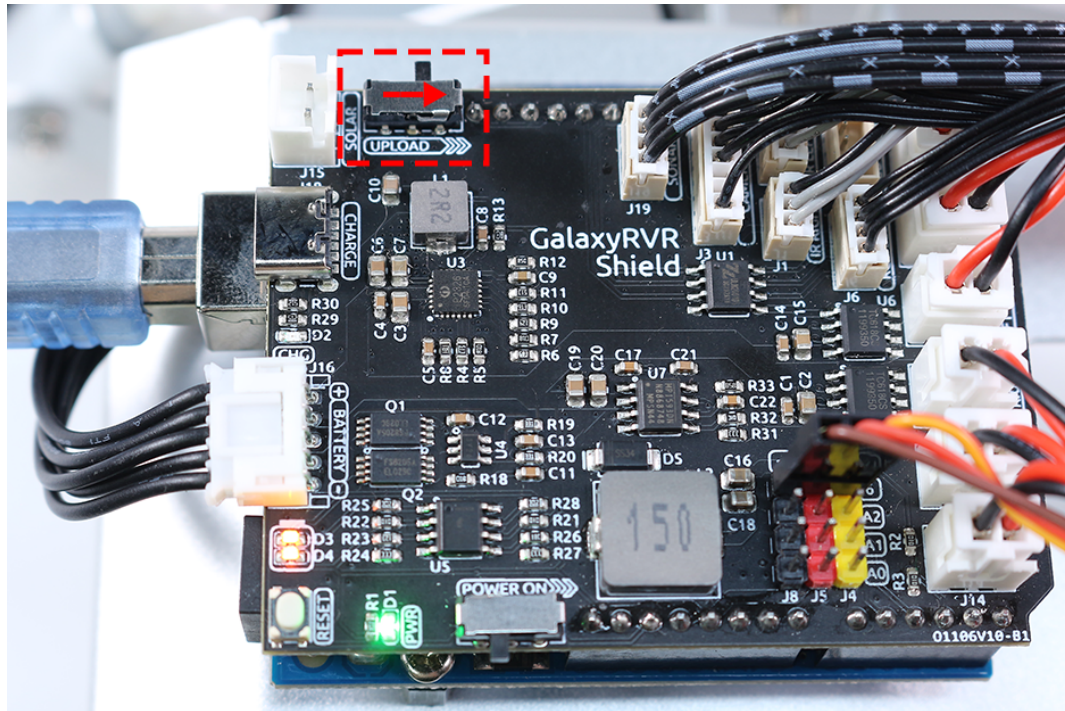
    // Set the power for the motors
    carSetMotors(throttle_L, throttle_R);
}

```

Here is the complete code:

Each time you rerun the code, you need to repeat the following four steps:

- Prior to uploading the code, ensure the switch is turned to the right.



- Once the code has been successfully uploaded, switch to the left to initiate the ESP32 CAM.
- Connect to the GalaxyRVR Network.
- Connect and run the controller.

Now, with a simple glide of the thumb on your Throttle widgets, you'll witness the Mars Rover in action, powering and pivoting with a renewed spirit. Let the exploration commence!

Step 3: Visualizing Sensor Readings

In our journey with the SunFounder Controller, we have been actively interacting with our Mars rover through control widgets, including sliders for adjusting the tilt and throttles for dictating the rover's movement. But what about harnessing the power of display widgets to paint a vivid picture of our rover's surroundings?

Let's see how we can bring this to life by visualizing the values from the left and right infrared (IR) avoidance modules and the distance captured by the ultrasonic module. This real-time data will provide us with a clear snapshot of the rover's operating environment.

Here's how we can achieve this:

1. Get started by adding three Number widgets to your SunFounder Controller. Don't forget, you can personalize their names and units using the settings button.



- Next, let's delve into the code. Start by bringing over the IR avoidance modules and ultrasonic module code snippets from our previous lessons.

```
...
// Define the pin for the ultrasonic module
#define ULTRASONIC_PIN 10

// Define the pins for the IR modules
#define IR_RIGHT 7
#define IR_LEFT 8

void setup() {
    ...

    // Set the IR module pins as inputs
    pinMode(IR_RIGHT, INPUT);
    pinMode(IR_LEFT, INPUT);
}

float readSensorData() {
    // A 4ms delay is required, otherwise the reading may be 0
    delay(4);

    //Set to OUTPUT to send signal
    pinMode(ULTRASONIC_PIN, OUTPUT);

    ...
}
```

- In the `onReceive()` function, extract the values from the avoidance modules and the ultrasonic sensor. Subsequently, update these values in the `sendDoc[]` dictionary. The N, P, and O keys correspond to the region codes of the three Number widgets you've added.

```
// Function to execute when data is received from the Controller
```

(continues on next page)

(continued from previous page)

```

void onReceive() {

    ...

    // Read values from IR sensors
    int leftValue = digitalRead(IR_LEFT);
    int rightValue = digitalRead(IR_RIGHT);
    aiCam.sendDoc["N"] = leftValue;
    aiCam.sendDoc["P"] = rightValue;

    // ultrasonic
    float distance = readSensorData();
    aiCam.sendDoc["O"] = distance;
}

```

Here is the complete code:

Once the code has been successfully uploaded, get your SunFounder Controller up and running. You'll be greeted with the real-time values of the avoidance modules and the distance detected by the ultrasonic sensor, painting a clear picture of the rover's immediate environment.



With this step behind you, you've successfully navigated the world of show widgets. Feel free to experiment with

different widgets to display the information that you find interesting. Happy exploring!

Step 4: Reflection and Conclusion

In this lesson, we've forged a deeper understanding of the SunFounder Controller, grasping how we can utilize its widgets to not only steer our Mars rover but also monitor its environmental data in real time.

Now, here's a challenge for you:

How about adding some Switch widgets to your SunFounder Controller? With these switches activated, the Mars rover could switch between avoidance and follow modes. Or, why not use the switches to control the light strip – switching it on or off, or even changing its color?

Do you have the confidence to take this on?

We're looking forward to seeing you conquer this challenge!

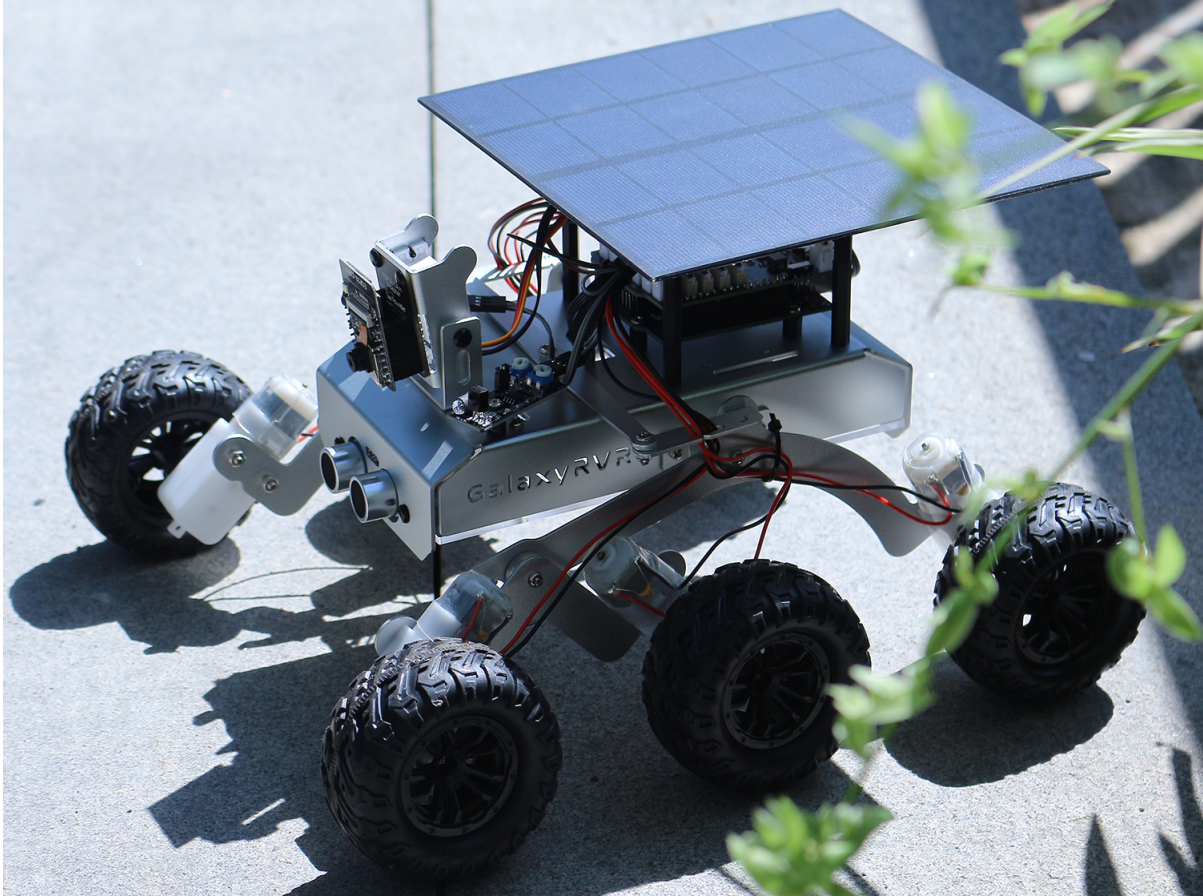
3.13 Lesson 13: Investigating the Mars Rover Energy System

Welcome to the final lesson of our Mars rover exploration journey. This time, we are going to delve into the heart of the rover - its energy system.

When we think about exploring distant planets like Mars, one of the most crucial aspects to consider is energy. How do these rovers power themselves in such harsh and remote environments? In this lesson, we'll explore this fascinating topic and learn how rovers, like our Mars rover model, harness and manage energy.

We'll investigate the working principles of battery and solar panel and even get our hands-on practice in installing and using these power sources on our rover model. Furthermore, we'll take our skills a notch higher by using Arduino to monitor the battery level.

By the end of the lesson, we'll be able to display these vital energy statistics right on our APP, allowing us to have a real-time understanding of our rover's energy status. Ready to power up our exploration? Let's get started!



3.13.1 Learning Goals

- Understand the working principles of battery and solar panel.
- Practice installing the solar panel on the Mars rover model.
- Learn how to use Arduino to monitor battery level and the charging status of solar panel.
- Display battery level on the APP.

3.13.2 Materials needed

- Mars Rover model (equipped with all components, except for solar panel and bottom plate)
- Solar panel and bottom plate
- Arduino IDE
- Computer
- Tablet or smartphone with SunFounder Controller app installed

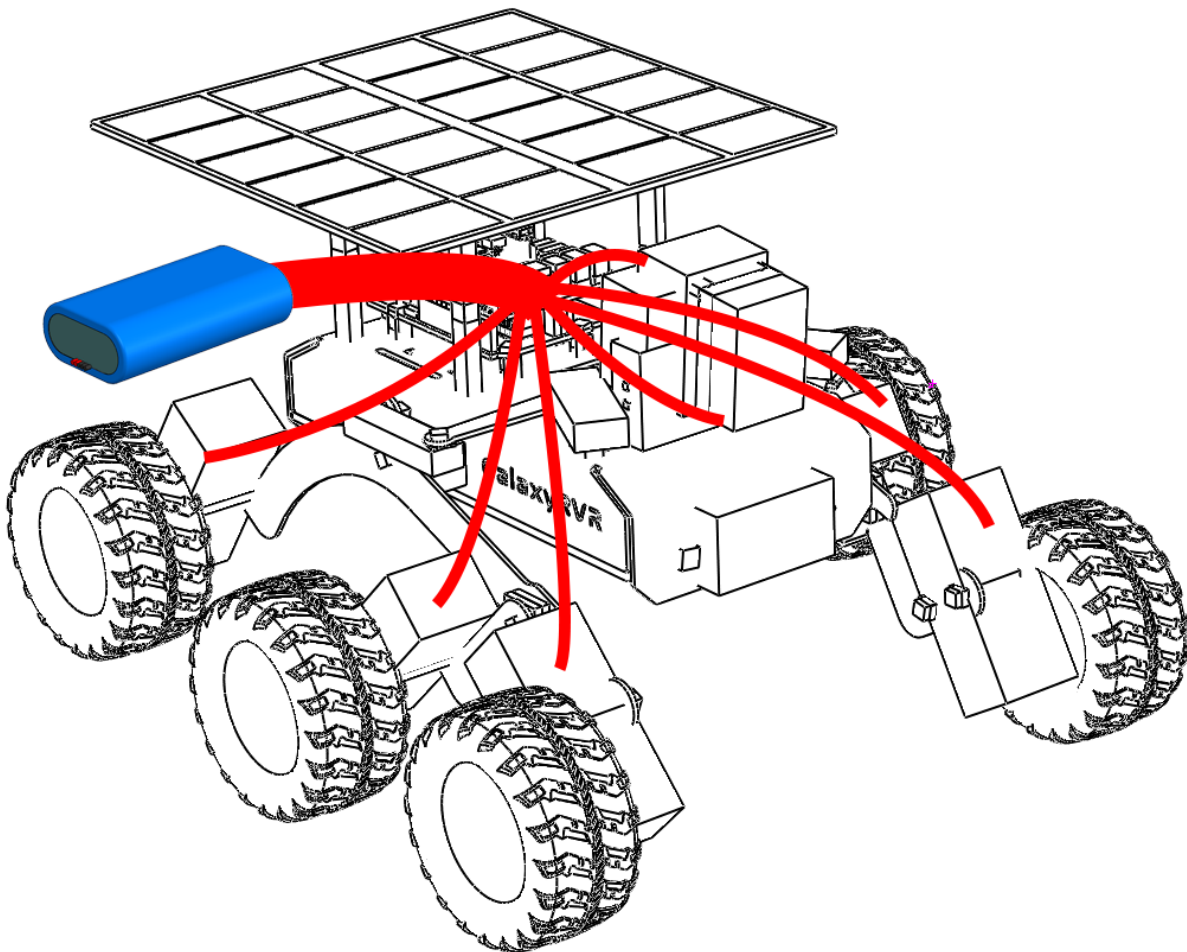
3.13.3 Course Steps

Step1: Introduction to the Mars Rover's Energy System

Just as our bodies need a constant supply of energy to function, our Mars Rover needs a way to store and generate power for its exploration missions. Imagine the Rover's energy system like the heart in our bodies. Just as our hearts pump blood to all parts of our body, supplying necessary oxygen and nutrients, the Rover's energy system keeps energy flowing to every part of the Rover, ensuring it can perform its tasks smoothly.

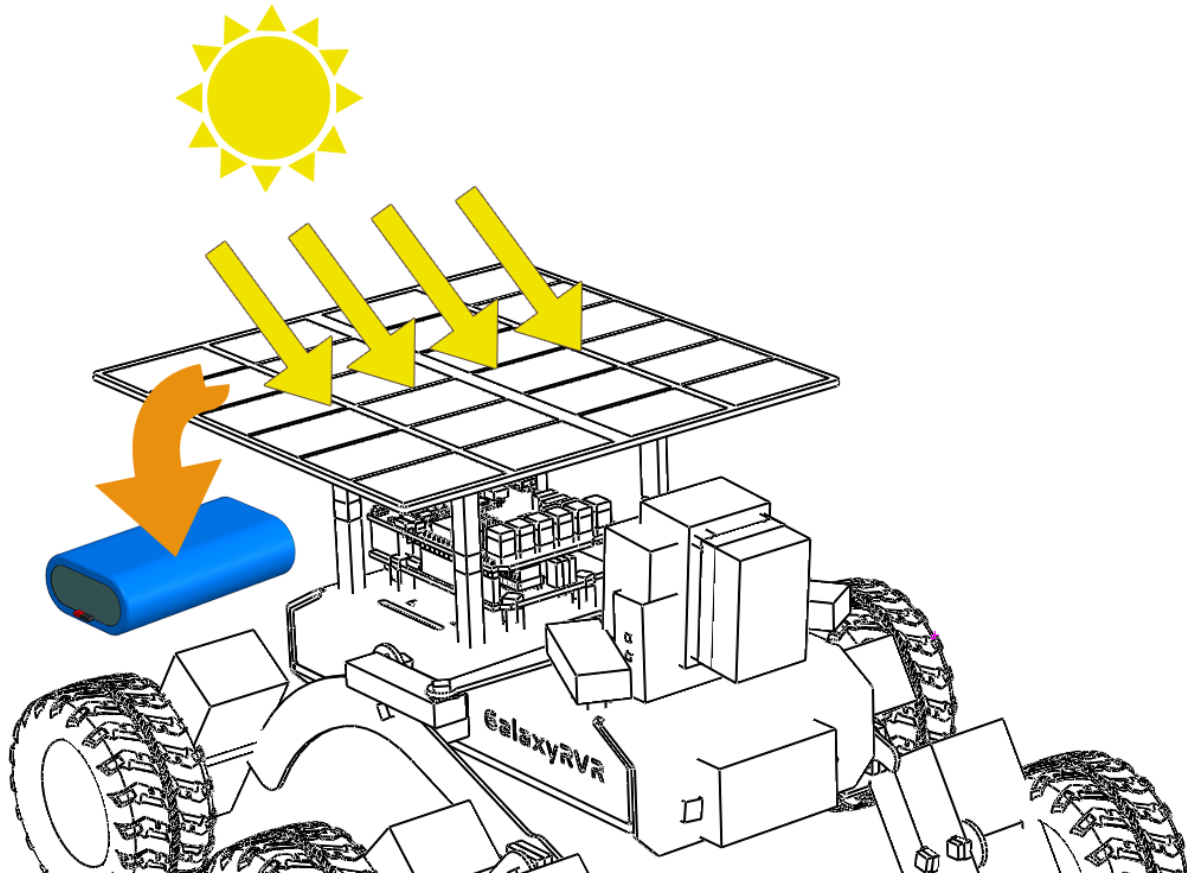
The main components of this energy system are the batteries and the solar panels, working in tandem to ensure the Rover can operate at all times, day or night.

The role of the batteries in the Rover's energy system is similar to the role of energy storage in our bodies. Just as we need to store energy for use when active, the Rover needs a way to store energy for its exploration missions. The energy stored in the batteries is continuously dispatched to various parts of the Rover, allowing it to carry out its tasks systematically.



But what happens when the energy in the batteries runs out? How does it replenish its energy stores? This is where the solar panels come into play.

Much like trees absorb sunlight for photosynthesis to create food, our Mars Rover uses solar panels to harness energy from the Sun, converting it into electricity that is stored in the batteries for use. Each solar panel is made up of many smaller solar cells. These cells are composed of a material that can convert light into electricity – a process called the photovoltaic effect. When sunlight hits the cells, they generate an electric current that can be used immediately or stored in the Rover's batteries for later use.



However, harnessing solar energy on Mars is not as easy as it sounds. Dust storms can reduce the amount of sunlight reaching the panels, and the weaker Martian sunlight (compared to Earth's) means that the panels generate less power than they would here at home. Despite these challenges, solar power is still the most practical and efficient way of powering our Mars Rover.

But how do we know when the solar panels are doing their job and when the batteries are getting low on power? This is where our Arduino comes in. In the next section, we will learn how to use Arduino to monitor the charging and discharging of the Rover's batteries.

Step 2: Mounting the Solar Panel on the Mars Rover

Before we begin this step, we need to have our Mars Rover model, a solar panel, and the cables necessary to connect the solar panel to the Rover's power system.

This is a process that allows us to put theory into practice and truly appreciate the charm of Science, Technology, Engineering, and Mathematics (STEM) education. Let's get started!

Step 3: Programming to Monitor Battery Voltage and Charge

Now that we have installed the solar panels on our Mars Rover model, the next step is to monitor the voltage and charge of the battery through programming.

This code effectively creates a simple battery monitor, which is particularly useful in applications like the Mars Rover where power management is crucial. It will allow you to monitor the state of the battery, helping you understand when the Rover needs to be recharged or when power-consuming tasks should be scheduled.

Sure, let's break down the different parts of this code:

- This line is defining `BATTERY_PIN` as the analog pin A3, which is where the battery voltage will be read from.

