THE SMART FACTORY
by Deloitte

Smart Factory Believers

EMPOWERING THE STEM WORKFORCE OF TOMORROW

FREQUENTLY ASKED QUESTIONS

FREQUENTLY ASKED QUESTIONS

THE SMART FACTORY by Deloitte

The purpose of this Frequently Asked Questions ('FAQ') guide is to provide comprehensive answers to the most frequently asked questions about the Smart Rover and curriculum resources. This document aims to assist Smart Rover users by addressing common inquiries, clarifying feature functionality, and providing detailed project code solutions for troubleshooting support. By consolidating these questions and answers, we strive to enhance understanding to support the successful integration of the Smart Rover. If you need additional assistance or have additions to this document, please contact our inbox: smartfactorybelievers@deloitte.com.

Table of Contents

GETTING STARTED

How do I get started?
Start with these foundational steps to familiarize yourself with the Smart Rover. Starting here will equip you with the confidence to successfully tackle all 12 Smart Rover projects.

1. **Visit the Smart Factory Believers website for program resources**: The program resources section includes links to the Smart Rover Project Manual in English and Spanish, project code solutions, and learning modules that include helpful guides to introductory concepts, as well as step-by-step video guides for each Smart Rover project.

2. **Gather hardware requirements and complete Kit Introduction and Setup module:** You will need a monitor, keyboard and mouse (or keyboard with trackpad) to connect to the Raspberry Pi; a laptop will not work. Follow the sequence of cables plugged in this Smart Module video to connect your Smart Rover to a monitor, keyboard/mouse, and power source. Make sure to plug into the power source last! Each Smart Rover requires 6 AA batteries to operate the onboard motor and camera, required for projects 4-12.

3. **Complete Introduction to Circuitry module and Introduction to Circuits projects:** The module introduces circuits, resistors, and capacitors in order to complete the two "Introduction to Circuits" projects in the Project Manual.

4. **Complete Smart Rover Project 1 & Smart Rover Project 2:** Project 1 introduces the basics of programming in Python and teaches students how to set up their program, create variables and loops to make the LED light blink. Project 2 expands upon Project 1 by adding to the previously built circuit and altering the Python code to add controls to the program.

The Smart Rover

How do I power the Smart Rover?
The Smart Rover requires two power sources to functionally operate.

- *Powering the Smart Rover body*: To power the Smart Rover body, you will need 6 AA batteries which are inserted into the bottom of the Smart Rover body. You may need a small screwdriver to access the battery port.
- *Powering the Smart Module*: Each kit also includes a USB cable and wall power adapter to connect your Raspberry Pi in the Smart Module to a power source.

What accessories are needed to use the Smart Rover?
To use and program the Smart Module, you will need a monitor, keyboard, and mouse (not included in the kit). Each kit also includes a USB cable to connect your monitor and a wall power adapter to connect your Raspberry Pi in the Smart Module to a power source. Below are specifications regarding the compatibility of the Smart Rovers (with embedded Raspberry Pi) and peripherals.

a. **Monitor**: a micro-HDMI connection for Smart Module, connecting to the appropriate connection for the monitor (standard HDMI or other). The Pi can handle resolutions up to 4k, so any monitor or TV with a resolution matching or lower should work.
b. **Keyboard and mouse**: standard-sized USB ports that can be used with a wired or wireless keyboard and mouse.
c. Note that the unit has an integrated Wi-Fi module with Bluetooth capabilities.
d. **Chromebooks and/or tablets are not compatible with the Smart Rover.**

My Smart Rover body does not turn on. What should I do?
Ensure you have placed 6 AA batteries in the bottom of the Smart Rover body.

My Smart Rover body turns on, but the wheels are not rotating. What should I do?
Your Smart Rover wheels may have debris stuck to them. Clean the wheels and ensure they are free of any foreign objects or debris.

How do I properly connect the jumper wires to the Smart Rover body?
The terminals of motor control and Smart Rover rear are color-coded to the jumper wires for ease of connection. Match the jumper wire snaps to the rear of the Smart Rover based on the color.

Why is the Smart Rover not responding to my code?
Ensure the circuit is properly built and all the components are fully connected by their snaps.

I'm having problems with debugging code.
We have documented known issues and their solutions on the Smart Factory Believers GitHub. You can access the latest code and documentation of known errors. These are updated regularly and can often be useful if you are experiencing technical issues. Reference the Project Code Solutions section below.

.

How do I access updated code for the Smart Rover projects?
You can access the latest code and documentation of known errors on the Smart Factory Believers GitHub.

How do I access the video guides for the Smart Rover projects?
You can access the video guides for each Smart Rover Project by selecting the corresponding project in the Project Guides section on the .

Why is my horn not responsive?
You have the option to use the horn component in projects 3 and 10. If it is not responsive, ensure that the horn is connected properly, with the "+" end oriented as shown in the Project Manual. If needed, refer to the troubleshooting guide within the Project Manual to test the horn.

Why is my LED light not responsive?

