David Schulz & Andrew Ruswick

Lab 4 – Server Client Programming

CS-4981-021

10/04/2021

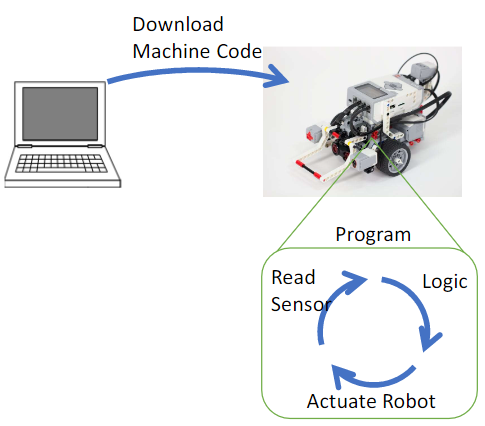
**Lab 4: Server-Client Programming**

**Abstract**

In this lab, we wrote our own server-client program based off of the provided sample code. We developed a messaging protocol for sending and receiving commands to the robot in order to carry out tasks wirelessly, such as live sensor querying and motor function. Also, by utilizing our server-client protocol, we rewrote the line following program that we developed in Lab 3. We were able to successfully complete each of the five tasks and in doing so, my partner and I both gained a much more solid understanding of the many benefits that server-client control has to offer.

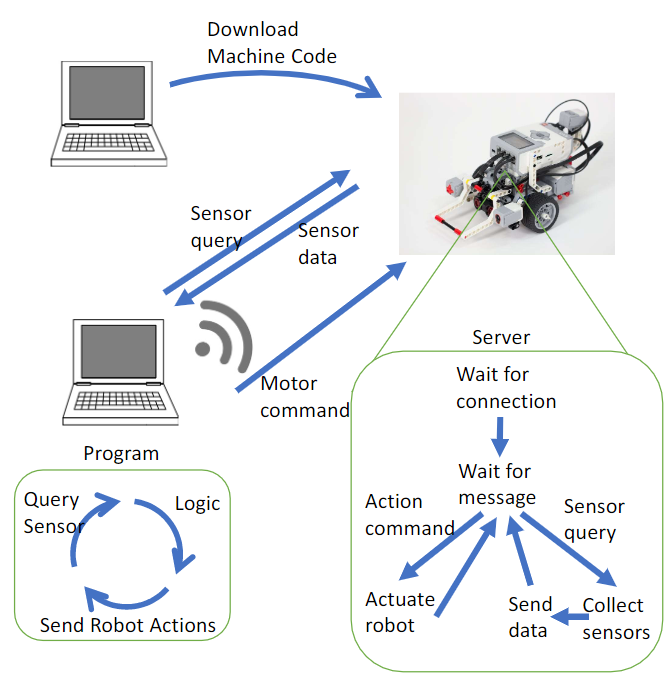
**Methods**

The main goal of this lab was to explore server-client control, an alternative to offline control. The problem with offline control is that it has many limitations such as a slower debugging process and the inability to send live commands to the robot. As shown below in Diagram 1, every time someone wants to test their code, they have to edit, compile, download, and then run the code. This can be a monotonous process and really slow down progress.



*Diagram 1:* The Offline Control Process

Server-client control improves upon these setbacks by granting the programmer the ability to wirelessly send commands to the robot while the machine code is still running. This greatly reduces debugging time and opens up a wide range of programmable tasks that would otherwise not be possible using an offline control system. Diagram 2 shows the server-client control system at work. As you can see, once the machine code is downloaded and running, the client is still able to send motor commands and sensor queries to the robot. The diagram also illustrates the server-client protocol that is established so that the server and client can communicate back and forth.



*Diagram 2:* The Server-Client Control Process

The protocol that the server and client both abide by is extremely crucial for wireless communication. It allows for the PC (client) and the robot (server) to set up a secure connection between them. Once the connection has been established, the server and client are able to talk to each other. They do this by sending, reading, and waiting for messages using tokens. The tokens we use are “forward”, “turn”, “sensors”, “wasd”, “sensor”, and “motors”. Each command has at least one argument after the token separated with a “:”. “Forward” and “turn” are used for task 2 for moving the robot forward and turning 90 degrees respectively. The argument for the forward command is the distance in centimeters, while the argument for the turn command can either be “left” or “right” for the direction in which the robot should turn 90 degrees. “sensors” is used to query all of the sensor readings for task 3, and the argument is a string specifying which port of the brick we want to get the sensor reading from. “wasd” is the command used with the key-controlled mode for task 4. Its argument is a string specifying which key is being pressed. “sensor” and “motors” are used for line following in task 5. “sensor” repeatedly reads the color sensor until the value changes and then sends the value back to the client. The command also takes an argument in order to follow the command syntax, but it doesn’t use it. “motors” is used to set the speeds of the two motors. It takes two integer arguments separated by a comma.