```
#define BATTERY_PIN A3
```

- This function calculates the battery's voltage. It first reads the analog value from BATTERY_PIN, then converts it into voltage. Because the Arduino's analog-to-digital converter (ADC) operates on a scale of 0-1023, we divide the raw reading by 1023. We then multiply by 5 (the reference voltage of the Arduino) and by 2 (assuming a voltage divider of 2), to convert this to a voltage reading.

```
float batteryGetVoltage() {
    // Reads the analog value from the battery pin
    int adcValue = analogRead(BATTERY_PIN);
    // Converts the analog value to voltage
    float adcVoltage = adcValue / 1023.0 * 5 * 2;
    // Rounds the voltage to two decimal places
    float batteryVoltage = int(adcVoltage * 100) / 100.0;
    return batteryVoltage;
}
```

The raw ADC reading from the Arduino's analog-to-digital converter is divided by 1023 to convert it into a fraction, then multiplied by 5 to translate it into voltage, as Arduino uses a reference voltage of 5 volts.

However, because the battery voltage higher than Arduino's maximum input voltage, a resistor is used to protect the Arduino. Therefore, we multiply the ADC voltage by 2 to counteract the effect of the resistor and obtain the correct battery voltage.

- This function calculates the battery's percentage of charge based on its voltage. It uses the map function to map the voltage value (ranging from 6.6 to 8.4 volts) to a percentage (ranging from 0 to 100).

```
uint8_t batteryGetPercentage() {
    float voltage = batteryGetVoltage(); // Gets the battery voltage
    // Maps the voltage to a percentage.
    int16_t temp = map(voltage, 6.6, 8.4, 0, 100);
    // Ensures the percentage is between 0 and 100
    uint8_t percentage = max(min(temp, 100), 0);
    return percentage;
}
```

Step 4: Putting the Mars Rover's Energy System to the Test: Indoor and Outdoor Runs

Having coded our battery monitoring system, it's now time to set the Mars Rover into action. Begin by charging the Rover to full capacity, and plan for two 30-minute exploratory missions - one indoors, and another outdoors in the sunlight. Record the initial battery level before each mission, and compare it with the battery percentage at the end of each test. The following table serves as a useful template to keep track of your findings:

Table 1: Power Test

	Sun Shine	In Room
Start Battery Percentage		
End Battery Percentage		

Observe the difference in the battery levels following each test. Did the Rover's battery last longer when it was basking in outdoor sunlight? What conclusions can we draw about the efficacy of the solar panel from this observation?

Understanding these variances will help us better comprehend how solar energy can effectively power a Mars Rover, even in remote, harsh environments such as those found on the Martian surface.

Step 5: Display the Battery Level on the App

Naturally, it's impractical to repeatedly upload new code to the Mars Rover just to check the remaining battery level. That could be quite inconvenient.

Instead, we can send the battery level to an app, allowing us to easily monitor how much playtime is left while we're having fun!

From previous lessons, we've learned that when we want to shown data on the SunFounder Controller or control the Mars Rover using widgets, we need to add these widgets first.

However, the battery level display is a special widget. It has a dedicated key (BV), and its display isn't located in the areas labeled from A to Q. Instead, it's represented by a battery icon in the top-right corner.

Here's how we include it in our code:

```
...
// This pin reads the voltage of the battery
#define BATTERY_PIN A3
...

void setup() {
    ...
    // Sets the battery pin as an input
    pinMode(BATTERY_PIN, INPUT);
}

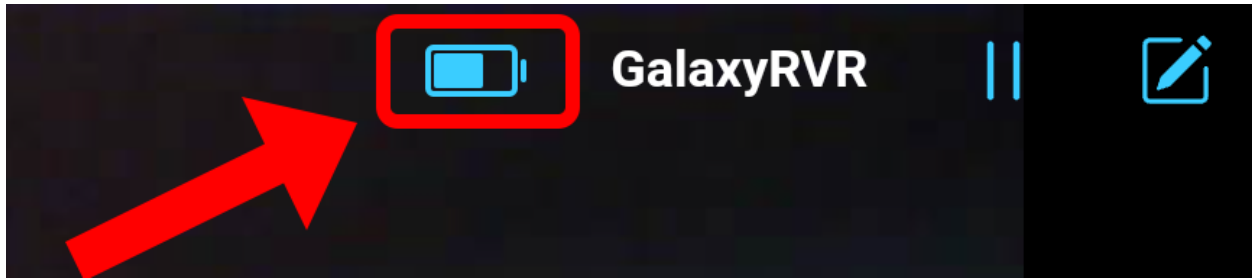
// Function to execute when data is received from the Controller
void onReceive() {
    ...
    //show battery voltage
    aiCam.sendDoc["BV"] = batteryGetVoltage();
}

...