You have the option to use the LED component in projects 1-3 and 9-10. If it is not responsive, ensure that the LED light is connected properly and aligns with the direction of current flow as shown in the Project Manual. If needed, refer to the troubleshooting guide within the Project Manual to test the LED light.

Where are the images the camera captures stored?
You will use the camera in projects 8-9 and 11. The camera images are immediately discarded and are not stored anywhere in the Raspberry Pi within the Smart Module.

The Smart Module

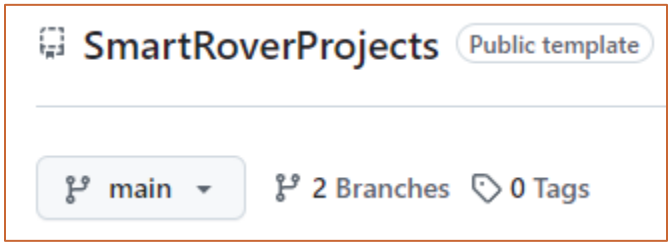
Can I connect my Smart Module to a laptop instead of using a monitor, keyboard, and mouse?
The Raspberry Pi in the Smart Module is a microcomputer. Therefore, the Smart Rover should only be used with a monitor, keyboard, and mouse, not a laptop or Chromebook. Although there are methods of connecting a Raspberry Pi to a laptop, they are not advised for using the Smart Rover kit.

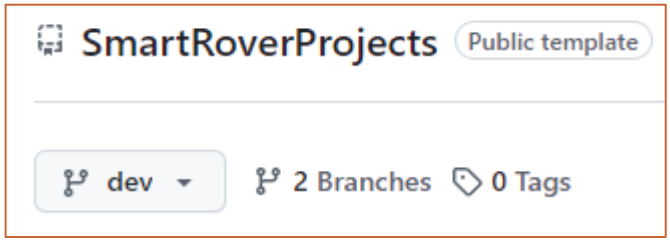
Can I power my Smart Module with another power source, or connect it to the batteries in the Smart Rover body?
The Raspberry Pi in the Smart Module should only be used with the power source provided. It cannot be powered by the batteries. There are alternative power and untethering options that we would be happy to support you with, please contact our inbox: smartfactorybelievers@deloitte.com.

Is it safe to connect the Smart Module to a power source overnight?
Do not keep the Raspberry Pi within the Smart Module connected to power overnight. Only connect it to power when it is in-use. We also recommend removing the 6 AA batteries from the Smart Rover body to avoid power drain.

Project Code Solutions

How do I access the project code or code solutions for the Smart Rover projects?
You can access the latest code, code solutions, and documentation of known errors on the Smart Factory Believers GitHub (www.github.com/SmartFactoryBelievers). There are two branches of code files in GitHub – the main and the dev branch. The main branch provides the code solutions, and the dev branch provides codes solutions with additional comments further explaining the lines of Python code. You can access the branches in the top left-hand corner. Photo provided below for reference.





There is also a Known Issues Log embedded in this document that includes known issues for each project and the code solutions to resolve known errors.

Project Manual

Does the Project Manual come in other languages?
The Project Manual is available in English and Spanish.

Can I work on a different project that is not in the Project Manual?
We encourage students to be creative and use the skills they've learned from the Project Manual to design their own projects. Remember to never build short circuits, as detailed on page 14 in the Project Manual. Short circuits will damage your components and quickly drain your batteries.

Curriculum Resources

What are the Smart Rover projects designed to do?
The Smart Rover projects begin with teaching the basics of computer programming on the Raspberry Pi microcomputer and take students all the way up to enabling the Smart Rover to self-drive and "see" using the embedded camera. The projects are connected to real-world technology, introducing users of the Smart Rover to Industry 4.0 skills and concepts.

How do I access the curriculum resources?
The Smart Rover includes 12 Smart Rover projects. We also developed six math and six science lesson sets that incorporate interactive, project-based learning in STEM. These resources are provided to our partner schools in collaboration with the National Math & Science Initiative. All partner schools have access to the Smart Factory Believers Learning Portal, which is the repository of the curriculum and coaching resources to support classroom instruction. If you are not affiliated with a partner school and are interested in learning more about the curriculum resources, please contact the Smart Factory Believers program team.

What are the lesson sets designed to do?
Lesson sets are designed to help teachers leverage the Smart Rover and introduce project-based learning in their general math and science courses as a discovery-based learning tool. The lessons are aligned to high school Common Core and Next Generation Science Standards and are adaptable for grades 9-12. The lessons are designed to use the Smart Rover functionality to support current math and science classroom content and relate core content to real-world scenarios.

How long is each lesson set?
Lessons are designed to be flexible. Each lesson will include a variety of activities and teachers will decide which activities to implement and how much time to give students. On average, a full lesson will involve 3-5, fifty-minute class periods.

What is the difference between a lesson and a Smart Rover project?
Smart Rover projects focus on computer science and circuit knowledge. They are intended to help students learn the foundational Python coding and programming skills so that they can then explore their own learning using the Raspberry Pi and Smart Rover.