**Results**

**Task 1**

We began task 1 by retrieving the EV3’s Bluetooth address and setting it as a constant inside of the client program. This helped to easily link the server and client together so that they can communicate with each other. After that we ran the server and client programs in unison to make sure that they were both properly connected.

**Task 2**

For task 2, we programmed our server/client communication protocol to send a command to the robot to move forward a specified distance and then another command to turn the robot 90 degrees either left or right. The speeds and times of both were set to the same values as the most accurate values we used in Lab 1. That way, we could take measurements using our approach in this lab and then perform t-tests to compare them with the ones from Lab 1.

Moving Forward

* Using our server client method, we got the following measurements when telling the robot to move forward 25 cm:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 24.4 cm | 24.6 cm | 24.7 cm | 24.8 cm | 24.9 cm |

* + Mean (x̅1): 24.68 cm
  + Std (s1): 0.192
* If we recall from Lab 1, when using the offline method, our measurements were the following:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 25 cm | 25.1 cm | 25.2 cm | 25.5 cm | 25.6 cm |

* + Mean (x̅2): 25.28 cm
  + Std (s2): 0.259
* α = 0.05 and df = n-1 = 4, so expected t-value is 2.776
* |t| > 2.776, so the difference between the means is statistically significant.
* Because the t-test indicates that the difference is statistically significant, the server/client method doesn’t quite work the same way as the offline method. This may, however, be due to human error, as the task may have been done somewhat differently back when we did Lab 1.

Turning 90 Degrees

* Using our server client method, we got the following error measurements when telling the robot to turn 90 degrees clockwise:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 mm | 5 mm | 1 mm | 4 mm | 8 mm |

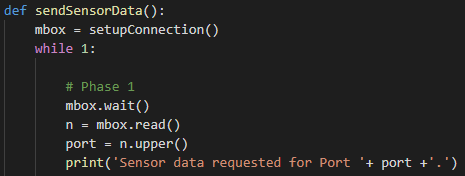
* + Mean (x̅1): 4.2 mm
  + Std (s1): 2.588
* If we recall from Lab 1, when using the offline method, our error measurements were the following:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 mm | 3 mm | 6 mm | 7 mm | 9 mm |

* + Mean (x̅2): 6 mm
  + Std (s2): 2.236
* α = 0.05 and df = n-1 = 4, so expected t-value is 2.776
* |t| < 2.776, so the difference between the means is **not** statistically significant.
* Because the t-test indicates that the difference is not statistically significant, the server/client method works the same way as the offline method.

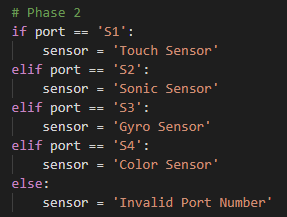
**Task 3**

We started task 3 out by attaching the touch, ultrasonic, gyro, and color sensor to the robot. We were tasked to create a server-client protocol that allows the client to query a sensor port on the robot and get back a data reading from the server. This task was broken up into three specific phases. For phase 1, the client sends a port number request, and the server receives it. Phase 2 had us program the server to look up the type of sensor that was occupying the requested port, which can be seen below in Figure 2. And for phase 3 the server queries the sensor, formats the data, and sends it to the client. Figure 3 shows the code snippet that we wrote to carry out phase 3.

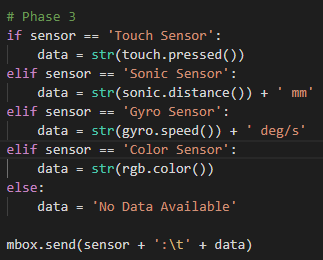


*Figure 1:* A snippet of code that sets up a connection with the client, waits to receive