// This function reads the battery voltage
float batteryGetVoltage() {
    // Reads the analog value from the battery pin
    int adcValue = analogRead(BATTERY_PIN);
    // Converts the analog value to voltage
    float adcVoltage = adcValue / 1023.0 * 5 * 2;
    // Rounds the voltage to two decimal places
    float batteryVoltage = int(adcVoltage * 100) / 100.0;
    return batteryVoltage;
}
```

Please find the full code attached:

After successfully uploading the code, get your SunFounder Controller up and running. You'll see the battery level shown in the top-right corner.



By completing this step, we've mastered the Mars Rover's energy system and gained the ability to monitor its power levels in real-time.

Now that we have learned how to harness the sun's power to operate the Mars Rover effectively, we can start planning more extensive explorations of our backyard or even venture into more challenging terrains!

Step 6: Reflection

Throughout this lesson, we've focused on understanding the crucial role of the energy system in the Mars Rover, and the mechanisms to monitor the Rover's remaining energy. The solar panel-based energy system not only powers the Rover but also underlines the importance of renewable energy sources in space exploration.

With the knowledge you have now, think about the real-life implications of this system. Consider the challenges that a solar energy system might encounter on Mars. How might extreme temperatures, dust storms, or long periods of darkness affect the energy supply? What solutions could you propose to tackle these obstacles?

Step 7: Looking Forward

Now that we've given our Mars Rover the ability to move, it's time to let it start its exploration journey! You can let it wander in various terrains mimicking the Mars environment.

For instance, you can let it climb over a heap of stones.

Or let it navigate through a thick grassy patch.

Or set it on a course on a gravel terrain full of stones.

However, please note that if the obstacle is too high, the rover might not be able to climb over it.

These varied terrains present unique challenges for the rover, just as they would for a real Mars Rover. As you watch your rover try to overcome these obstacles, you're experiencing a small part of what scientists and engineers at NASA do when they send rovers to Mars!

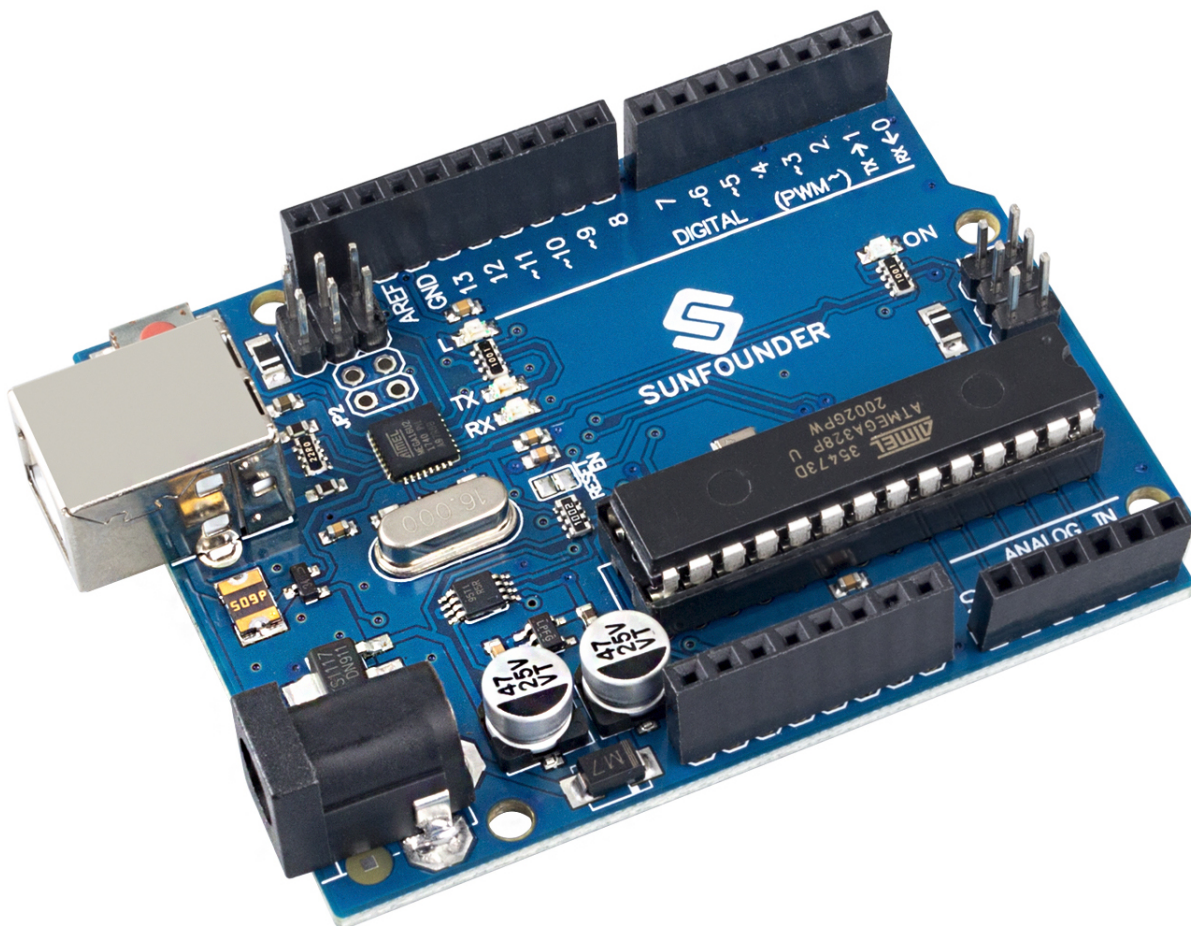
As we conclude our Mars Rover lessons, it's important to reflect on what we've learned. We hope this journey has not only expanded your knowledge and skills but also sparked curiosity and a desire to explore. Whether your Rover roams in your backyard or across the vast expanse of your imagination, the discoveries you make along the way are sure to be extraordinary.

HARDWARE

When you are writing code, you may need to know how each module works or the role of each pin, then please see this chapter.

In this chapter you will find a description of each module's function, technical parameters and working principle.

4.1 SunFounder R3 Board

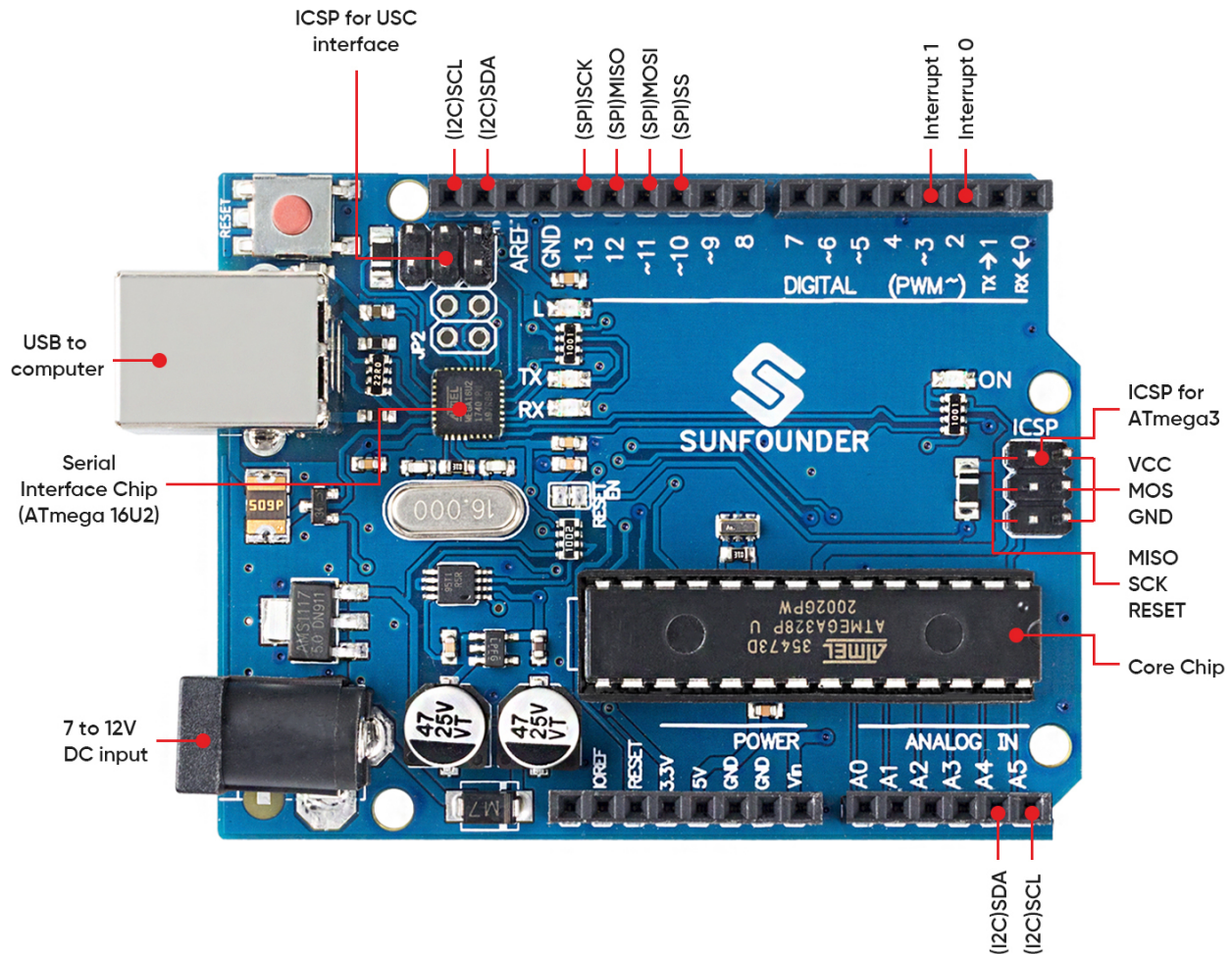


Note: The SunFounder R3 board is a mainboard with almost the same functions as the [Arduino Uno](#), and the two

boards can be used interchangeably.

SunFounder R3 board is a microcontroller board based on the ATmega328P ([datasheet](#)). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

Technical Parameters



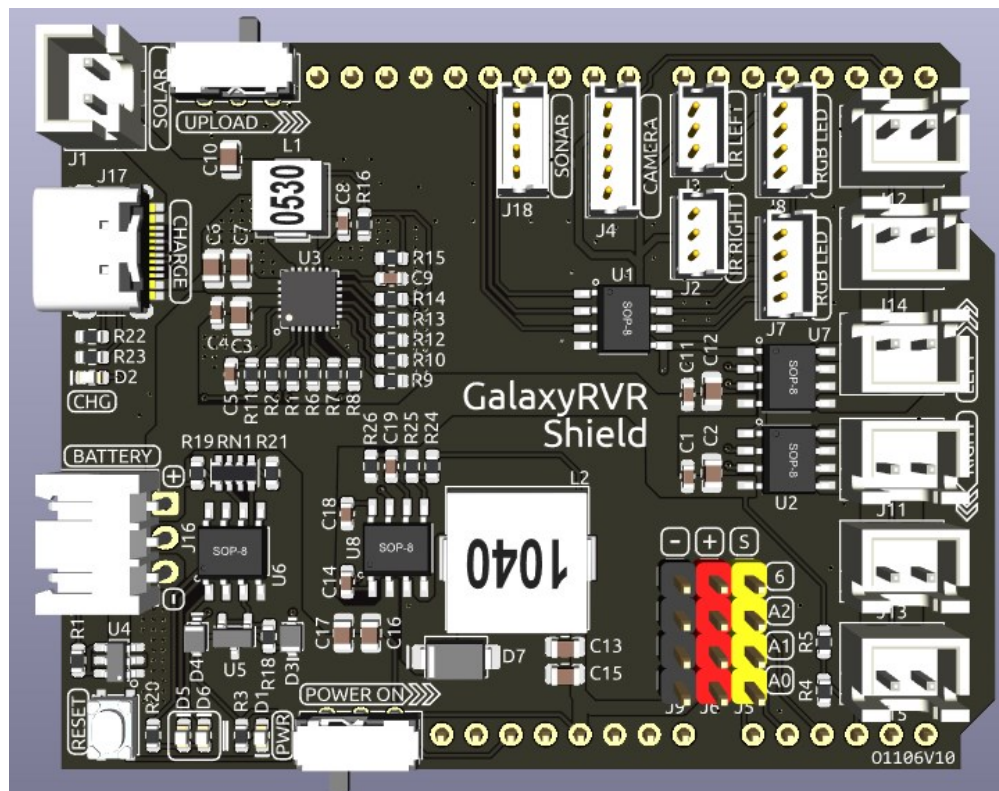
- MICROCONTROLLER: ATmega328P
- OPERATING VOLTAGE: 5V
- INPUT VOLTAGE (RECOMMENDED): 7-12V
- INPUT VOLTAGE (LIMIT): 6-20V
- DIGITAL I/O PINS: 14 (0-13, of which 6 provide PWM output(3, 5, 6, 9-11))
- PWM DIGITAL I/O PINS: 6 (3, 5, 6, 9-11)
- ANALOG INPUT PINS: 6 (A0-A5)
- DC CURRENT PER I/O PIN: 20 mA
- DC CURRENT FOR 3.3V PIN: 50 mA

- FLASH MEMORY: 32 KB (ATmega328P) of which 0.5 KB used by bootloader
- SRAM: 2 KB (ATmega328P)
- EEPROM: 1 KB (ATmega328P)
- CLOCK SPEED: 16 MHz
- LED_BUILTIN: 13
- LENGTH: 68.6 mm
- WIDTH: 53.4 mm
- WEIGHT: 25 g
- I2C Port: A4(SDA), A5(SCL)

What's More

- [Arduino IDE](#)
- [Arduino Programming Language Reference](#)
- [ATmega328P Datasheet](#)

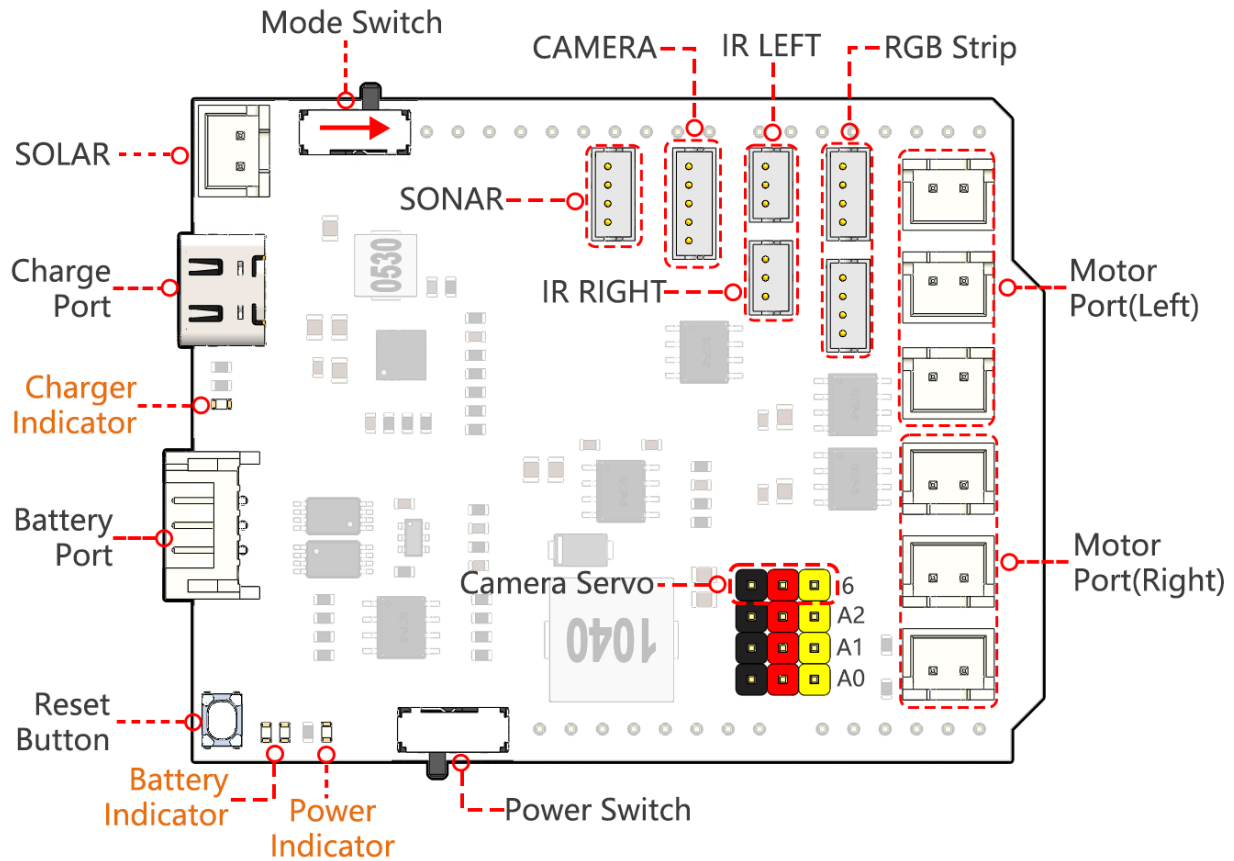
4.2 GalaxyRVR Shield



This is an all-in-one expansion board designed for Arduino by SunFounder, which contains various module ports such as motor, RGB strip, obstacle avoidance, grayscale, ESP32 CAM and ultrasonic module.

This expansion board also has a built-in charging circuit, which can charge the battery with PH2.0-3P interface, and the estimated charging time is 130 minutes.

Pinout

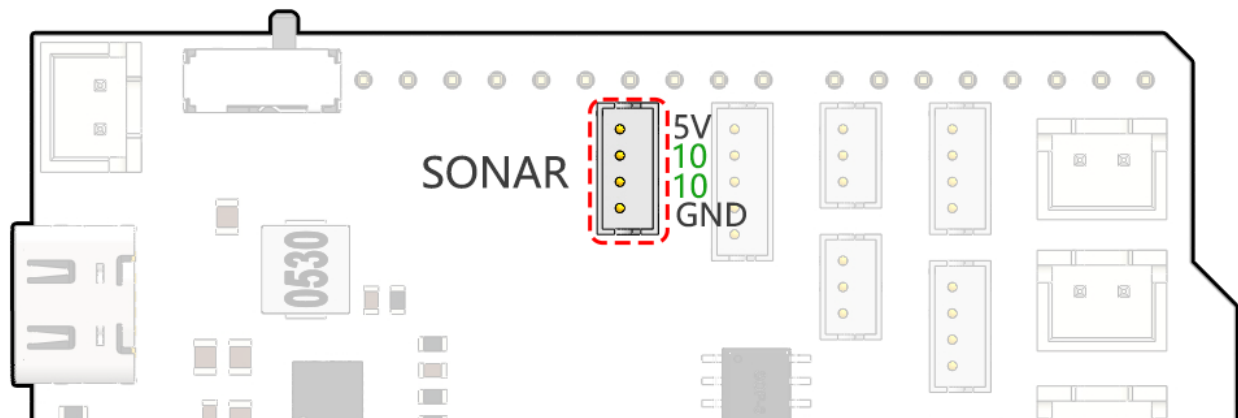


- **Charge Port**
 - After plugging into the 5V/2A USB-C port, it can be used to charge the battery for 130min.
- **Battery Port:**
 - 6.6V~8.4V PH2.0-3P power input.
 - Powering the GalaxyRVR Shield and Arduino board at the same time.
- **Reset Button**
 - Press this button to reset the program on the Arduino board.
- **Indicators**
 - **Charge Indicator:** Glows red when the shield is charging through the USB-C port.
 - **Power Indicator:** Glows green when the power switch is in the “ON” position.
 - **Battery Indicator:** Two orange indicators represent different battery levels. They flash during charging and turn off when the battery needs charging.
- **Power Switch**
 - Slide to ON to power on the GalaxyRVR.
- **Camera Servo**
 - The servo on the camera is connected here.

- The brown wire connects to “-”, the red wire connects to “+”, and the yellow wire connects to Pin 6.
- **Motor Port**
 - **Motor Port(Right)**: 3 motors can be connected, but all 3 motors are controlled by the same set of signal **pins 2 and 3**.
 - **Motor Port(Left)**: 3 motors can be connected, but all 3 motors are controlled by the same set of signal **pins 4 and 5**.
 - Port Type: XH2.54, 2P.
- **RGB Strip**
 - For connecting 2 RGB LED Strips, the three pins of the strip are connected to **12, 13 and 11** respectively.
 - Port Type: ZH1.5, 4P.
- **LEFT/RIGHT IR**
 - Used for connecting two IR obstacle avoidance modules.
 - The **left obstacle avoidance module** is connected to **pin 8**, the **right obstacle avoidance module** is connected to **pin 7**.
 - Port Type: ZH1.5, 3P.
- **CAMERA**
 - The Camera Adapter Board port.
 - Port Type: ZH1.5, 5P.
- **SONAR**
 - To connect the ultrasonic module, both Trig & Echo pins are connected on **pin 10** of the Arduino board.
 - Port Type: ZH1.5, 4P.
- **Mode Switch**
 - The ESP32-CAM and the Arduino board share the same RX (receive) and TX (transmit) pins.