The lesson sets are designed to help teachers leverage the Smart Rover projects in general math and science courses. These lessons are based on national math and science standards and include learning beyond computer science and circuits.

What training is available?
Virtual training opportunities are offered in the Fall, Winter, and Spring. Please contact the Smart Factory Believers program team if interested in attending a virtual training. Partner schools can access recordings of previous trainings in the Learning Portal in the Professional Learning category.

Contact Us

How do I get in touch with the Smart Factory Believers team?
You can email the team the Smart Factory Believers program team for any program related questions. For technical troubleshooting the Smart Rover, you can email the technical team at believerstechteam@deloitte.com.

Replacement Parts

What do I do if parts of my Smart Rover are missing?
If you are missing parts in your Smart Rover kit, please contact our Technical Team via email at believerstechteam@deloitte.com.

Where can I get replacement parts?
If you suspect your parts are broken, please refer to the troubleshooting guide in the Project Manual to test each component iteratively.

If you believe you have issues with your Smart Module, including the Raspberry Pi or the camera within the component, please contact our Technical Team via email at believerstechteam@deloitte.com.

I need support with Snap Circuits® components.
If you believe you have issues with your Snap Circuits® components, please contact Elenco Electronics® Customer Support via elenco.com or by emailing support@elenco.com.

KNOWN ISSUES LOG

Purpose
The purpose of this Known Issues Log is to provide Smart Rover users with a quick reference to common issues encountered while completing Smart Rover projects. The Known Issues Log offers code solutions to hopefully minimize disruptions when engaging with the Smart Rovers.

For any additional errors not listed, please contact the Smart Factory Believers Technical Team at believerstechteam@deloitte.com. We aim to address issues within 1-3 business days.

Index of Known Issues by Project:

- Projects 1-7
  - RuntimeWarning: This channel is already in use Error
- Projects 8-12
  - Resources Error
  - Picamera Error

In *Projects 1 – 7,* you may encounter a **"RuntimeWarning: This channel is already in use Error".** If you receive this error, please use the code solutions below that are specific to the projects you are working on. Each project has a different code solution.

In *Projects 8-12,* you may encounter two errors that are related to the camera. The two known errors are:
1) **"Resources Error",** and
2) **"picamera error"**

Both errors are related to each other. If you receive either of these errors, there are two methods to resolve the errors. Both methods can be used for projects 8-12.

Project 1:

To solve a, **"RuntimeWarning: This channel is already in use Error"**, you must run a cleanup function on the GPIO pins at the end of your code, GPIO.cleanup() and run a setwarnings(False) line of code at the beginning of the program.

In the end your code should match this GitHub code file:

https://github.com/SmartFactoryBelievers/SmartRoverProjects_Solutions/blob/main/Project01_BlinkingLED_Solution.py

We've included the code below with the necessary changes highlighted:

```
# Project 1
```

```
# Learning to program and use outputs

# Build the the Project 1 circuit and blink a LED

#Challenge 1
# Try changing the LED_On and LED_Off variables to change the blinking pattern

#Challenge 2
# Replace the color LED with buzzer or white LED to try other outputs

#Importing libraries
# Libraries are defined sets of code for specific uses
# Here we want the sleep function for timing and GPIO for the Pi's pin
from time import sleep
import RPi.GPIO as GPIO

GPIO.setwarnings(False)

#Let's define variables so we can use them later
# Variables are words that take on values within the code
# This way, we can edit the value at the beginning and the changes flow through
LED_Pin =  40 #the internal Pi pin number that goes to snap 7

# For challenge 1, we can try different values here to blink in new patterns
LED_On = 3 #duration of LED flash, seconds
LED_Off = 1 #duration in between flashes, seconds

#Setting up our pin
GPIO.setmode(GPIO.BOARD)
GPIO.setup(LED_Pin, GPIO.OUT, initial=GPIO.LOW)  #Output pin, start off

while True: #Looping over and over again
    sleep(LED_Off) #Keep LED off for defined duration
    GPIO.output(LED_Pin, GPIO.HIGH) #Turn LED on
    sleep(LED_On) #Keep LED on for defined duration
    GPIO.output(LED_Pin, GPIO.LOW) #Turn lED off

print(list(f(10)))

GPIO.cleanup()
```

Project 2:

To solve a **"RuntimeWarning: This channel is already in use Error"**, you must run a cleanup function on the GPIO pins at the end of your code, GPIO.cleanup(), and run a setwarnings(False) line of code at the beginning of the program.

