Longlevens Library Code Club

Python – STS-PI Rover Controller

Description

This project enables you to create a controller for one of the club's STS-PI Rovers. A program has been written for the rover to allow you to interact with it. The following commands are available:

"**Move**" The rover responds with: "Left wheel speed?", "Right wheel speed?" and "Duration?".

"**Stop**" This immediately halts the motion of the rover.

"**Photo**" This takes a photo at the time of issuing the command.

"**Video**" This leads to a "Duration?" response after which video is taken for that amount of time.

Photos and videos are saved to the Rover folder on the rover's desktop for viewing later with a screen connected to the rover. An advanced exercise will be created that enables you to view the camera from the remote computer you are using to control the rover.

The rover has eight numbered buttons and four LED lights on the top. The lights are used to indicate the following:

Blue: Flashing, awaiting connection to a controller, fully on, connected.

Orange: On indicates an exchange of data is taking place with the controller.

Red: Flashing indicates controlled movement is taking place.

Green: The camera is currently in use.

Four of the buttons are programmed to carry out the following functions:

1. Disconnect from the current connected controller.

3. Emergency stop.

7. Reboot

8. Shutdown

You must press button 8 and wait for the green light by the red "power on" light facing forwards underneath to stop for several seconds before unplugging the power.

While you testing your code, you may find that the rover stops responding to connection and motion commands. Try pressing button 1 and reconnect, but if no improvement, press button 7 to reboot and restart your code once the blue light starts flashing again.

CAUTION

The rovers are delicate and expensive to repair or replace. Please make sure you do not drive the rover into the path of other library users who may not be expecting robots running around where they are walking. Do not under any circumstances drive the rovers close to the edge of tables or other places where they may fall and become damaged.

Load and Save the Starter Code

Open a Python editor of your choice and load in the “STS-PI/STS-PI-Rover-Controller.py” file from the “Exercises/Python/Intermediate/STS-PI” folder. Then save the project under a new name in your own “Coders” folder.

Windows in Python

We are going to create a controller using a graphical user interface. This will be created using a Python package called “GUI Zero” which enables windows to be created onto which we can easily add buttons, slider controls and other common windows elements know as “widgets”. Once you have the functions required, you can design the your own look and appearance of the controller.

In the starter code you will see a function definition called “setup_gui”. To create a window you need to declare an “App”.

```
global app
app = App("STS Pi Controller", layout="grid")
```

At the bottom of the code, make a call to the function with the command (no indentation):

```
setup_gui()
```

Run the code and you should see a blank window appear on the screen. Press the cross in the top right hand corner to close the window. A message will have appeared on the console stating “Connection made = False”. At the bottom of the starter code there is some code which starts up a receiver for the messages from the rover, but as there is no connections, there is nothing for the controller to receive. Do not worry about that at this stage. First we need to add in a button to make a connection to a rover. In the “setup_gui” definition, add the following statement:

```
rover1Button = PushButton(app,
                          command=connect,
                          args=[1],
                          text="Connect to STS-Pi 1",
                          grid=[1,0])
```

Copy the statement and replace 1 with 2 so you can also use Rover number 2. Increment the grid position to [2, 0] so that the button is placed in column 2 of the first row after the first button.

Save and run your code. You should now see two buttons at the top of the window. Close the window by clicking on the “X” in the top right hand corner of the window. We next need to add in a “Disconnection” button. Add the following under the declaration of the two previous buttons:

```
disconnectButton = PushButton(app, command=disconnect, text="Disconnect", grid=[3, 0])
```

Connecting to the Rover

The controller will communicate with the rover via Bluetooth. When the rover starts up, it will automatically run the code needed to talk to your controller. A blue light on the rover will flash when it is ready to accept a Bluetooth connection.

The connect and disconnect function definitions have already been declared for you in the “start-up” code. Run your code and click the connection button for the rover you are using. The console should report that you have connected and state it is listening for a response from the rover. Click on the “Disconnect” button and the console should show a series of lines that show the disconnection taking place. Ignore any error messages that look like “Unable to close socket”. Click the connection button again and check a second connection has been made, then disconnect again. Close the window with the “X” in the top right corner.

Before we continue, we want to make sure we disconnect before closing our program to let the rover know we have stopped using it so that it is ready for new connections in the future. Add the following line into the “setup_gui” function:

```
app.when_closed = closeDown
```

Making the Rover Move

Now we can connect to the rover, we want to be able to send it commands to control it. The STS-Pi rovers have two powered wheels at the rear and an ball bearing at the front which simply rolls along. Each of the rear wheels has an independently controlled motor so that the rover can move not only forward or backwards, but turn or spin left or right by giving each wheel different speeds. If the right wheel moves forwards but the left wheel stays still, the rover will turn to the left pivoting on the left wheel. If the left wheel is moved backwards at the same speed as the right hand wheel turns forwards them the rover will around anti-clockwise.

In the “setup_gui” function definition, add “Slider” widgets to set the speed of each wheel. Here’s the code for the left wheel:

```
leftSliderText = Text(app, "Left Wheel Speed %", grid=[1, 2])
leftSlider = Slider(app, command=changeLeftSpeed, start=-100, end= 100, grid=[2, 2])
```

Repeat the above for the right hand wheel. Note the “grid” values set the position of each widget in the window. You will need to increment the row or column number so that the right hand wheel does not overlay on top of the left wheel widgets. (The first value is the column and the second is the row.)