 - So, when you’re uploading code, you’ll need to toggle this switch to the **right side** to disconnect the ESP32-CAM to avoid any conflicts or potential issues.
 - When you need to use the camera, toggle this switch to the **left side** so that the ESP32-CAM can communicate with the Arduino board.
- **SOLAR**
 - This is the port for the solar panel, which can charge the battery when plugged into the solar panel.
 - Port Type: XH2.54, 2P.

4.2.1 SONAR

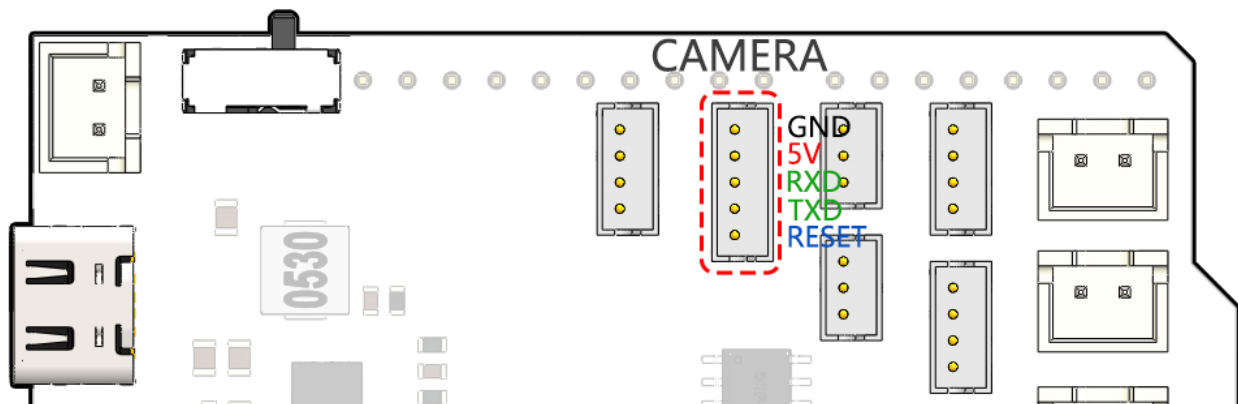
This is the pinout for the ZH1.5-4P ultrasonic port, with the Trig & Echo pins connected to pin 10 of the Arduino board.



4.2.2 CAMERA

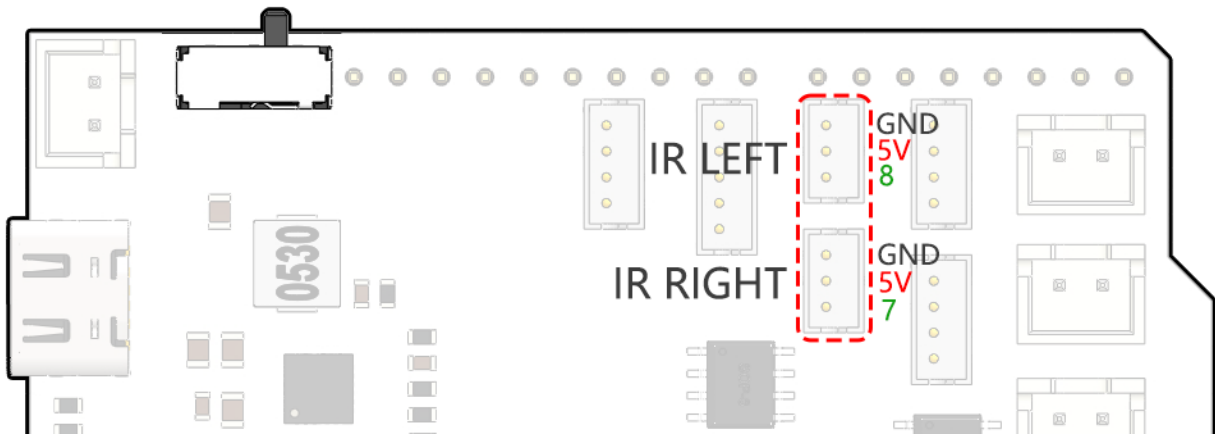
The camera adapter interface pin diagram is shown here, the type is ZH1.5-7P.

- TX and RX are used for ESP32 CAM.



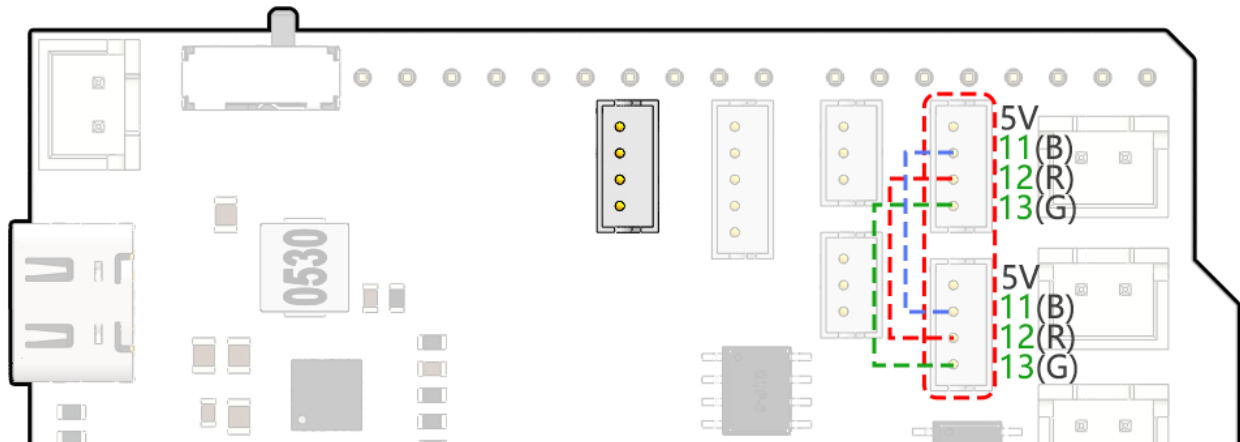
4.2.3 LEFT/RIGHT IR

These are the pins for the left and right obstacle avoidance modules.



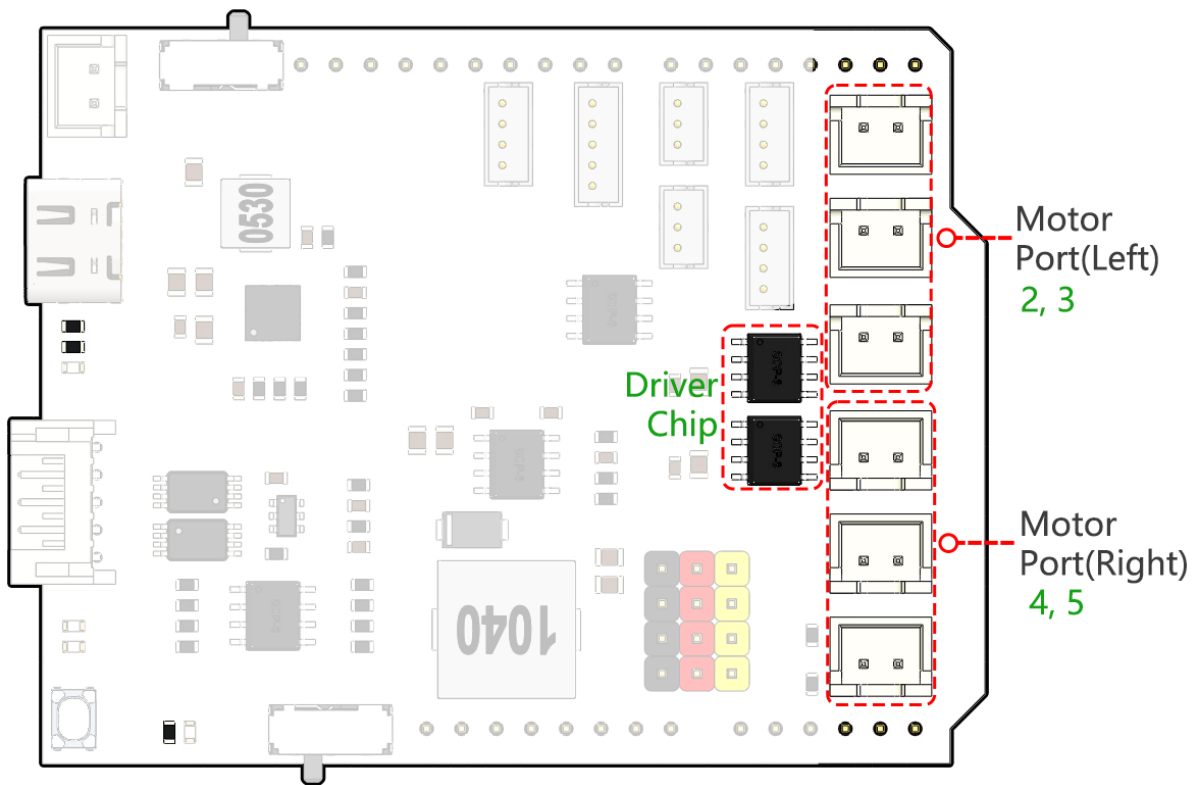
4.2.4 RGB Strip

Below is the pinout diagram of the two RGB LED Strip, they are connected in parallel and the pinouts are the same.

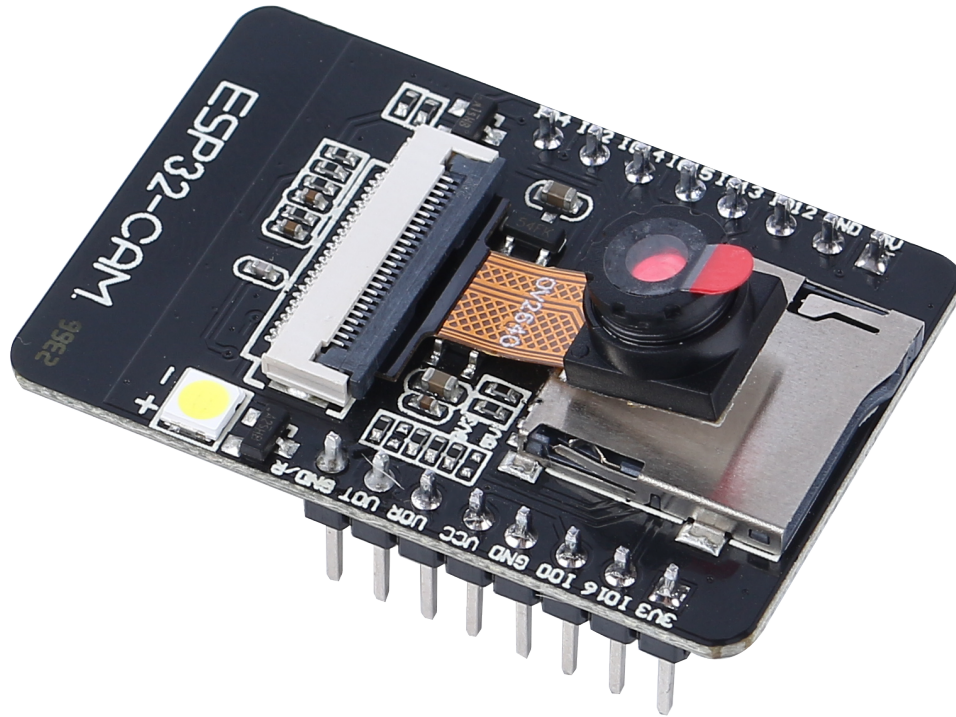


4.2.5 Motor Port

Here is the pinout of the 2 sets of motor ports.



4.3 ESP32 CAM



The ESP32-CAM is a very small camera module with the ESP32-S chip that costs approximately \$10. Besides the OV2640 camera, and several GPIOs to connect peripherals, it also features a microSD card slot that can be useful to store images taken with the camera or to store files to serve to clients.

The module can work independently as the smallest system, with a size of only 27*40.5*4.5mm, and a deep sleep current as low as 6mA.

ESP32-CAM can be widely used in various IoT applications, suitable for home smart devices, industrial wireless control, wireless monitoring, QR wireless identification, wireless positioning system signals and other IoT applications. It is an ideal solution for IoT applications.

Technical Specifications

Module Model	ESP32-CAM
Package	DIP-16
Size	27*40.5*4.5±0.2mm
SPI Flash	default 32Mbit
RAM	Internal 520KB + External 8MB PSRAM
Bluetooth	Bluetooth 4.2 BR/EDR and BLE standards
Wi-Fi	802.11 b/g/n/e/i
Support Interfaces	UARTSPII2CPWM
Support TF Card	up to 4G
IO Pins	9
Serial Port Speed	default 115200 bps
Image Output Format	JPEG(only OV2640 support),BMP,GRAYSCALE
Spectrum range	2400 ~2483.5MHz

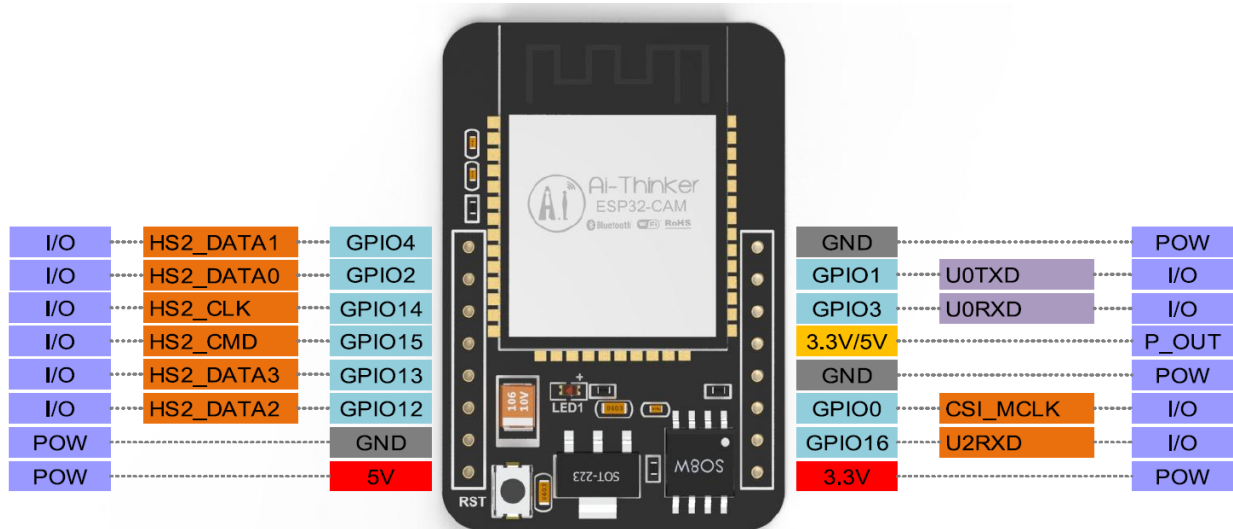
continues on next page

Table 1 – continued from previous page

Antenna Type	On-board PCB antenna, gain 2dBi
Transmit Power	802.11b: 17±2 dBm (@ 11Mbps)
	802.11g: 14±2 dBm (@ 54Mbps)
	802.11n: 13±2 dBm (@ MCS7)
Receive Sensitivity	CCK, 1 Mbps: -90dBm,
	CCK, 11 Mbps: -85 dBm
	6 Mbps (1/2 BPSK): -88 dBm
	54 Mbps (3/4 64-QAM): -70dBm
	MCS7 (65 Mbps, 72.2 Mbps): -67dBm
Power Consumption	Flash off: 180mA@5V,
	Flash on and brightness to maximum: 310mA@5V
	Deep-sleep: the lowest power consumption can reach 6mA@5V
	Moderm-sleep: minimum 20mA@5V
	Light-sleep: minimum 6.7mA@5V
Security	WPA/WPA2/WPA2-Enterprise/WPS
Power supply range	4.75-5.25V
Operating Temperature	-20 °C ~ 70 °C
Storage Environment	-40 °C ~ 125 °C , < 90%RH

ESP32-CAM Pinout

The following figure shows the ESP32-CAM pinout (AI-Thinker module).



- There are three **GND** pins and three pins for power: 3.3V, 5V and either 3.3V or 5V.
- **GPIO 1** and **GPIO 3** are the serial pins. You need these pins to upload code to your board.
- Additionally, **GPIO 0** also plays an important role, since it determines whether the ESP32 is in flashing mode or not. When **GPIO 0** is connected to **GND**, the ESP32 is in flashing mode.
- The following pins are internally connected to the microSD card reader:
 - GPIO 14: CLK
 - GPIO 15: CMD
 - GPIO 2: Data 0
 - GPIO 4: Data 1 (also connected to the on-board LED)

- GPIO 12: Data 2
- GPIO 13: Data 3

Note

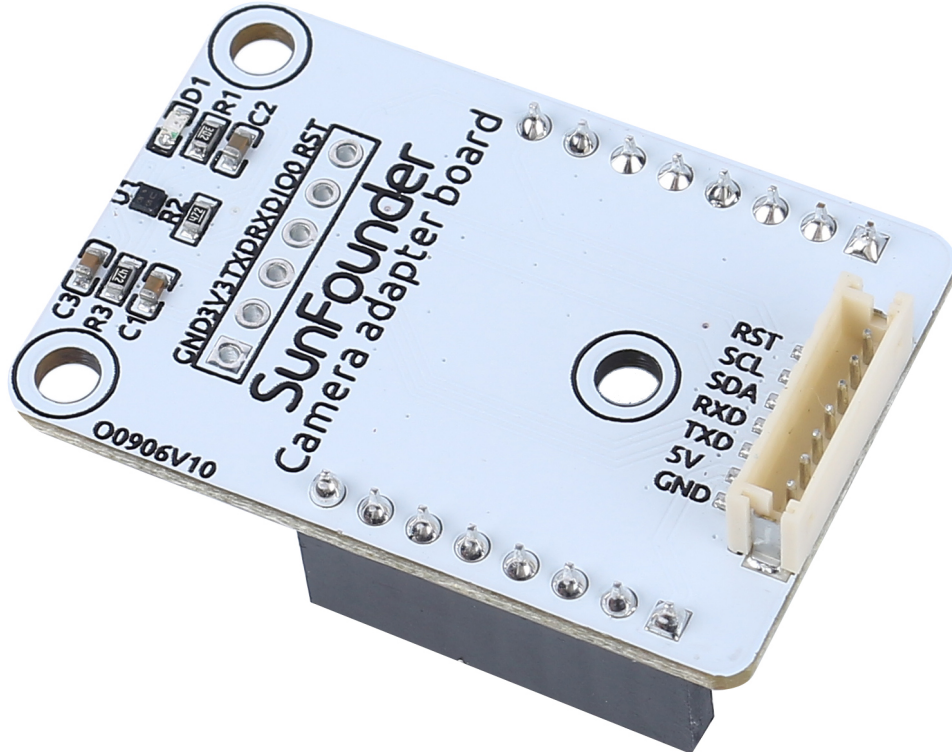
- Please make sure that the input power of the module is at least 5V 2A, otherwise the picture may have water lines.
- The ESP32 GPIO32 pin controls the camera power. When the camera is working, please pull GPIO32 low.
- Since GPIO0 is connected to the camera XCLK, please leave GPIO0 in the air when using it, and do not connect it to high or low level.
- The default firmware is already included in the factory, and no additional download is provided. Please be careful if you need to re-burn other firmware.

Document

- Schematic diagram:
- Camera specification (English version):

Note: All information above comes from

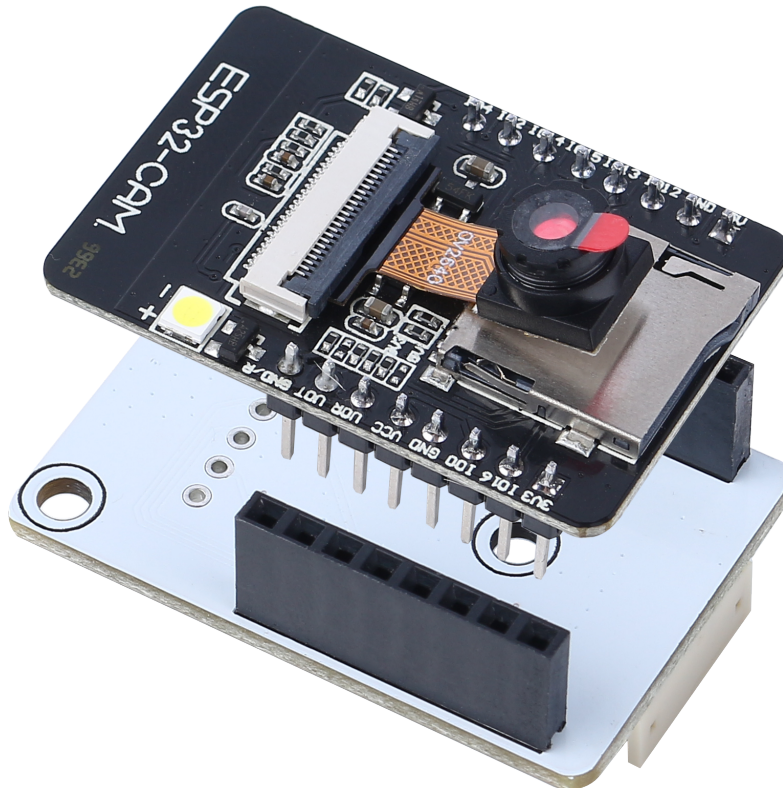
4.4 Camera Adapter Board



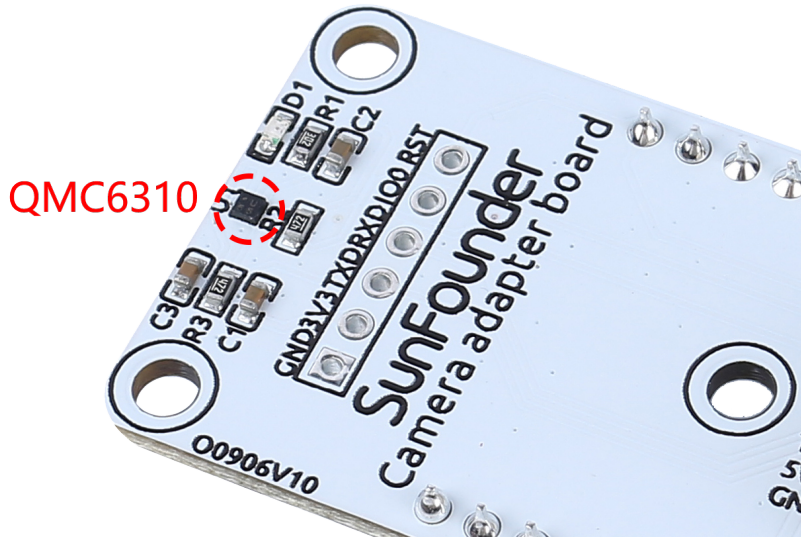
- **RST:** Used to reset the ESP32-CAM.
- **SCL:** Serial data pin for QMC6310

- **SDA**: Serial clock pin of the QMC6310
- **RXD**: The RXD of ESP32-CAM, you need to upload code to ESP32-CAM through these two serial pins, RXD and TXD.
- **TXD**: TXD of ESP32-CAM
- **5V**: 5V DC Supply Input
- **GND**: Ground Input

The Camera Adapter Board, as the name implies, is an expansion board for the ESP-32 CAM, used to expand the ESP32-CAM so that it can be secured to the robot, and can be easily wired.



Also because the geomagnetic chip QMC6310 is susceptible to interference from motors, we put it on this camera adapter board to keep it as far away from the motors as possible.

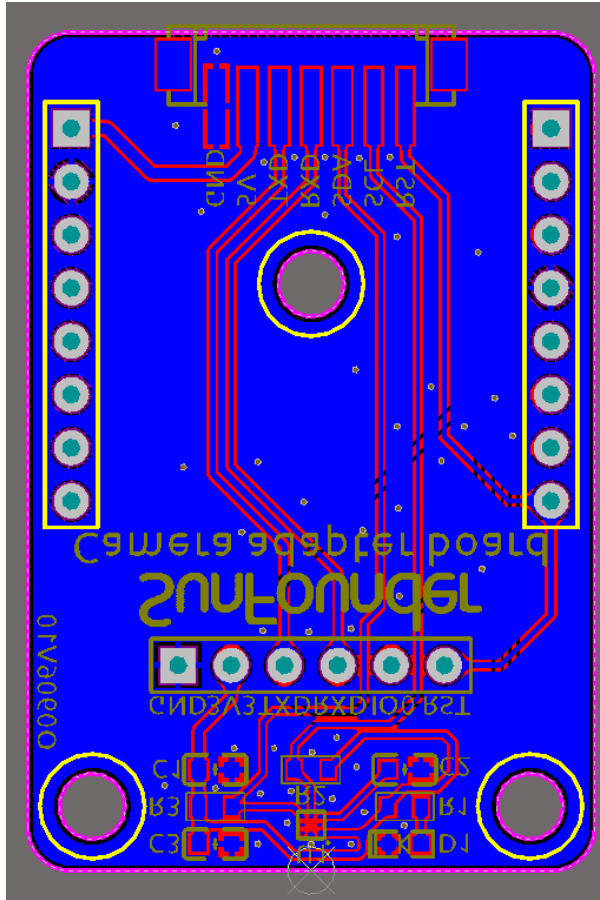


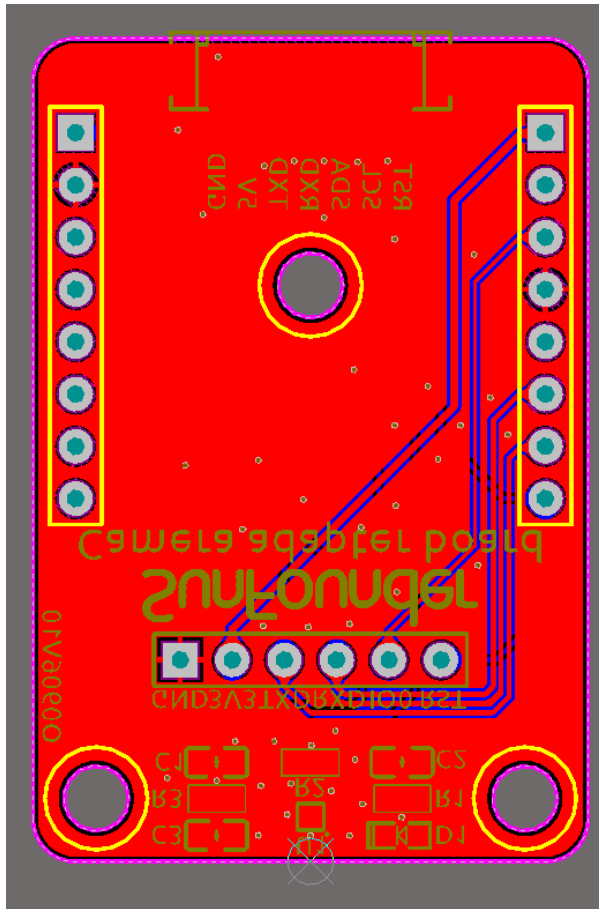
Features

- Working voltage: 5V
- Interface Model: ZH1.5, 7P
- Dimension: 40mm x 27mm x 15mm
- Communication protocol: UART and I2C

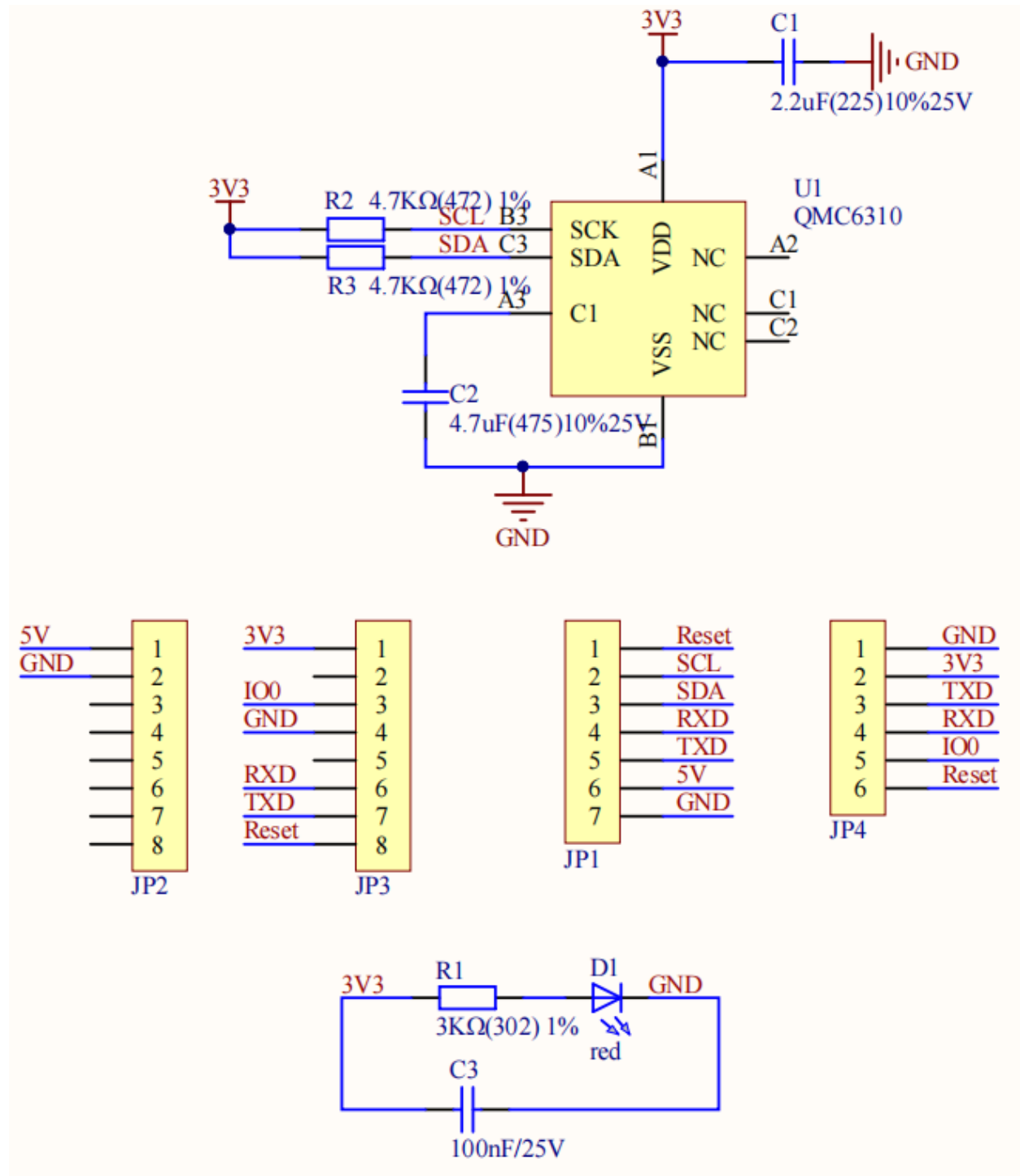
Documents

- PCB





- Schematic



About QMC6310

The QMC6310 is a three-axis magnetic sensor, which integrates magnetic sensors and signal condition ASIC into one silicon chip. This Land Grid Array package (LGA) is targeted for applications such as e-compass, map rotation, gaming and personal navigation in mobile and wearable devices.

The QMC6310 is based on state-of-the-art, high resolution, magneto-resistive technology. Along with the custom-designed 16-bit ADC ASIC, it offers the advantages of low noise, high accuracy, low power consumption, offset cancellation and temperature compensations. QMC6310 enables 1° to 2° compass heading accuracy. The I²C serial bus

allows for easy interface.

The QMC6310 is in a 1.2x1.2x0.53mm³ surface mount 8-pin LGA package.

•

4.5 Ultrasonic Module



- **TRIG:** Trigger Pulse Input
- **ECHO:** Echo Pulse Output
- **GND:** Ground
- **VCC:** 5V Supply