In the end your code should match this GitHub code file:

https://github.com/SmartFactoryBelievers/SmartRoverProjects_Solutions/blob/main/Project02_LightActivatedLED_Solution.py

We've included the code below with the necessary changes highlighted:

```
# Project 2

# Learning to program and using inputs and outputs

# Build the the Project 2 circuit and control a LED with a button

#Challenge 1
# Try changing the LED_On and LED_Off variables to change the blinking pattern

#Challenge 2
# Replace the color LED with buzzer or white LED to try other outputs

#Challege 3
# Replace the push button with the phototransistor and cover it with your hand - wha
t happens?

#Challege 4
# Try changing the "If" statement from True to False - now what does the button do?

#Importing libraries
# Here we want the sleep function for timing and GPIO for the Pi's pins
from time import sleep
import RPi.GPIO as GPIO

GPIO.setwarnings(False)

#Let's define variables so we can use them later
Button_Pin =  38 #the internal Pi pin number that goes to snap 6
LED_Pin = 12 #the internal Pi pin number that goes to snap 3

# For challenge 1, we can try different values here to blink in new patterns
LED_On = 3 #duration of LED flash, seconds
```

```
LED_Off = 1 #duration in between flashes, seconds

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
GPIO.setup(LED_Pin, GPIO.OUT, initial=GPIO.LOW)  #Output pin, start off
GPIO.setup(Button_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #Input pin, start open

while True: #Looping over and over again

    # Here we use the If statement which evaluates a logical expression
    # It is checking if the button is pressed by reading tha value of the pin
    # If the button pin reads True (on), then it executes the indented code

    if GPIO.input(Button_Pin) == False: #When the button is pressed, blink LED
        sleep(LED_Off) #Keep LED off for defined duration
        GPIO.output(LED_Pin, GPIO.HIGH) #Turn LED on
        sleep(LED_On) #Keep LED on for defined duration
        GPIO.output(LED_Pin, GPIO.LOW) #Turn lED off

    # If the button is not pressed, the code will go to the else statement
    else:
        print('Button not pressed')
        sleep(1)
GPIO.cleanup()
```

Project 3:

To solve a **"RuntimeWarning: This channel is already in use Error"**, you must run a cleanup function on the GPIO pins at the end of your code, GPIO.cleanup(), and run a setwarnings(False) line of code at the beginning of the program.

In the end your code should match this GitHub code file:

https://github.com/SmartFactoryBelievers/SmartRoverProjects_Solutions/blob/main/Project03_SelectorOutput_Solution.py

We've included the code below with the necessary changes highlighted:

```
# Project 3

# Learning to program, writing functions, and using inputs and outputs

# Build the the Project 3 circuit and control a LED and buzzer with a selector
# Press and hold buttons A, B, and C on the selector

#Challenge 1
```

```
# Try changing the Pin_On and Pin_Off variables to change the blinking pattern

#Challenge 2
# Replace the color LED with buzzer or white LED to try other outputs

#Challenge 3
# Try changing input pins A and C in the While loop to switch what A and C do when p
ressed

#Challenge 4
# Try changing output pins LED and Buzzer in the While loop to switch what A and C d
o when pressed

#Challenge 5
# Try switching the order of the LED and Buzzer functions for a cool lightshow when
pressing B

#Importing libraries
# Here we want the sleep function for timing and GPIO for the Pi's pins
from time import sleep
import RPi.GPIO as GPIO

GPIO.setwarnings(False)

#Let's define variables so we can use them later
A_Pin =  40 #the internal Pi pin number that goes to snap 7
C_Pin =  38 #the internal Pi pin number that goes to snap 6
LED_Pin = 12 #the internal Pi pin number that goes to snap 3
Buzzer_Pin = 35 #the internal Pi pin number that goes to snap 4

# For challenge 1, we can try different values here to blink in new patterns
Pin_On = 3 #duration of LED flash, seconds
Pin_Off = 0.5 #duration in between flashes, seconds

#Setting up our pins
GPIO.setmode(GPIO.BOARD)

#Our output pins, start off
GPIO.setup(LED_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Buzzer_Pin, GPIO.OUT, initial=GPIO.LOW)

#Our input pins from the selector
GPIO.setup(A_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(C_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

#Let's write some functions we can use to make the coding easier
# For a code snippet we will reuse, we can turn it into a function to call later
# The function name is in blue, and then the arguments it takes are in parentheses
```

```
#Here's a function for seeing if a selector button is pressed
# So, read_selector_button reads and returns the value of In_Pin
# This will be helpful for reading the A and C button pins
def read_selector_button(In_Pin):
    return GPIO.input(In_Pin)

#Here's a function for turning an output pin on
#So, output_pin_on takes in the pin number and turns it on after a defined delay
def output_pin_on(Out_Pin, Delay):
    sleep(Delay)
    GPIO.output(Out_Pin, GPIO.HIGH)

#Here's a function for turning an output pin off, can you fill in the missing pieces
?
# Replace the ?? with the variables and then uncomment
def output_pin_off(Out_Pin, Delay):
    sleep(Delay) #wait the Delay
    GPIO.output(Out_Pin, GPIO.LOW) #turn the Out_Pin off

while True: #Looping over and over again

    # Here we can use the functions we defined to read buttons and control outputs
    # For the challenges, try changing the button and output pins in the below code

    # If A is pressed and C is not, let's blink the LED
    if read_selector_button(A_Pin) and not(read_selector_button(C_Pin)):
        output_pin_on(LED_Pin, Pin_Off)
        output_pin_off(LED_Pin, Pin_On)

    # If C is pressed and A is not, let's buzz the buzzer
    if read_selector_button(C_Pin) and not(read_selector_button(A_Pin)):
        output_pin_on(Buzzer_Pin, Pin_Off)
        output_pin_off(Buzzer_Pin, Pin_On)

    # If A and C are both pressed, by pressing B, maybe we can flash both LED and bu
zzer?
    # Replace the ?? with the LED_Pin and Buzzer_Pin variables and then uncomment
    if read_selector_button(A_Pin) and read_selector_button(C_Pin):
        output_pin_on(LED_Pin, Pin_Off)
        output_pin_off(Buzzer_Pin, Pin_On)
        output_pin_on(LED_Pin, Pin_Off)
        output_pin_off(Buzzer_Pin, Pin_On)

    # Wait 1 second to reset
    sleep(1)
GPIO.cleanup()
```