We need to retrieve the value of the sliders whenever the rover asks for their value. The function named by the “command” parameter is called every time the value is changed using the mouse. The “start-up” code already has function definitions declared for “changeLeftSpeed” and “changeRightSpeed”. Scroll down in the code until you see “def changeLeftSpeed(slider_value)” and insert the following under the description for the function.

```
global speedLeft
speedLeft = int(slider_value)
print("New left wheel speed=", speedLeft)
```

Repeat this code for the changeRightSpeed function but modify it for the right wheel speed.

Run you code and adjust the sliders. You should be able to change the speed of each wheel from “-100” to “100”. Each is a percentage of maximum speed the rover can apply to each wheel.

Even if you press connect and establish a connection with a rover, simply adjusting the sliders will have no affect. We need to be able to tell the rover we want it to move and for how long. To adjust the duration, add another “Slider” widget.

```
durationText = Text(app, "Duration (secs)", grid=[1, 4])
durationSlider = Slider(app, command=changeDuration, start=1, end= 5, grid=[2, 4])
```

Create a new function so that a variable called “duration” is changed each time the slider is adjusted.

Create the following two buttons to start and stop the rover:

```
moveButton = PushButton(app, move, text="Move", grid=[1, 5])
stopButton = PushButton(app, stop, text="Stop", grid=[2, 5])
```

Before moving the rover, we need to be able to stop it in case it goes too far or the wrong way. Scroll down to the placeholder function definition “stop”. Add the following to the function:

```
sendCommand("Stop")
```

The “sendCommand” function uses the Bluetooth connection to send the request to the rover.

Now we can try and get the rover moving. Find the placeholder function definition “move”. Add the following two lines of code:

```
    print("Move for ",
          duration,
          "seconds at speed of {}% left and {}% right".format(leftSpeed, rightSpeed))
    sendCommand("Move")
```

That doesn’t tell the rover what speed to set for the left and right wheels. The rover enquires for these values once it has received the “Move” command. Find the “receiveResponse” function definition and then find the “if” or “elif” statements that check for “Wheel Speed Left?” and “Wheel Speed Right?” questions from the rover. Add if the following line of code for when the left wheel speed is requested:

```
    sendInteger(speedLeft)
```

Add in the same statement but for the right hand wheel. Underneath is the “elif” statement for “Duration?”. Again add in the appropriate call to send the rover the duration for the move.

If everything is correct, you should now be able to move the rover. Restart your code and connect to the rover you are using. Once the message “Listening for response from the rover” appears, leave the sliders at zero for the wheel speeds and click move. The console should show the exchange of commands, values and status messages. While the values are being exchanged the orange light on the rover will be lit, while it is carrying out the move the red light will flash. If all that happened and the last message on the console is “Listening for response from the rover”, adjust the sliders and see if you can get the rover to move. Increase the duration slider to 5, start the rover moving and as soon as it starts click the stop button. Make sure the rover stops immediately.

Why do you think it would be a bad idea to set a long duration for the move command?

What happens if you request another move while the rover is already moving?

Adding Controls to the Controller

When a connection is made, the button that was pressed to make that connection can be pressed again which confuses the rover. Pushbutton widgets can be disabled to stop multiple presses causing problems. You may notice that while the connection is being established all the buttons and sliders do not respond to clicks. We need to disable the connection buttons when the connection is successfully made but before the buttons become usable again. In the “connect” function definition, find the line of the code that states “connected = True” and put the following lines of code under it:

```
    rover1Button.disable()
    rover2Button.disable()
```

Repeat these two lines in the “disconnect” function but to enable them after the “Bye” command is sent.

Python requires widgets created in the “setup_gui” function to be declared as “global variables” so they can be changed in other functions. In the “setup_gui” function, change the line “global app” to:

```
    global app, rover1Button, rover2Button
```

Now restart your code and see if the buttons are disabled when the rover is connected and become usable again after disconnecting.

Can you disable and enable the “Move” button so that it can’t make another request while the rover is moving?

Additional Challenges

A. Redesign the Controls

The sliders and move button are simple but a little awkward to use. Can you come up with new widgets that make the rover direction more controllable? (Eg go left, go right, forward, backwards, spin right, spin left by making the two wheel speeds linked rather than separate.)

For information on the PushButton and other widgets look at:

https://lawsie.github.io/guizero/pushbutton/

B. Show Status Indicators

When a button is pressed on the rover this is communicated back to the controller. Can you add widgets to show what is happening? This would inform the controller when the rover is disconnecting, stopped, being rebooted or shutdown.

Status messages such as “Moving” can also be interpreted and presented to the user.

C. Add Camera Controls

You can make the rover take photos and videos using the “Photo” and “Video” commands. The duration slider can be re-used to set the video length or you can add another one. It is probably best to take photos only when the rover is stopped to prevent blurring. Add disable and enable controls to make photos only available when the rover is not moving.

Photos and videos can be viewed by shutting down the rover, plugging it into a screen, keyboard and mouse and looking inside the “Rover” folder on the desktop. The folder also contains a log of what the rover was doing in response to the controller.

D. Draw a Map

The rover has no guidance system of its own, but you could draw a map of where the rover is expected to have travelled if it has followed your commands correctly. Do not forget to include any emergency stops that affect the route taken by the rover. Can you convert these into sequence of “Move” commands that can be issued to the rover to bring it back to where it started? Do you think it would return precisely to its starting point? If not, why not?

E. Study the Connection Code

The code to make Bluetooth connections and send and receive data was pre-written for you. In the advanced exercises you will have to write the code yourself. Have a go at understanding what is happening and ask a volunteer if you need any help.

F. Suggestions for Improvements

Do you think the rover would benefit from additional commands? Ask one of the volunteers if you would like a command adding for you to program into your controller.