This is the HC-SR04 ultrasonic distance sensor, providing non-contact measurement from 2 cm to 400 cm with a range accuracy of up to 3 mm. Included on the module is an ultrasonic transmitter, a receiver and a control circuit.

You only need to connect 4 pins: VCC (power), Trig (trigger), Echo (receive) and GND (ground) to make it easy to use for your measurement projects.

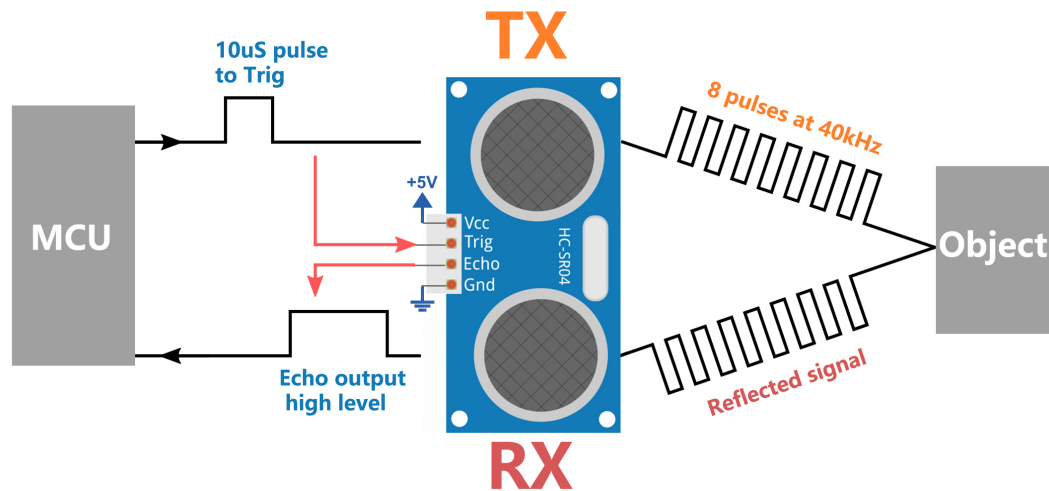
Features

- Working Voltage: DC5V
- Working Current: 16mA
- Working Frequency: 40Hz
- Max Range: 500cm
- Min Range: 2cm
- Trigger Input Signal: 10uS TTL pulse
- Echo Output Signal: Input TTL level signal and the range in proportion
- Connector: XH2.54-4P
- Dimension: 46x20.5x15 mm

Principle

The basic principles are as follows:

- Using IO trigger for at least 10us high level signal.
- The module sends an 8 cycle burst of ultrasound at 40 kHz and detects whether a pulse signal is received.
- Echo will output a high level if a signal is returned; the duration of the high level is the time from emission to return.
- Distance = (high level time x velocity of sound (340M/S)) / 2

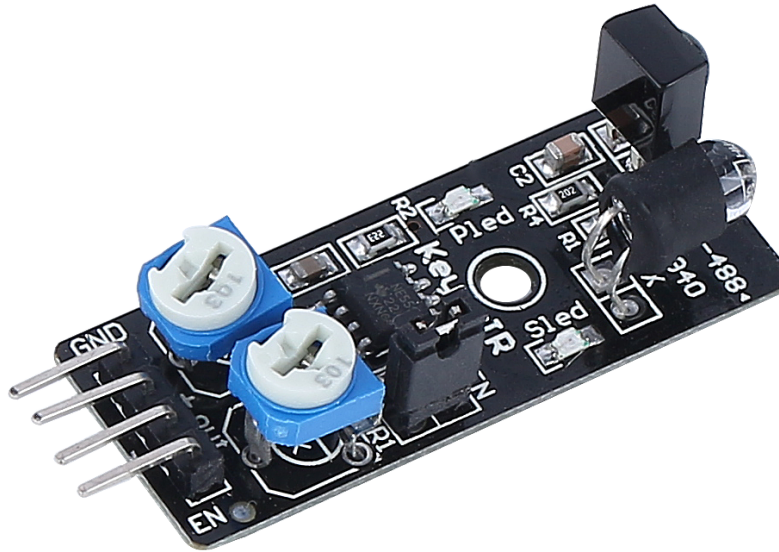


Application Notes

- This module should not be connected under power up, if necessary, let the module's GND be connected first. Otherwise, it will affect the work of the module.
- The area of the object to be measured should be at least 0.5 square meters and as flat as possible. Otherwise, it will affect results.

4.6 IR Obstacle Avoidance Module

This is an infrared obstacle avoidance module that can detect the presence of objects ahead. It is commonly used in robots, automation systems, and other intelligent devices. Its detection range is 2cm to 40cm, and objects of different colors have different levels of reflectivity. Therefore, the darker the object, the shorter the detection distance.

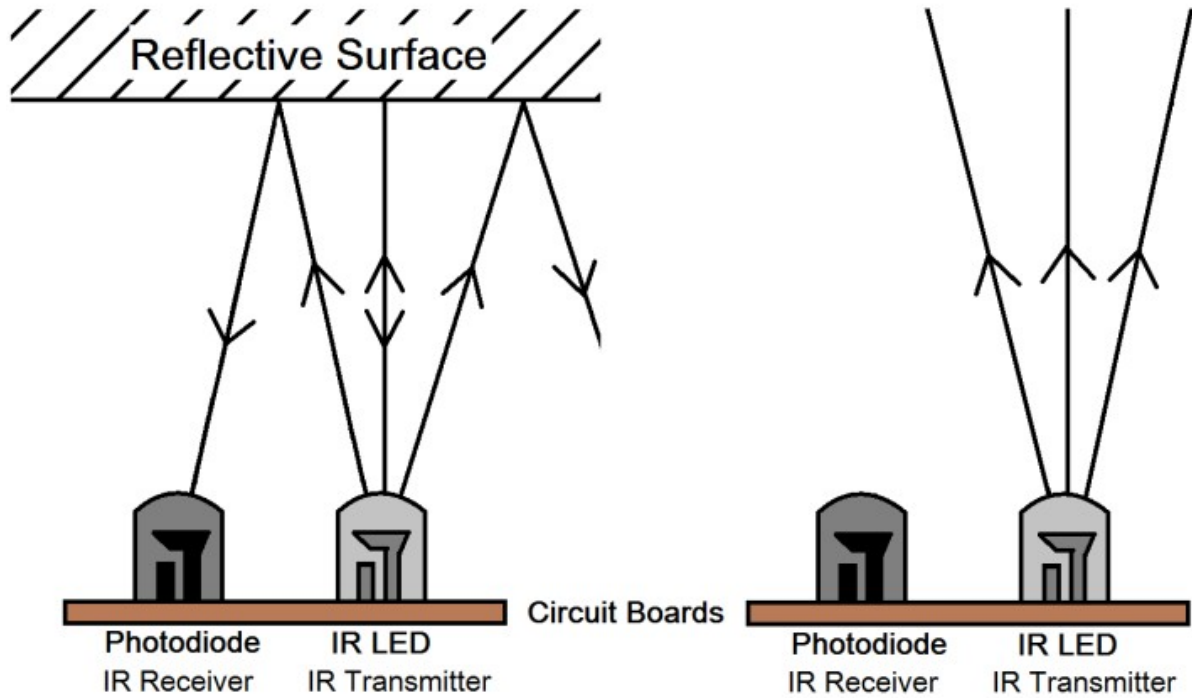


Here are the pin definitions:

- **GND**: Ground
- **+**: Power supply, 3.3 ~ 5V DC.
- **Out**: By default, it stays high and only goes low when it spots an obstacle.
- **EN**: This **enable** pin decides when the module should work. By default, it is connected to GND, meaning the module is always on the job.

How it works?

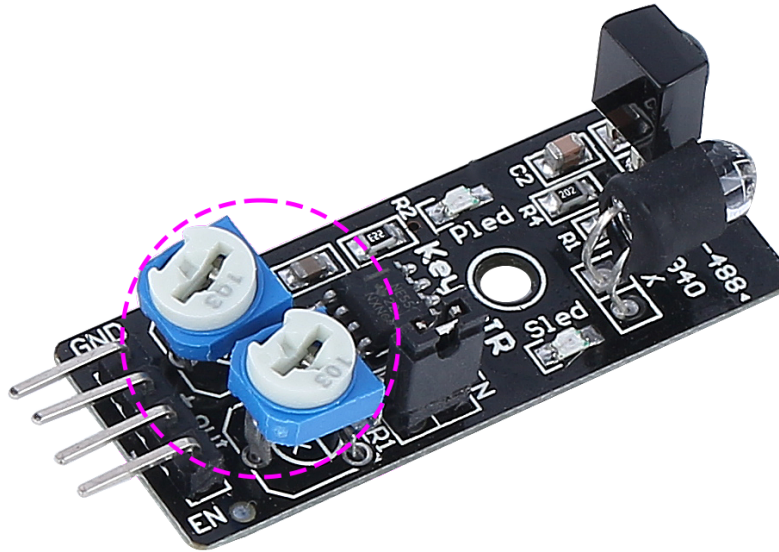
This module contains a pair of IR transmitting and receiving components. Basically, the transmitter emits infrared light, when the emitted infrared light encounters an obstacle, it is reflected back and received by the receiver. Upon detection, the indicator lights up. After circuit processing, it outputs a low level signal.



The **EN** pin's low-level state activates the module, with the jumper cap securing the **EN** pin to the GND. To control the **EN** pin programmatically, remove the jumper cap.



There are two potentiometers on the module, one for adjusting the transmitting power and one for adjusting the transmitting frequency, and by adjusting these two potentiometers you can adjust its effective distance.



Adjust the detection distance

The obstacle avoidance module's detection range must be precisely calibrated for optimal performance, as default factory settings may not align with specific requirements.

Calibration involves the following steps:

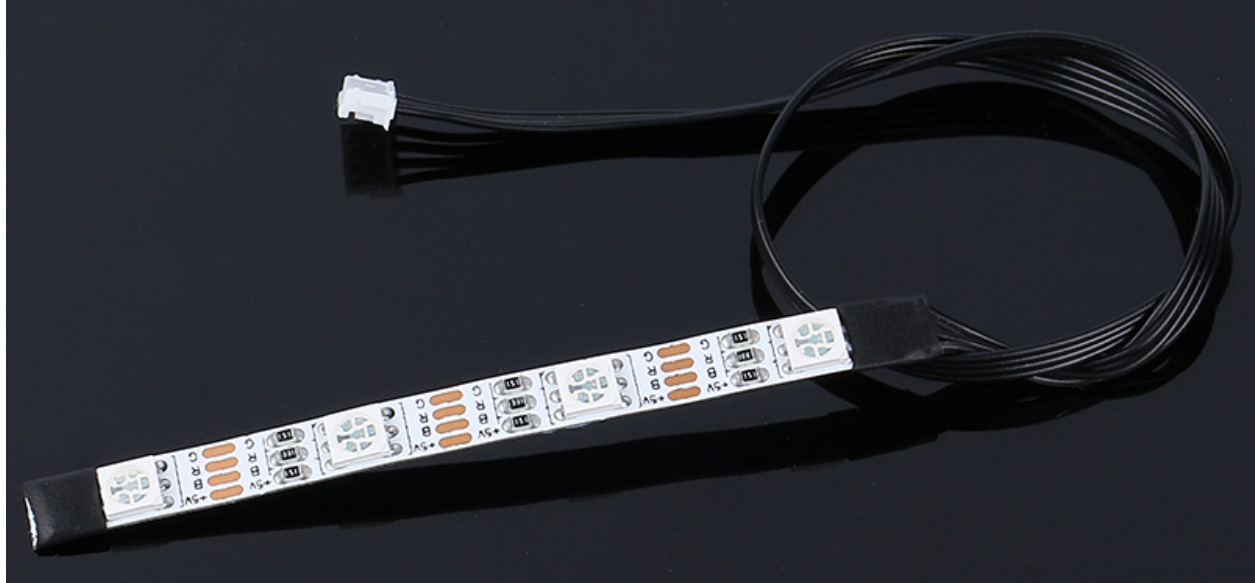
1. Alignment of the Module: Start with the right-hand side obstacle avoidance module. Transportation can occasionally displace the alignment of the module's infrared transmitter and receiver. They should be manually realigned to ensure accuracy.
2. Place an obstacle about 20 cm directly in front of the right module. The box in which our Rover kit came is a good choice for this! Now, turn the potentiometer on the module until the indicator light on the module just lights up. Then, keep moving the obstacle back and forth to check if the indicator light comes on at the desired distance. If the light doesn't turn on at the correct distance or if it remains on without going out, you'll need to adjust the other potentiometer.
3. Repeat the same process for another module.

Features

- operating voltage: 3.3 V to 5 V
- output: digital (on/off)
- detection threshold: adjustable by 2 potentiometers
- distance range: 2 to 40 cm
- adjustment R5: frequency adjustment 38 kHz (already optimized)
- adjustment R6: IR LED duty cycle adjustment (already optimized)
- operating temperature: -10 °C to +50 °C
- effective angle: 35°
- I/O interface: 4 wire interface (- / + / S / EN)
- dimensions: 45 x 16 x 10 mm

- weight: 9 g

4.7 4 RGB LEDs Strip



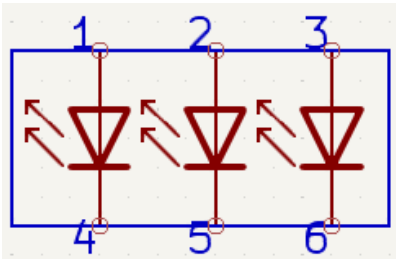
- **+5V**: Common anode of the three LEDs and needs to connect to DC 5V
- **B**: Cathode of the blue LED
- **R**: Cathode of the red LED
- **G**: Cathode of the green LED

This RGB LED strip features four R5050 RGB LEDs, capable of creating any color shade by combining the three primary colors: red, blue, and green.

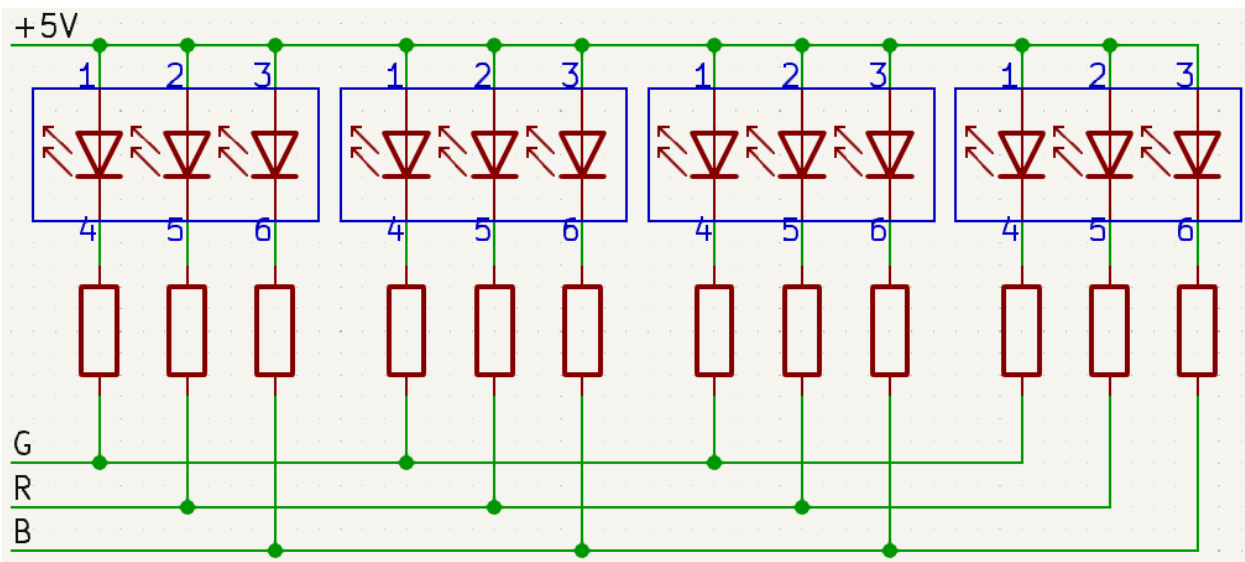
The R5050 RGB LEDs are designed with a common anode configuration. Each LED on the strip functions as an independent circuit, allowing you to cut the strip at designated points without affecting other sections. The strip's flexibility and adaptability are enhanced by its construction on an FPC board, which is backed with double-sided adhesive for easy installation.

What is R5050 RGB LED?

The R5050 RGB LED is a type of LED that combines red, blue, and green light-emitting diodes in a single package. Each LED within this package has its own pin, allowing for individual control. This configuration enables the production of a wide range of colors by varying the intensity of each LED.



In a typical application, multiple R5050 RGB LEDs are arranged on a flexible circuit in a smart configuration. This is done by connecting the “positive” ends (anodes) of all LEDs together, while the “negative” ends (cathodes) are connected to their respective color lanes. This means that all green cathodes are connected together, all red cathodes are connected together, and all blue cathodes are connected together. This arrangement allows for efficient control of color blending and light intensity, making these LEDs popular in applications where customizable color lighting is desired, such as in decorative lighting, signage, and display technologies.



Features

- Work Voltage: DC5V
- Color: Full color RGB
- Working Temperature: -15-50

- RGB Type: 5050RGB
- Current: 150mA (single circuit)
- Power: 1.5W
- Light Strip Thickness: 2mm
- Light Strip Width: 5.5mm
- Cable: ZH1.5-4P, 25cm, 28AWG, Black

4.8 Servo