Project 4:

To solve a **"RuntimeWarning: This channel is already in use Error"**, you must run a cleanup function on the GPIO pins at the end of your code, GPIO.cleanup(), and run a setwarnings(False) line of code at the beginning of the program.

In the end your code should match this GitHub code file:

https://github.com/SmartFactoryBelievers/SmartRoverProjects_Solutions/blob/main/Project04_ProgrammedDriving_Solution.py

We've included the code below with the necessary changes highlighted:

```
# Project 4

# Learning to program, writing functions, and using motor control outputs

# Build the the Project 4 circuit and drive the rover along your designed path

#Challenge 1
# Try changing the drive time variable to create a new driving path

#Challenge 2
# Reorder the drive functions to create a new driving path

#Challege 3
# Create your own custom driving path using the different drive functions and time a
rguments

#Importing libraries
# Here we want the sleep function for timing and GPIO for the Pi's pins
from time import sleep
import RPi.GPIO as GPIO

GPIO.setwarnings(False)

#Let's define variables so we can use them later
Left_Forward_Pin =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4

#Here we can define the timing variables for the driving functions, in seconds
# For challenge 1, we can try different values here to drive in new patterns
Forward_Time = 2
Backward_Time = 1
```

```
Left_Turn_Time = 0.5
Right_Turn_Time = 0.5
Wait_Time = 1

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)


#Let's write some driving functions we can use later to program a driving path
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('forward')
    sleep(1)

def drive_left_turn(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('left turn')
    sleep(1)

def drive_right_turn(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('right turn')
    sleep(1)

# Can you finish the function by filling in the blanks for the pins and states?
# This is a backward driving function, so both backward pins should be High then Low
# Uncomment the code when complete
def drive_backward(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
```

```
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('backward')
    sleep(1)

#Here we can use a for loop to control the number of times the code is executed
# Changing the value of range() increases the number of loops performed
for n in range(1):

    # Let's use the driving functions defined above to create a driving path
    # For challenges 2 and 3, try changing the driving functions and order here
    sleep(Wait_Time)
    drive_forward(Forward_Time)
    drive_left_turn(Left_Turn_Time)
    drive_backward(Backward_Time)
    drive_right_turn(Right_Turn_Time)
GPIO.cleanup()
```

Project 5:

To solve a **"RuntimeWarning: This channel is already in use Error"**, you must run a cleanup function on the GPIO pins at the end of your code, GPIO.cleanup(), and run a setwarnings(False) line of code at the beginning of the program. For this project you will also have to define which GPIO pin relates to which motor at the beginning of the while loop.

In the end your code should match this GitHub code file:

https://github.com/SmartFactoryBelievers/SmartRoverProjects_Solutions/blob/main/Project05_TimedDriving_Solution.py

We've included the code below with the necessary changes highlighted:

```
# Project 5

# Learning to program, writing functions, using motor control outputs, adding callback function

# Build the the Project 5 circuit and drive the rover through button pushes, Simon Says style
# Press and hold the button to drive for that duration

#Challenge 1
# Try changing the drive functions to switch the driving directions

#Challenge 2
```

```
# Add new drive functions to activate in the loop to create a new path

#Challege 3
# Use the modulo opertor to change the driving directions based on even or odd numbe
red presses

#Challege 4
# With the modulo, add new dirivng functions for even or odd numbered presses

#Importing libraries
# Here we want the sleep function for timing and GPIO for the Pi's pins
from time import sleep
import RPi.GPIO as GPIO
# We also now are using the general time library for the timer function
import time

GPIO.setwarnings(False)

#Let's define variables so we can use them later
Left_Forward_Pin =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4
Button_Pin =  38 #the internal Pi pin number that goes to snap 6

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
#Our input pin from the button
GPIO.setup(Button_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)


#Let's write some driving functions we can use later to program a pathdef drive_forw
ard():
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('forward')
    sleep(1)

def drive_left_turn(time):
```

```
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('left turn')
    sleep(1)

def drive_right_turn(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('right turn')
    sleep(1)

def drive_backward(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('backward')
    sleep(1)

# Here we are creating a timer function to record the duration of the button press
def button_press_timer():
    Start_Time = time.time() #start the timer
    while GPIO.input(Button_Pin): #while the button is pressed...
        print("Button Pressed")
    return round(time.time() - Start_Time,2) #stop the timer, return elapsed time

# For challenges 3 and 4, we will use a dummy variable to help with modulo operator
count = 0
# Replace the True with the modulo operator statement as %, which means remainder in
 division
# So modulo 2 keeps track of odd and even presses since even divided by 2 has remain
der of 0
# To use this as a logical, let's try count % 2 == 0

while True: #Looping over and over again
    sleep(0.25)
    GPIO.setmode(GPIO.BOARD)
    #Our output pins, start off
    GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
```

```
    GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
    #Our input pin from the button
    GPIO.setup(Button_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    # If the button is pressed, let's use the timer function to see how long
    if GPIO.input(Button_Pin):
        Button_Time = button_press_timer()
        print('Button pressed ' + str(Button_Time) + ' seconds')

        if count % 2 == 0: # Try changing the True to the modulo for challenges 3 an
d 4
            #For challenges 1 and 2, try adding new driving functions here
            drive_forward(Button_Time)

        else: # To be used in  challenges 3 and 4
            drive_backward(Button_Time)
            # Add other drive functions here for odd button presses

        count = count + 1 # We increment the counter for the next button press
GPIO.cleanup()
```