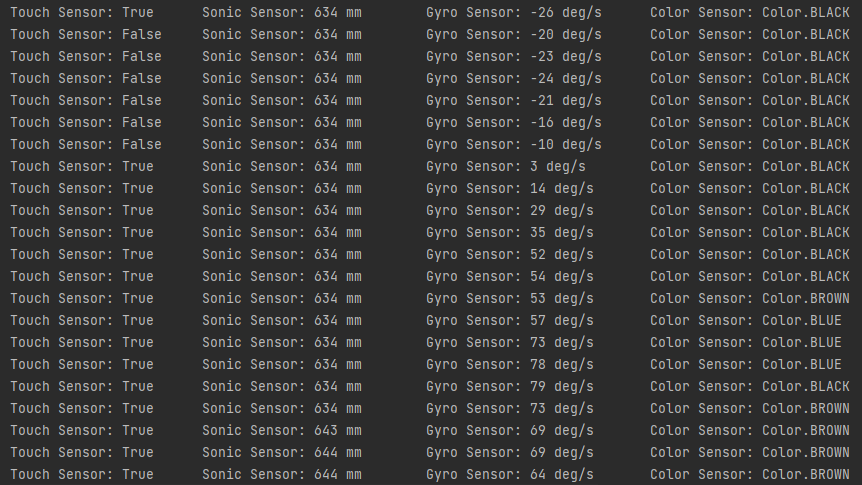
the sensor request and reads the desired port number.



*Figure 2:* A screenshot of the code that looks up the sensor connected to the requested port.



*Figure 3:* The code used to query the sensor, format the data, and send it to the client.



*Figure 4:* A screenshot of the client’s console, showing the live data stream of the four sensors from Task 3.

**Task 4**

For task 4, we used our message protocol to start a program that runs constantly in a loop, checking the last key that was pressed and moving the robot in a certain direction based on whether W, A, S, or D were pressed, and then stops if the last key pressed was the spacebar.



*Figure 5:* A screenshot of the video we recorded of the robot being controlled with the WASD keys.

**Task 5**

For task 5, we translated our finite state machine from Lab 3 to be usable with server/client communication protocols. This time, the logic of the FSM itself happens on the PC hosting the client and the server on the robot only has to send color sensor readings and receive motor commands.

A toy car on a table

Description automatically generated with low confidence

*Figure 6:* A screenshot of the video we recorded of the robot following the tape.

**Discussion**

1. Explain the difference between a 1-sample and 2-sample t-test? Why are we using a 2-sample test here instead of 1-sample?
   * A 1-sample t-test compares the mean of a single series of measurements with an expected value, while a 2-sample t-test compares the means of two series of measurements with each other. We use a 2-sample t-test in task 2 because we are comparing the measurements we took using the offline approach in Lab 1 with the measurements we took using the server/client approach in this lab.
2. Was there a statistical difference between telling the robot to move or turn with the offline control from Lab 1 and the server-client control you created? If there was no difference, what factors could lead to a difference? If there was a difference, where do you think it came from? Hint, think about the cons for server-client control.
   * There was a statistical difference when telling the robot to move forward, but no statistical difference when telling it to turn. Regardless of whether there was a difference or not, some factors that can cause a difference involve making the robot take a longer amount of time to perform the same operations as they would in an offline approach, such as getting the communication to work correctly, managing the formatting of the messages that are sent, and lag.
3. What were some of the designed considerations you thought about when coming up with the message protocol for motor commands and sensor query?
   * For task 3, we made sure to be as concise and simple as possible with our messages so that we could reduce as much lag as possible. This is evident in *Figure 4*, a screenshot of our live data stream that continuously prints sensor data to the client’s console. As you can see, the message is short and to the point, containing only the necessary sensor data needed to inform the client.
4. What are some of the applications of controlling a robot live like you did with the WASD program?
   * The applications for live control of a robot are seemingly endless. To name just a few, a military drone strike carried out from thousands of miles away, rescue missions that require the robot to traverse tough terrain, and robots used on construction sites to carry and move heavy material around.
5. What was the most difficult part of this Lab?
   * For the most part, we both thought that this lab was not difficult at all. We had a little trouble connecting the client and server when we first started the lab, but once we made sure that the client had access to the pybricks folder, that was no longer an issue. We also had a little difficulty translating the FSM to a server/client structure, but it was also quickly resolved.

**Supplementary Material**

**Section 1**

Task4WASD.mp4: The video we took of the robot being controlled by the WASD keys for task 4

Task5Line.mp4: The video we took of the robot following a line for task 5

server\_main.py: The code that runs on the robot to receive motor commands and send sensor readings for tasks 2, 4, and 5

client\_main.py: The code that runs on the PC to query sensor readings, send motor commands, and run the logic for tasks 2, 4, and 5

stateMachine.py: The code that is used for the structure and functionality of the finite state machine for task 5

Task\_3\_Server.py: The code that runs on the robot to receive motor commands and send sensor readings for task 3

Task\_3\_Client.py: The code that runs on the PC to query sensor readings, send motor commands, and run the logic for task 3

**Section 2**

Lab4Code.pdf: A PDF file containing all of the code from every file, separated and labelled by python file in the same order as stated above