A servo is a specialized motor known for its precision in controlling specific angular positions.



- **Brown Line:** GND
- **Orange Line:** Signal pin, connect to the PWM pin of main board.
- **Red wire:** VCC

Unlike regular motors that spin continuously, a servo can move to a precise position and hold it accurately. It achieves this through a combination of gears, a potentiometer, and control circuitry. Servos are commonly used in various applications that require precise control over the position of objects or mechanisms.

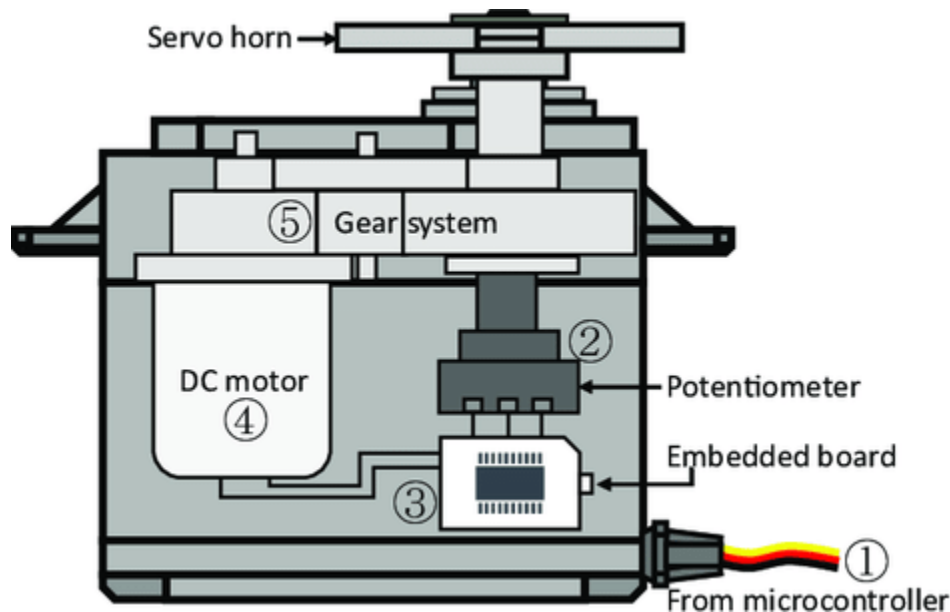
Features

- **Motor Type:** Core motor
- **Operating Voltage:** 4.8~6V DC
- **Standby Current:** 4 mA
- **Consumption Current(at 4.8V No Load):** 50mA
- **Consumption Current(at 6 V no load):** 60mA
- **Stall Current(at locked 4.8V):** 550mA
- **Stall Current(at locked 6V):** 650mA
- **Rated Torque:** 4.8V, 0.6 kgf·cm; 6V, 0.7 kgf·cm
- **Max. Torque:** 4.8V, 1.4 kgf·cm; 6V, 1.6 kgf·cm

- **No Load Speed:** 4.8V, 0.14sec/60°; 6V, 0.12sec/60°
- **Note:** Torsion protection:0.9 kgf.cm;Power failure protection after 5 seconds of continuous
- **Operating Temperature Range:** -10°C~+50°C
- **Storage Temperature Range:** -20°C~+60°C
- **Operating Humidity Range:** 90%RH
- **Storage Humidity Range:** 90%RH
- **Weight:** 10± 0.5g
- **Material:** ABS
- **Operating Angle:** 180°±10°(500~2500us)
- **Mechanical Limit Angle:** 360°
- **Left & Right Travelling Angle:** 6°
- **Centering deviation:** 1°
- **Back Lash:** 4 us
- **Amplifier Type:** Digital
- **Cable Materia:** 1.08,19 PVC
- **Cable Length:** 245±5mm(Exsert without plugs)
- **Connector Type:** JR2.54mm/3Pin

Operating Principle

Inside a servo, essential components contribute to its unique functionality. At its core, a servo incorporates a conventional motor, this motor is intricately linked to a large gear, which in turn engages with a smaller gear on the motor shaft. This gearing arrangement efficiently converts the motor's rapid circular motion into slower yet potent movements.



But the real magic happens within the servo, thanks to a minuscule electronic marvel known as a “potentiometer” and sophisticated “control circuitry.” When the servo undergoes movement, the potentiometer rotates, altering its electrical resistance. The control circuitry detects and interprets this change in resistance with remarkable precision, thereby determining the servo’s exact position. This is a testament to its ingenuity.

In the realm of servo control, a unique signaling method called “pulse-width modulation” or PWM comes into play. By skillfully adjusting the width of these pulses, operators can command the servo to move with precision and maintain its position. This is the essence of servo motor technology, a realm where precision and control converge to enable an array of applications.

4.9 TT Motor



This is a TT DC gear motor with a gear ratio of 1:120. It comes with two 250mm wires with an XH2.54-2P connector. It can be powered with 3VDC.

How Motors Work

A motor functions as the heart of a machine, transforming electrical energy into mechanical energy. This conversion brings to life various devices, from children’s toys and household appliances to large vehicles.

Here’s the process:

When electricity flows into a motor, it generates a magnetic field. This field interacts with other magnets within the motor, prompting it to spin. This spinning action, akin to a top whirling around, can then drive the movement of wheels, propellers, or other moving parts in a machine.

The TT Gear Motor is a specialized type of motor. It combines a standard motor with a series of gears, all housed within a durable plastic shell.

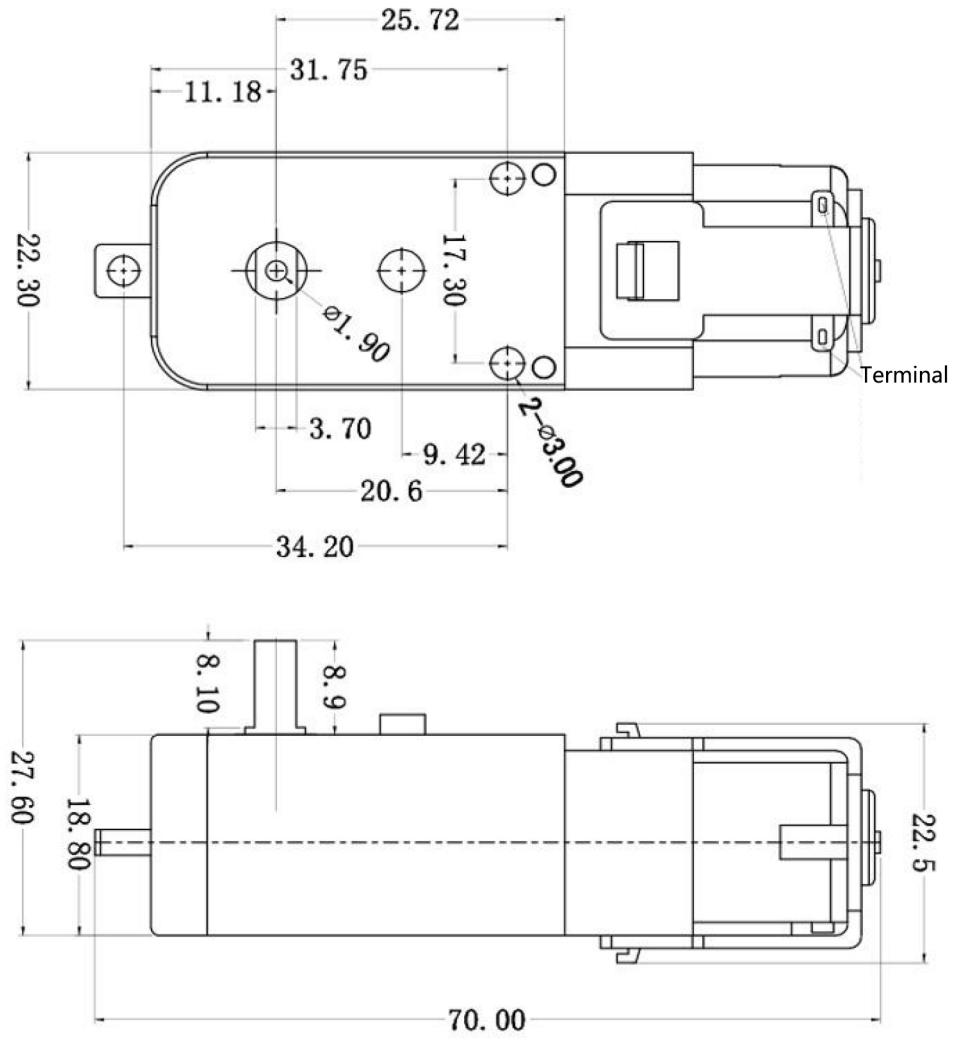
As the motor spins, the gears effectively transmit this rotational motion to the wheels of our rover. The integration of gears is pivotal, as it amplifies torque. This increased torque capacity enables the motor to maneuver larger and heavier loads, an essential capability in various applications.

Features

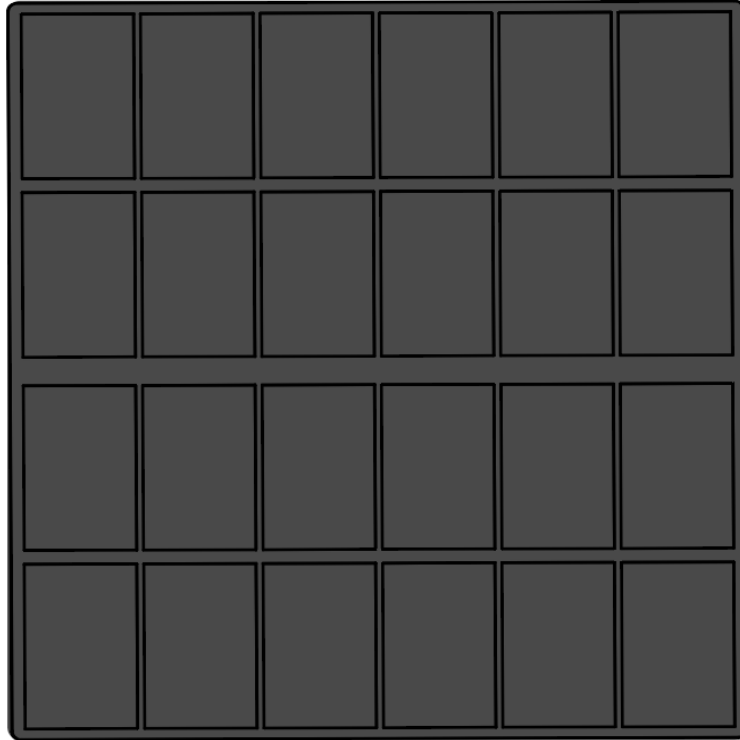
- Suggested Voltage 3V~4.5V DC
- Number of Shafts: Single shaft
- Gear Ratio: 1:120
- No load current: 130mA
- No load speed: $38\text{rpm} \pm 8\%$
- Starting Voltage: 2V (max.) under no load
- Output torque: 3V 1.2kgf.cm
- Useful life: 70-120H
- Direction of rotation: Bi-directions
- Body Dimensions: 70 x 22.5 x 36.6mm
- Wires: Gray and Black, 24AWG, 250mm
- Connector: White, XH2.54-2P
- Weight: 28.5g

Dimensional Drawing

Unit: mm



4.10 Solar Panel



Solar panels are devices that convert sunlight into electricity. They are made up of photovoltaic (PV) cells, which are made of semiconductor materials such as silicon. When sunlight hits a PV cell, it knocks electrons loose from their atoms. These electrons flow through the cell, creating an electric current.

Solar panels can be used to generate electricity for a variety of purposes, including powering homes, businesses, and even entire communities. They are a clean and renewable source of energy that can help reduce our reliance on fossil fuels.

Features

- Output power: 6V/660mA
- Time to fully charge the battery: 7.2h (theoretical value, assuming strong sunlight)
- Size: 170mm x 170mm
- Wires: Gray and Black, 24AWG, 200mm
- Connector: White, XH2.54-2P

4.11 18650 Battery



- **VCC:** Battery positive terminal, here there are two sets of VCC and GND is to increase the current and reduce the resistance.
- **Middle:** To balance the voltage between the two cells and thus protect the battery.
- **GND:** Negative battery terminal.

This is a custom battery pack made by SunFounder consisting of two 18650 batteries with a capacity of 2000mAh. The connector is XH2.54 3P, which can be charged directly after being inserted into the shield.

Features

- Battery charge: 5V/2A
- Battery output: 5V/5A
- Battery capacity: 3.7V 2000mAh x 2
- Battery life: 90min
- Battery charge time: 130min
- Connector: XH2.54 3P

5.1 Q1: Compilation error: SoftPWM.h: No such file or directory

If you get a “Compilation error: SoftPWM.h: No such file or directory” prompt, it means you don’t have the SoftPWM library installed.

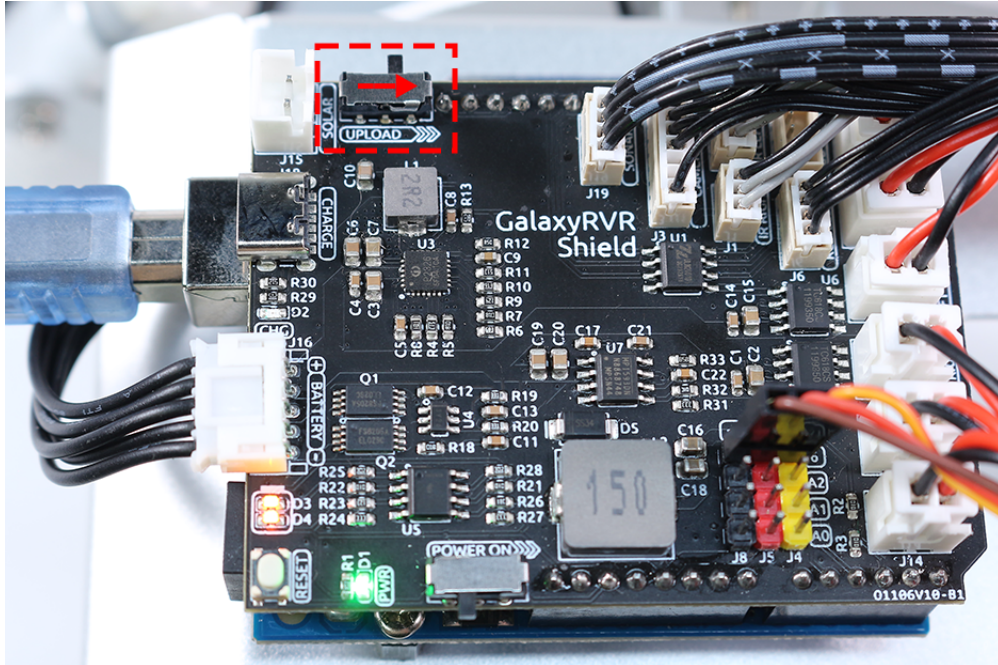
Please install the two required libraries SoftPWM and SunFounder AI Camera as shown.

5.2 Q2: avrdude: stk500_getsync() attempt 10 of 10: not in sync: resp=0x6e?

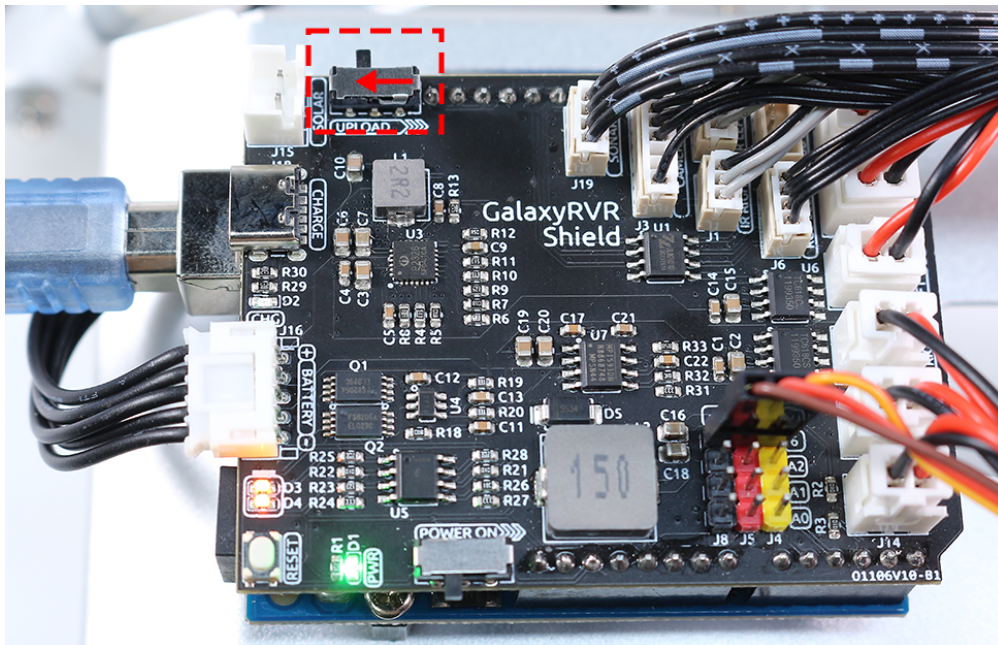
If the following message keeps appearing after clicking the **Upload** button when the board and port have been selected correctly.