Project 6:

To solve a **"RuntimeWarning: This channel is already in use Error"**, you must run a cleanup function on the GPIO pins at the end of your code, GPIO.cleanup(), and run a setwarnings(False) line of code at the beginning of the program. For this project you will also have to define which GPIO pin relates to which motor at the beginning of the while loop.

In the end your code should match this GitHub code file:

https://github.com/SmartFactoryBelievers/SmartRoverProjects_Solutions/blob/main/Project06_LightActivatedDriving_Solution.py

We've included the code below with the necessary changes highlighted:

```
# Project 6

# Learning to program, writing functions, using motor control outputs, adding loop c
omplexity

# Build the the Project 6 circuit and have the rover be controlled by ambient light

# Turn down the ligth and point a flashlight at the rover to direct it

#Challenge 1
# Try changing the drive functions to switch the driving directions
```

```
#Challenge 2
# Add new drive functions to change its light seeking spin pattern

#Challege 3
# Add the 100 Ohm resistor in series with the photoresistor to increase light sensit
ivity

#Challege 4
# With the modulo operator, have the rover alternate left or right spins in light se
arching

#Challenge 5
# After a certain amount of time, have the rover spin to look for light

#Importing libraries
# Here we want the time and sleep for timing and GPIO for the Pi's pins
import time
from time import sleep
import RPi.GPIO as GPIO

GPIO.setwarnings(False)

#Let's define variables so we can use them later
Left_Forward_Pin =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4
Photo_Pin =  38 #the internal Pi pin number that goes to snap 6

#Here we can define the timing variables for the driving functions, in seconds
Forward_Time = 2
Backward_Time = 1
Left_Turn_Time = 0.5
Right_Turn_Time = 0.5
Wait_Time = 1

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
#Our input pin from the button
GPIO.setup(Photo_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

```
#Let's write some driving functions we can use later
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('forward')
    sleep(1)

def drive_left_turn(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('left turn')
    sleep(1)

def drive_right_turn(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('right turn')
    sleep(1)

def drive_backward(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('backward')
    sleep(1)

# For challenge 4, we will use a dummy variable to help with modulo operator
count = 0
# Replace the True with the modulo operator statement as %, which means remainder in
 division
# So modulo 2 keeps track of odd and even presses since even divided by 2 has remain
der of 0
# To use this as a logical, let's try count % 2 == 0

# For challenge 5, we will set a maximum light search time for the loop
Max_Search_Time = 4 #seconds
```

```
# If the rover has not found light by then, we can get out of the loop with a break
statement
# break exits the innermost loop and allows the rover to return to the first sleep c
ommand


while True: # Continuous outer while loop

    GPIO.setmode(GPIO.BOARD)
    #Our output pins, start off
    GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
    #Our input pin from the button
    GPIO.setup(Photo_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    sleep(0.25)
    count = count + 1 # Increment the counter for the modulo

    # If the phototransistor detects enough light, drive towards it
    if GPIO.input(Photo_Pin):
        # For challenges 1 and 2, change driving instructions here
        drive_forward(Forward_Time)

    # If there's not enough light, let's look for it by spinning the rover
    else:
        # For challenge 5, we can use the timer function to control the light search
        Start_Time = time.time()
        while not(GPIO.input(Photo_Pin)):
            Elapsed_Time = round(time.time() - Start_Time,2)
            print('Not enough light, searching for more')

            if Elapsed_Time < Max_Search_Time: # Try changing the True to a comparat
ive (<) between
                    # Elapsed_Time and Max_Search_Time for challenge 5

                if count % 2 == 0: # Try changing the True to the modulo for challen
ge 4
                    drive_left_turn(Left_Turn_Time)
                    sleep(Wait_Time)

                else: # For challenge 4, modulo uses these drive commands on odd loo
ps
                    drive_right_turn(Right_Turn_Time)
                    sleep(Wait_Time)
            else:
                GPIO.cleanup()
```