```
avrdude: stk500_recv(): programmer is not responding
avrdude: stk500_getsync() attempt 1 of 10: not in sync: resp=0x00
avrdude: stk500_recv(): programmer is not responding
avrdude: stk500_getsync() attempt 2 of 10: not in sync: resp=0x00
avrdude: stk500_recv(): programmer is not responding
avrdude: stk500_getsync() attempt 3 of 10: not in sync: resp=0x00
At this point, you need to make sure that the ESP32 CAM is unplugged.
```

The ESP32-CAM and the Arduino board share the same RX (receive) and TX (transmit) pins. So, before you’re uploading code, you’ll need to first disconnect the ESP32-CAM to avoid any conflicts or potential issues.



After the code is successfully uploaded, if you need to use the ESP32 CAM, then you need to move the switch to the left to start the ESP32 CAM.



5.3 Q3: How can I use the STT mode on my Android device?

The STT mode requires the Android mobile device to be connected to the Internet and to install the Google service component.

Now follow the steps below.

1. Modify the AP mode of `galaxy-rvr.ino` file to STA mode.

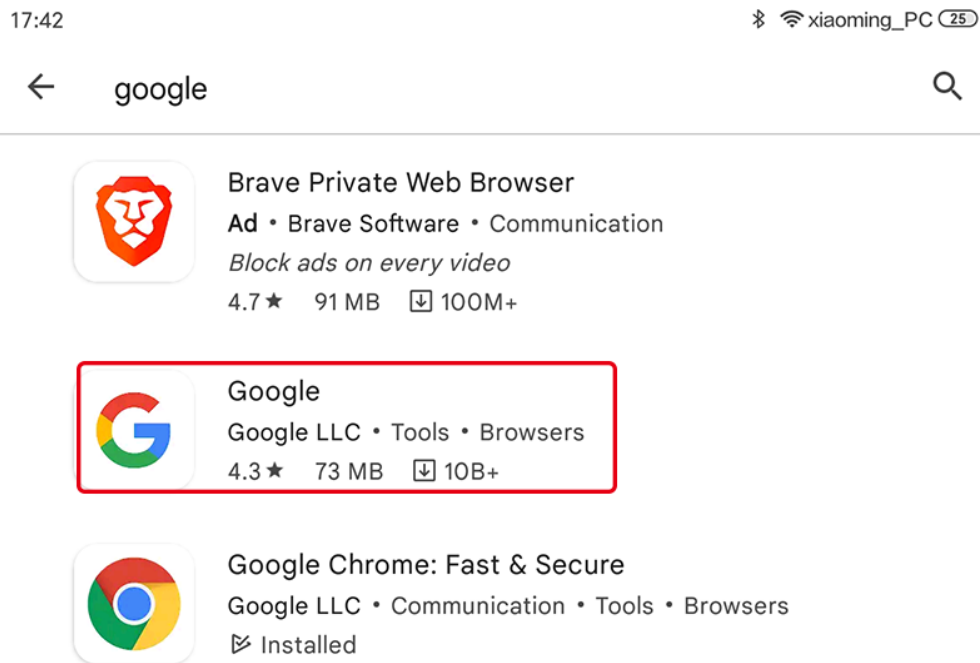
- Open the `galaxy-rvr.ino` file located in the `galaxy-rvr-main\galaxy-rvr` directory.
- Then comment out the AP mode related code. Uncomment the STA mode related code and fill in the SSID and PASSWORD of your home Wi-Fi.