```
        break # Exits the loop after Max Search Time exceeded
```

Project 7:

To solve a **"RuntimeWarning: This channel is already in use Error"**, you must run a cleanup function on the GPIO pins at the end of your code, GPIO.cleanup(), and run a setwarnings(False) line of code at the beginning of the program.

In the end your code should match this GitHub code file:

https://github.com/SmartFactoryBelievers/SmartRoverProjects_Solutions/blob/main/Project07_ControlledDriving_Solution.py

We've included the code below with the necessary changes highlighted:

```
# Project 7

# Learning to program, writing functions, using motor control outputs, adding
complex logic

# Build the the Project 7 circuit and drive the rover with button presses A, B, and
C

# Set the controls for the rover for 3 unique commands, and possibly more?

#Challenge 1
# Try changing the drive functions to switch the driving directions for
forward/backwards and turning

#Challenge 2
# Add new drive functions to change the driving patterns for each button press

#Challege 3
# Incorporate the button press timer from project 5 to add Simon Says to driving
functions

#Challege 4
# See how B uses a double If to see if its pressed and then released or held? Can
you try
# something similar for A and C to create different commands there too?

#Challenge 5
# Replace the length-3 snap connector with the phototransistor - now all three
buttons
# are light dependant. Try controlling the rover to stay in the light.

#Importing libraries
# Here we want the time and sleep for timing and GPIO for the Pi's pins
import time
from time import sleep
import RPi.GPIO as GPIO
```

```
GPIO.setwarnings(False)

#Let's define variables so we can use them later
Left_Forward_Pin = 35 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin = 31 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin = 26 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin = 21 #the internal Pi pin number that goes to snap 4
A_Pin = 7 #the internal Pi pin number that goes to snap 7
C_Pin = 18 #the internal Pi pin number that goes to snap 6

#Here we can define the timing variables for the driving functions, in seconds
Forward_Time = 2
Backward_Time = 1
Left_Turn_Time = 0.5
Right_Turn_Time = 0.5
Wait_Time = 0.5

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
#Our input pin from the button
GPIO.setup(A_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(C_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

#Let's write some driving functions we can use later to program a pathdef
drive_forward():
def drive_forward(time):
  GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor fwd
  GPIO.output(Right_Forward_Pin, GPIO.HIGH) #R motor fwd
  sleep(time)
  GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor fwd
  GPIO.output(Right_Forward_Pin, GPIO.LOW) #R motor fwd
  print('fwd')
  sleep(1)

def drive_backward(time):
  GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor bkwd
  GPIO.output(Right_Backward_Pin, GPIO.HIGH) #R motor bkwd
  sleep(time)
  GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor bkwd
  GPIO.output(Right_Backward_Pin, GPIO.LOW) #R motor bkwd
  print('bkwd')
  sleep(1)

def drive_left_turn(time):
  GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor bkwd
  GPIO.output(Right_Forward_Pin, GPIO.HIGH) #R motor fwd
  sleep(time)
  GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor bkwd
  GPIO.output(Right_Forward_Pin, GPIO.LOW) #R motor fwd
  print('left turn')
  sleep(1)
```

```
def drive_right_turn(time):
  GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor bkwd
  GPIO.output(Right_Backward_Pin, GPIO.HIGH) #R motor fwd
  sleep(time)
  GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor bkwd
  GPIO.output(Right_Backward_Pin, GPIO.LOW) #R motor fwd
  print('right turn')
  sleep(1)

# Here we are creating a timer function to record the duration of the button press
def button_press_timer():
  Start_Time = time.time() #start the timer
  while GPIO.input(Button_Pin): #while the button is pressed...
    print("Button Pressed")
  return round(time.time() - Start_Time,2) #stop the timer, return elapsed time
# For challenge 3, try uncommenting the Press_Time statements, then use it for the
# the drive commands time arguments

while True: #Looping over and over again
  sleep(0.5)

  # Only pressing A
  if GPIO.input(A_Pin) and not GPIO.input(C_Pin): #only pressing A
    # For challenge 4, you can use a sleep delay and second if, else to see
    # whether A was pressed and released or held
    sleep(0.5)
    #Press B and hold, check if still pressed after delay
    if GPIO.input(A_Pin) and not GPIO.input(A_Pin):
      drive_forward(Forward_Time)
      drive_backward(Backward_Time)
    else:
      Press_Time = button_press_timer(A_Pin) # For challenge 3
      drive_forward(Forward_Time)

  # Only pressing C
  if GPIO.input(C_Pin) and not GPIO.input(A_Pin): #only pressing C
    # For challenge 4, you can use a sleep delay and second if, else to see
    # whether C was pressed and released or held
    Press_Time = button_press_timer(C_Pin) # For challenge 3
    drive_backward(Backward_Time)