```
/** Configure Wifi mode, SSID, password*/
// #define WIFI_MODE WIFI_MODE_AP
// #define SSID "GalaxyRVR"
// #define PASSWORD "12345678"

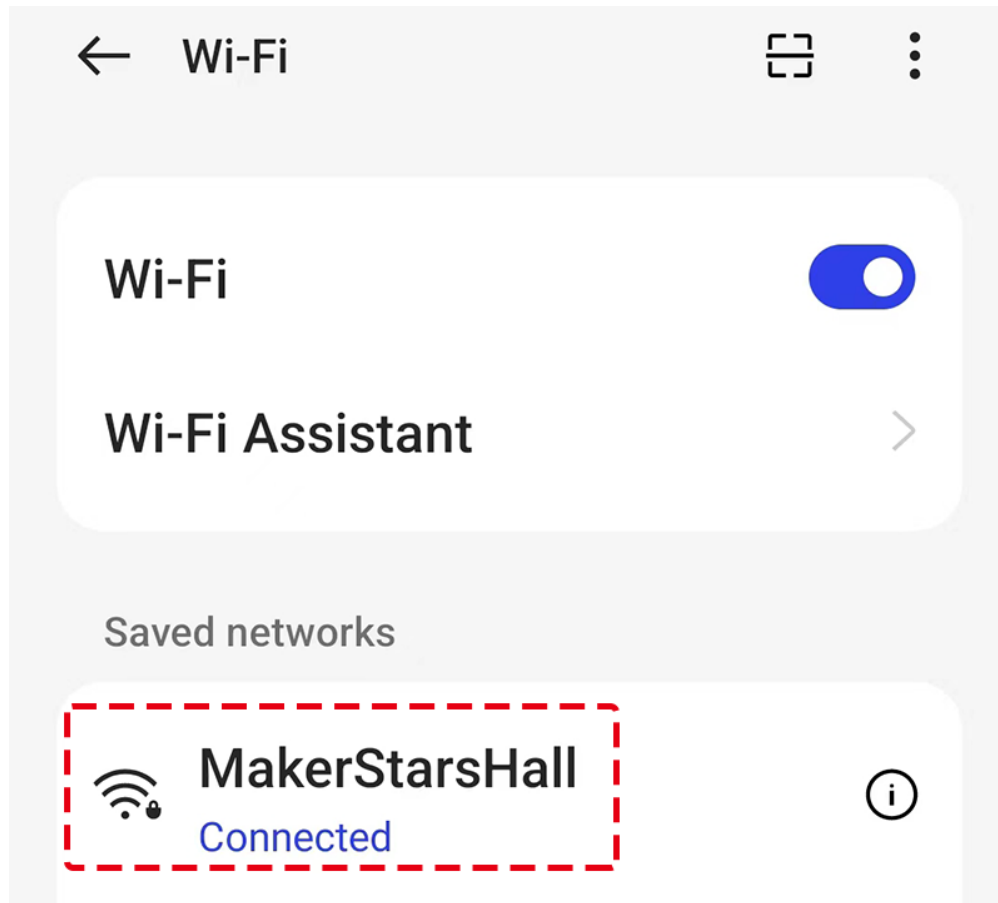
#define WIFI_MODE WIFI_MODE_STA
#define SSID "xxxxxxxxxx"
#define PASSWORD "xxxxxxxxxx"
```

- Save this code, select the correct board (Arduino Uno) and port, then click the **Upload** button to upload it to the R3 board.

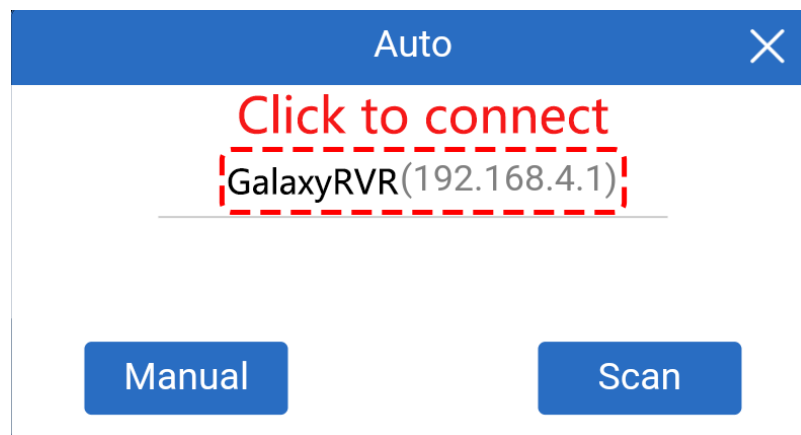
2. Search google in Google Play, find the app shown below and install it.



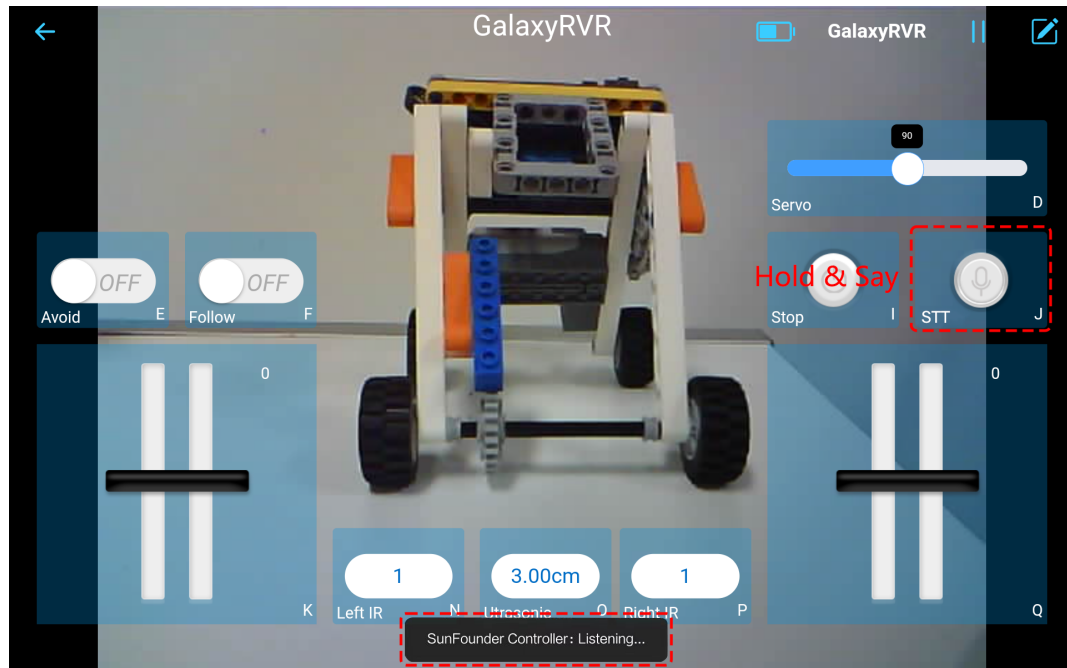
3. Connect your mobile device to the same Wi-Fi as you wrote in the code.



4. Open the controller previously created in SunFounder Controller and connect it to GalaxyRVR through the  button.



5. Tap and hold the **STT(J)** widget after clicking the  button. A prompt will appear indicating that it is listening. Say the following command to move the car.



- stop: All movements of the rover can be stopped.
- forward: Let the rover move forward.
- backward: Let the rover move backward.
- left: Let the rover turn left.
- right: Let the rover turn right.

5.4 Q4: About the ESP32 CAM Firmware

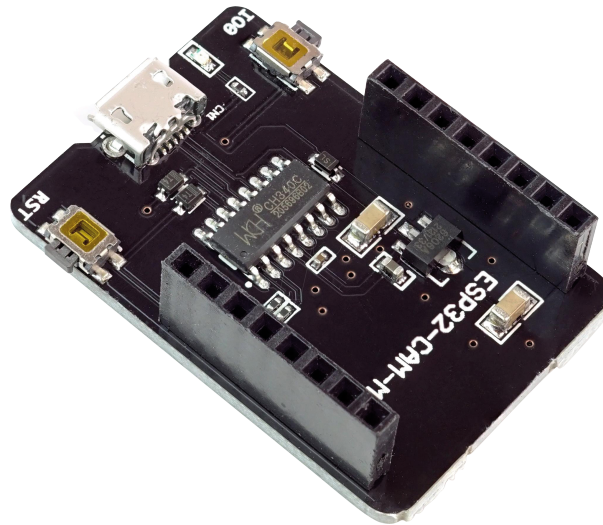
Here is the firmware link of ESP32 CAM:

5.5 Q5: How to Flash New Firmware to an ESP32 CAM?

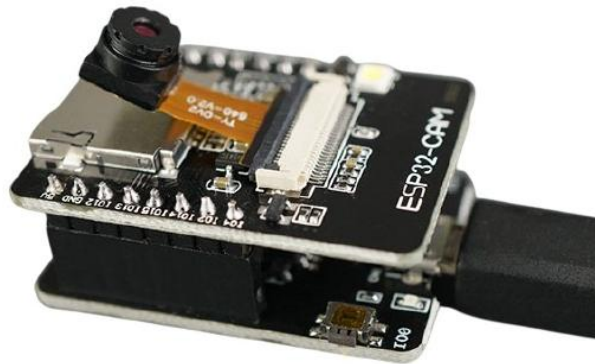
The camera module comes pre-flashed from the factory. However, if you encounter a data corruption issue, you can re-flash it with new firmware using the Arduino IDE. Here's how:

1. Prepare the Programmer

1. First, get a programmer ready.



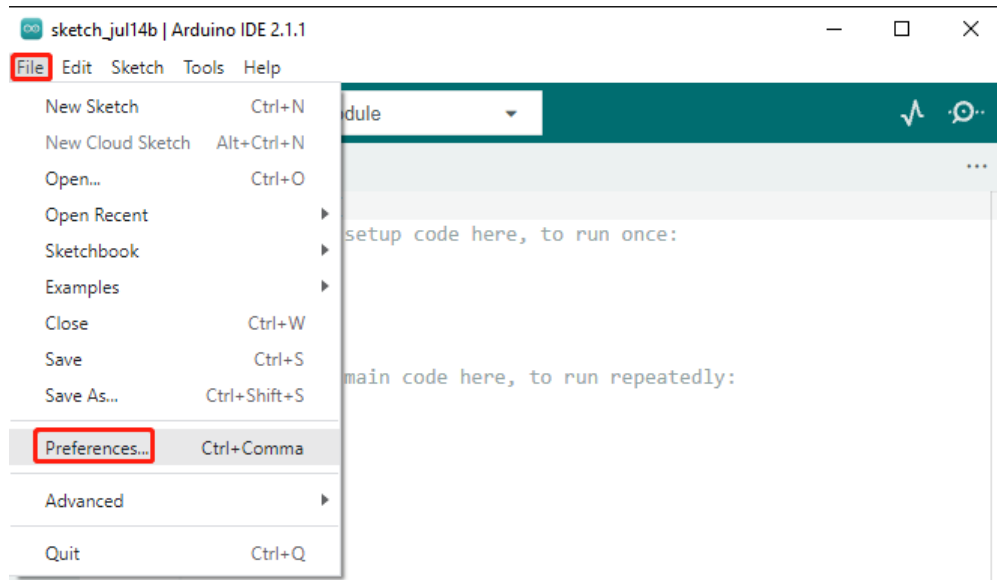
2. Insert the ESP32 CAM into the programmer and then plug the programmer into your computer.



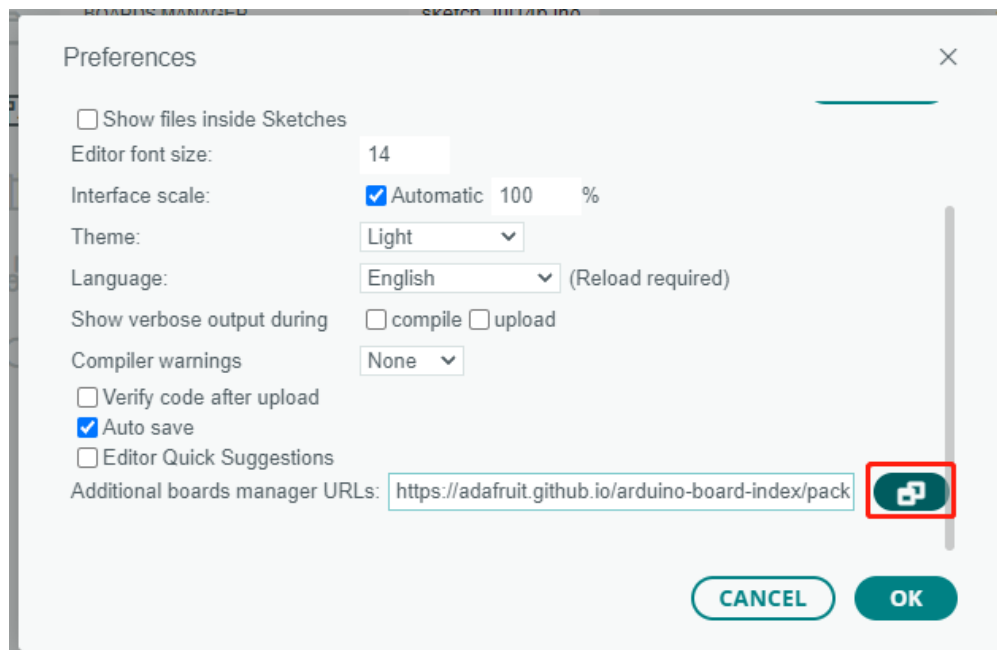
2. Install the ESP32 Board

To program the ESP32 microcontroller, you must install the ESP32 board package in the Arduino IDE. Follow these steps:

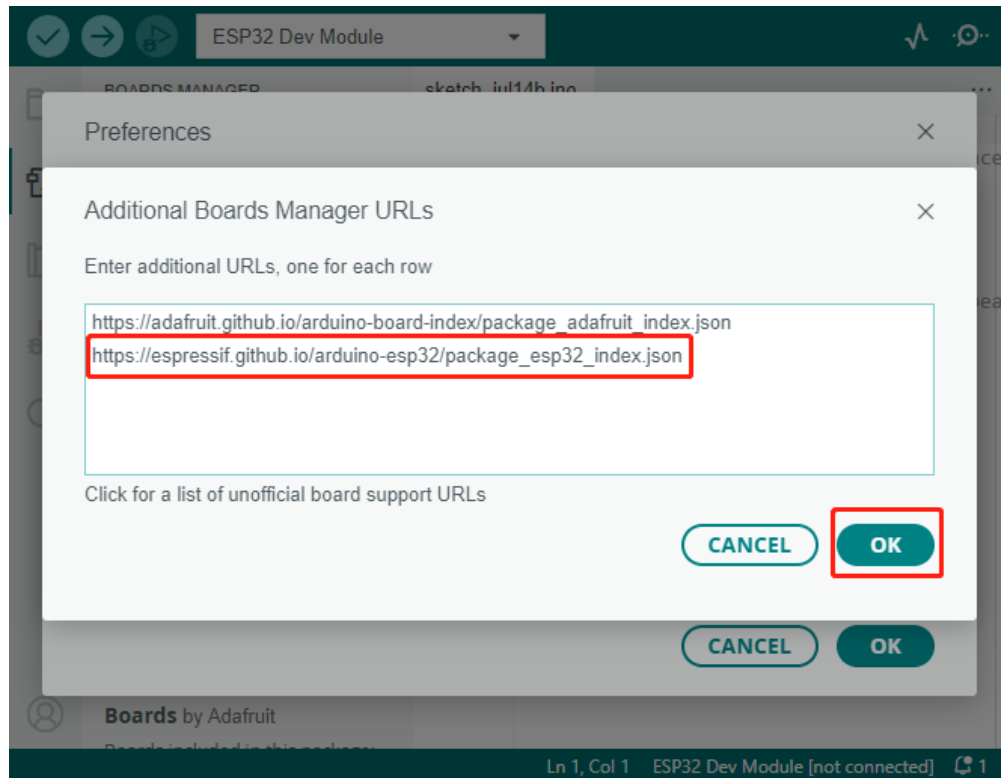
1. Go to **File** and select **Preferences** from the drop-down menu.



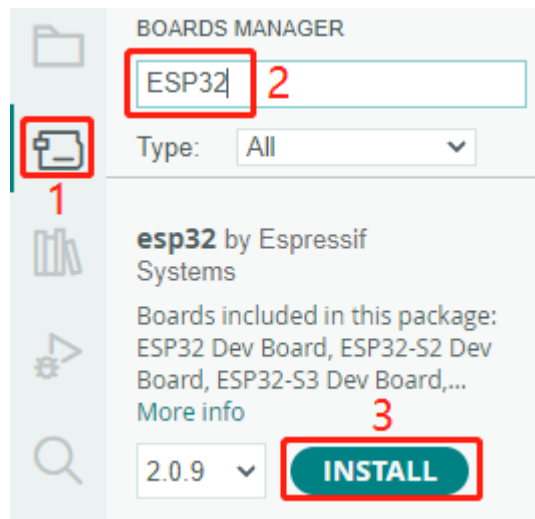
2. In the **Preferences** window, find the **Additional Board Manager URLs** field. Click on it to enable the text box.



3. Add this URL to the **Additional Board Manager URLs** field: https://espressif.github.io/arduino-esp32/package_esp32_index.json. This URL links to the package index file for ESP32 boards. Click **OK** to save the changes.

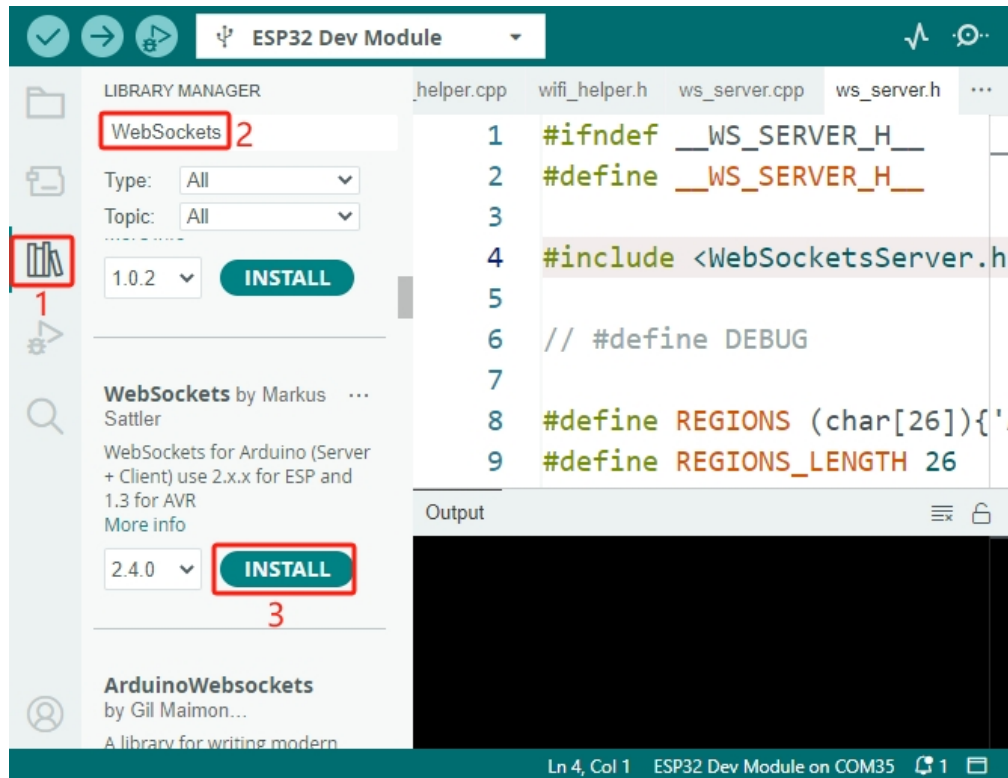


4. In the **Boards Manager** window, search for **ESP32**. Click the **Install** button to begin installation. This downloads and installs the ESP32 board package.

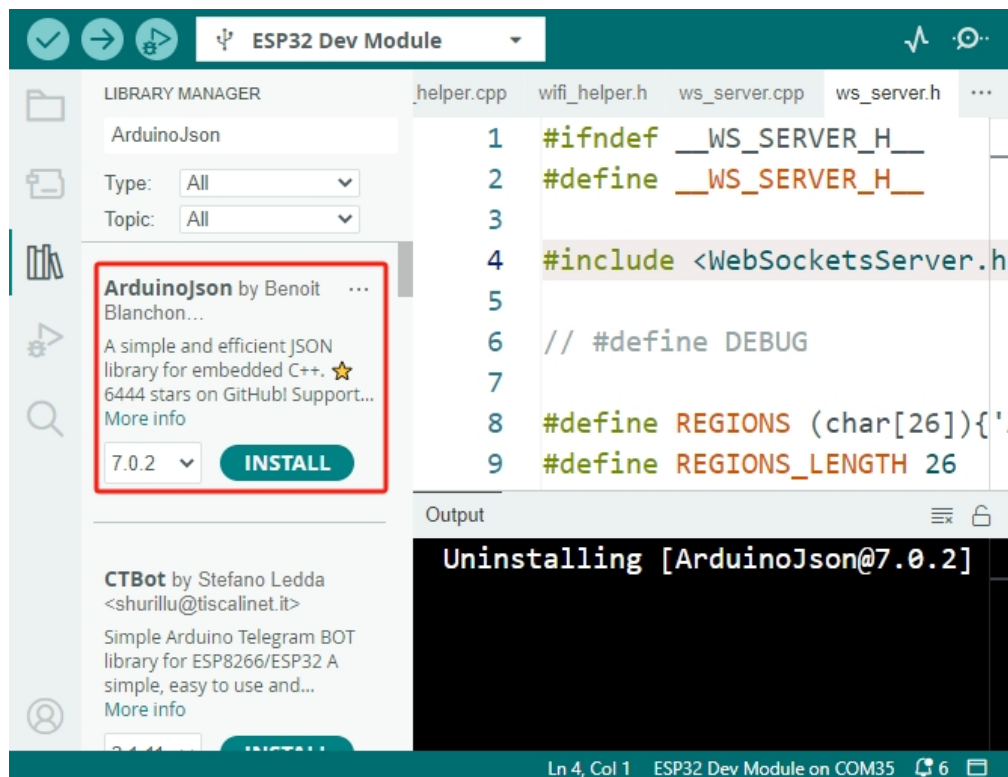


3. Install the Libraries

1. Install the WebSockets library from the **LIBRARY MANAGER**.



2. Follow the same steps to install the ArduinoJson library.

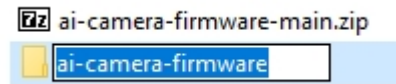


3. Download and Upload Firmware

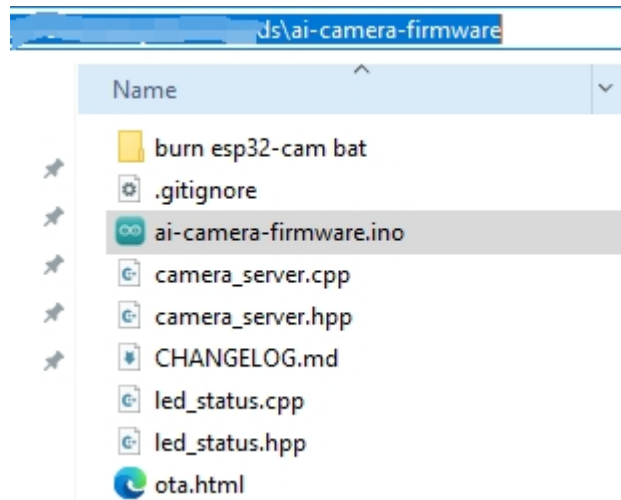
1. Download the firmware file.

- ai-camera-firmware

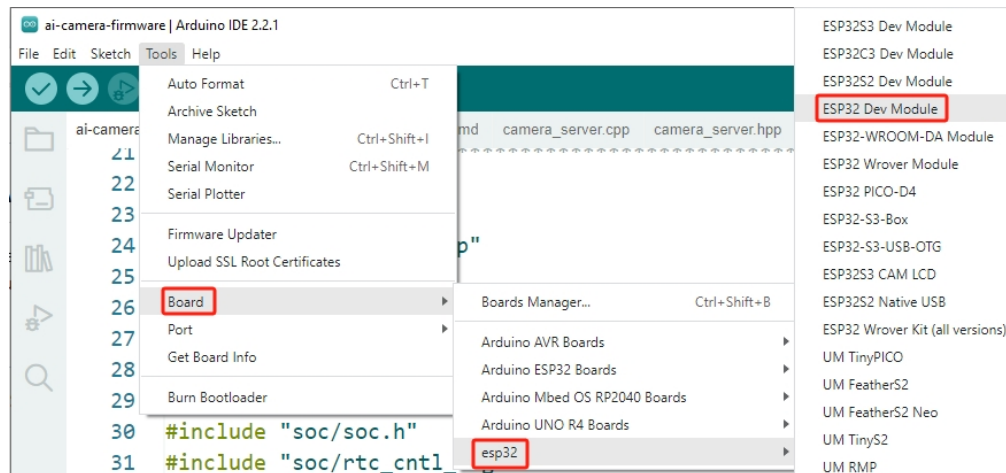
2. Extract the downloaded firmware file and rename the extracted folder from ai-camera-firmware-main to ai-camera-firmware.



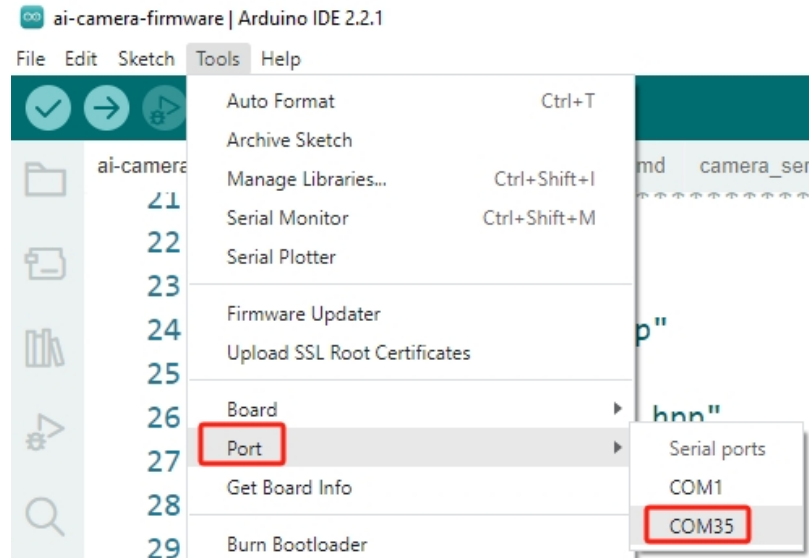
3. Open ai-camera-firmware.ino with the Arduino IDE, which also opens the associated code files.



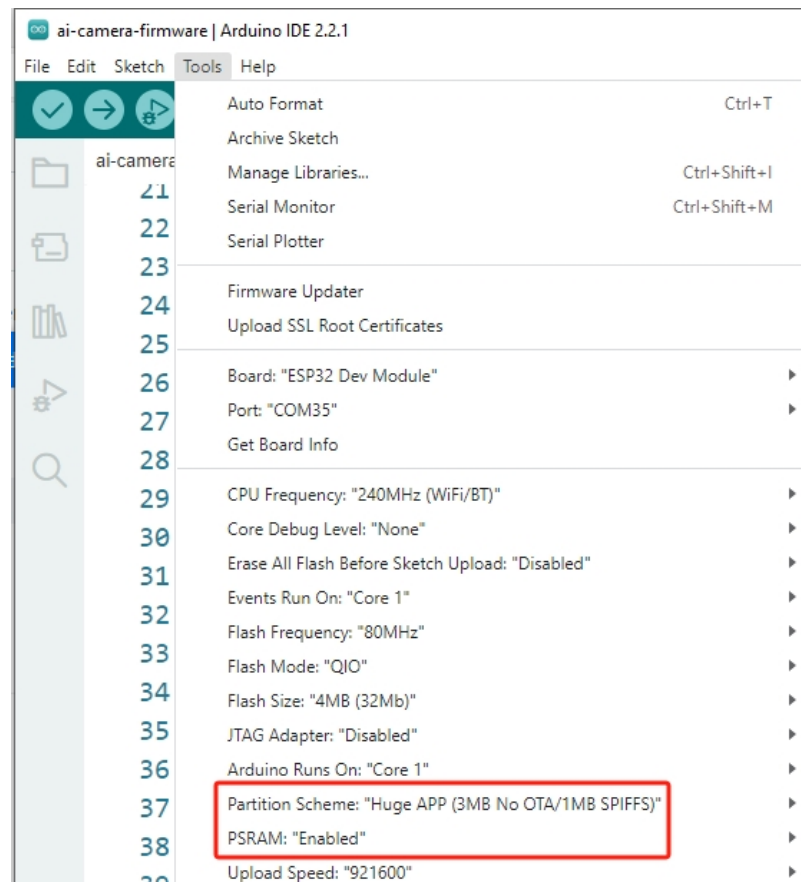
4. Select **Board** -> **esp32** -> **ESP32 Dev Module**.



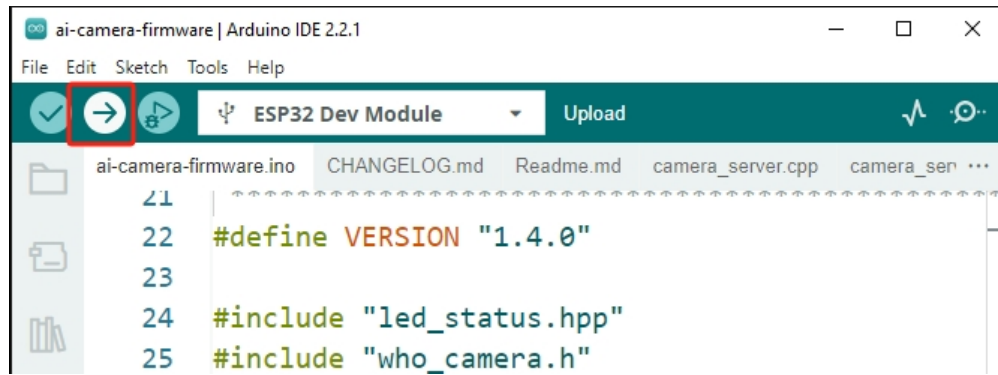
5. Choose the correct port.



6. Ensure to enable **PSRAM** and select **Huge APP** in the **Partition Scheme**.



7. Finally, upload the firmware to the ESP32 CAM.



8. After successful firmware upload, you can find more information at this link: <https://github.com/sunfounder/ai-camera-firmware>.

Copyright Notice

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.