  # Pressing B, we can use timing to determine if it's released or held
  if GPIO.input(C_Pin) and GPIO.input(A_Pin):
    sleep(0.5)
    #Press B and hold, check if still pressed after delay
    if GPIO.input(C_Pin) and GPIO.input(A_Pin):
      drive_left_turn(Left_Turn_Time)
    # Press B and released, not still pressed after delay
    else:
      drive_right_turn(Right_Turn_Time)
GPIO.cleanup()
```

Projects 8-12:

"Resources Error"

If you are getting a **"Resources Error"** when running any camera project in projects 8-12, it is possible that you have a file enabled on your Raspberry Pi that should be disabled.

To fix this you can either open the terminal and run the following commands:

"sudo systemctl disable ft-test3.service"

"sudo reboot"

Or you can run the following code file in the Thonny code editor on the Raspberry Pi:

https://github.com/SmartFactoryBelievers/SmartRoverProjects_Solutions/blob/main/camFix.py

If the above commands and/or .py file doesn't serve as a solution, continue troubleshooting by following the instructions to resolve a "picamera error".

"picamera error"

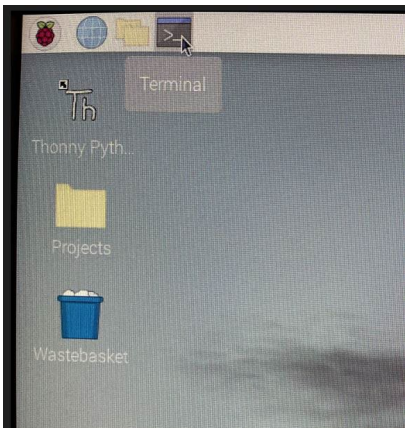
If you are getting a **"picamera error"** when running any camera project in projects 8-12, it is possible that you must enable the camera included in your Smart Module. To do this, you will need:

- Smart Module (house Raspberry Pi and Camera)
- HDMI to Micro HDMI cord
- USB-C power supply
- Keyboard
- Mouse
- Monitor

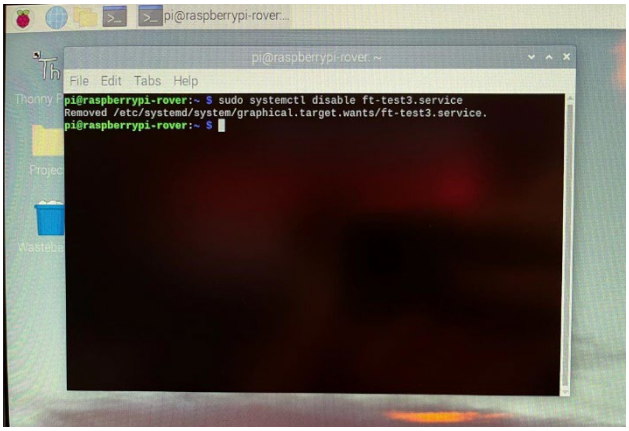
There is an automated solution (preferred method) and manual steps to enable the camera. Each option is detailed below.

Automated Solution:
https://github.com/SmartFactoryBelievers/SmartRoverProjects_Solutions/blob/main/AutomatedCameraRepair.pdf

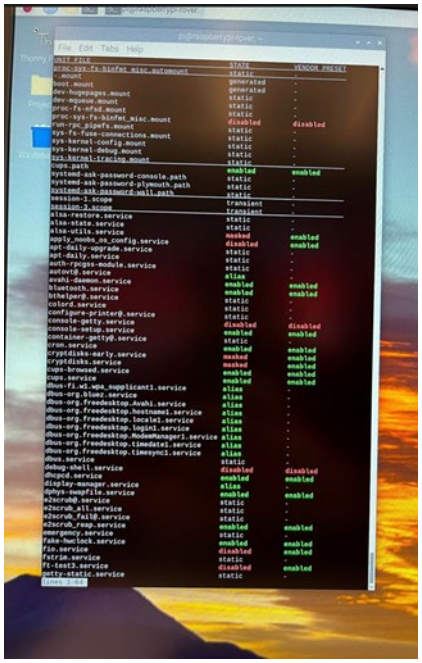
Manual Steps:



1. Connect all cords to appropriate ports.
    a. If the screen does not turn on, unplug the power, and plug it back in.
    b. Ensure all other cords are plugged in prior to powering on the unit.
2. In the top left-hand corner of the desktop is a Terminal.
    a. When you open the terminal, it should open a window on the desktop.
3. Type "**sudo systemctl disable ft-test3.service**".
    a. DO include spaces in the commands.
    b. DO NOT include the quotation marks when entering commands.
    c. After you input the command, you should see a message confirming the script was removed.



4. To check if the command has been input successfully, input the following command into the terminal: "**sudo systemctl list-unit-files**" WITH spaces and WITHOUT quotation marks.
    a. The list of scripts should appear.



5. Locate "**ft-test3.service**". In the "STATUS" column, the script should read "**disabled**".
    a. If the script is not listed as disabled, repeat the steps.
    b. Once you have verified the program is disabled, shut down the Raspberry Pi by entering the following command into the terminal: "sudo poweroff -f."