Capstone Python Project – ***Features***

CSSE 120, Introduction to Software Development

General instructions:

The following assumes a 3-person team. If you are a 2-person or 4-person team, see your instructor for how to deal with that.

General Project Requirements:

- All features MUST be implemented in a ***Graphical User Interface (GUI)***. This means that you ***should NOT be using the Console for any of your input or output.***
    - All team members ***must*** contribute to the GUI.
- ***To earn a passing grade, a team member must:***
    - ***work well with her team, integrating her work into the team's code consistently throughout the project.***
    - complete one ***green*** feature, one ***gray*** feature, one ***blue*** feature, and at least attempt a ***yellow*** feature.

  This is a ***BARE MINIMUM*** and will result in a C if you demonstrate good process and reasonably high-quality code, else a lower grade.

  Green and grey features are simpler, blue are more sophisticated, and yellow are rather challenging. Uncolored features are open-ended and provide room for creativity.
- ***To earn a higher grade, a team member must*** complete the ***minimum requirements*** above and also provide a complete and effective solution to her yellow feature. To earn an A, she must also complete additional feature(s), including at least one that is genuinely challenging. There is no set number for this; do the best you can (and ask your instructor for guidance as needed). To earn an A, at least one feature must be intellectually challenging, that is, requires code that is sophisticated and demonstrates mastery of skills from this course.

  Also, to earn a high grade, a team member must ***use a strong software development process*** and ***provide high-quality code.*** (See the section on ***Grading*** for more on this.)
- Use as many ***different kinds of GUI widgets*** as you can.
- Use a strong software development ***process*** throughout.

The best projects will take care to re-use each other's GUI, functions and data wherever practical.

Grading and demos:

The grading is based on the ***success of the team as a whole*** as well as ***your own individual contributions*** to the team, including but not limited to:

- Did you implement your required features, with correct and complete code?
- To what extent did you go beyond that, both in quantity and in level of challenge?
- Did you use a strong software development process, including using Trello throughout to track your work, keeping track of your hours, using meaningful commit messages, and using iterative enhancement?
- Is your code high-quality?
- Is your code documented appropriately?

On Friday of 10th week (or possibly during the weekend or week that follows it), your instructor will require that each team give a demo of all of the features that were implemented. We might ask you to demonstrate green or blue features earlier.

Due date:

The final project code is due at the start of class on Friday of 10th week.

Features:

For each of the following, **ask your instructor for more details as needed**.

1. [Team-coded, with your instructor] The user can:
    - ***connect to the robot wired,*** with a way to specify the robot's port.
    - ***connect to the robot wireless,*** with a way to specify the robot's wifly number.
    - and ***disconnect the robot cleanly***, with an indicator showing whether or not the robot is currently connected.
2. The ***GUI displays the contents of the hours-X.txt file*** for ***each team member***.
    - See Feature #5 below for a continuation of this item.
3. The GUI provides a way for the user to set:
    - ***The speeds of the wheels*** (with some way to indicate ***forwards*** vs. ***backwards***), in some reasonable units. **The entire team should discuss the units to use.**
    - ***The time in SECONDS that the robot should move***.

    Additionally, there are ***functions that the entire team can use to get the current values*** of the above.

    Additionally, there is a ***button or other mechanism to make the robot move*** at the specified wheel speeds for the specified number of ***seconds*** (and then stop).
4. ***The robot can GO STRAIGHT*** for a specified ***DISTANCE (in inches or centimeters)*** in a specified ***direction*** (***forward or backward***) at a specified ***speed***, at some reasonable degree of accuracy. Here, "reasonable" means "reasonably straight" and with accuracy that is generally within a few inches. See Feature #10 for an extension of this Feature that strives for better accuracy.

    Additionally, the code contains ***functions that the entire team can use*** whenever the robot needs to go a specified distance in a specified direction (forwards or backwards).

    The implementation of this feature MUST use relevant aspects of the previous feature – both GUI widgets and functions.

IMPORTANT: All subsequent features should use the ***GUI and functions*** provided by Feature #3 and Feature #4 wherever appropriate.

IMPORTANT: If you find yourself ***writing functions that are almost the same***, then ***refactor the code*** to unify those functions into a single function, wherever possible.

5. **[If you did Feature #4, you are NOT permitted to do this Feature.]** ***The robot can SPIN "in place"*** for a specified number of ***DEGREES*** in a specified ***direction*** (***clockwise or counterclockwise***) at a specified ***speed***, at some reasonable degree of accuracy. Here, "reasonable" means accuracy that is generally within 30 degrees or so. See Feature #10 for an extension of this Feature that strives for better accuracy.
6. The GUI allows the user to:
    - ***Display the current values of ALL of the sensors on the robot.***
7. The GUI allows the user to run a simple TEST that demonstrates quickly whether or not the robot is working correctly. In particular, it must test whether:
    - The motors are working correctly.
    - The sensors are working correctly.
    - The buzzer and LED are working correctly.
8. **[If you did Feature #2, you are NOT permitted to do this Feature.]** The ***GUI indicates,*** for ***each Sprint*** and ***each team member***:
    - The ***total hours*** that the team member worked **during that Sprint**.
    - The ***cumulative hours*** that the team member has worked **up to and including that Sprint**.

    This feature is an extension of Feature #2. Keep the code for Feature #2 unchanged (so that the student can get credit for it). ADD code for this feature. If you wish, you may replace the display from Feature #2 with the display from this Feature.
9. The robot can be ***tele-operated*** (i.e., remote-controlled, like a remote-control car) with **keyboard keys** (and optionally, other ways to control the robot). **The robot MUST be responsive**, just as when you drive a remote-control car.

    The user should be able to set speeds using the GUI item(s) from Features 3 and/or 4. If the keys themselves also modify speeds, the current speed should be displayed.
10. The GUI allows the user to set, for any sensor that the user wishes, a "threshold". The GUI then allows the user to make the robot move until that sensor is bigger than that threshold value. Additionally, it can do the same, but moving until that sensor is ***smaller*** than that threshold. Additionally, the user can choose from any of four movements: forward, backward, spin in place clockwise and spin in place counterclockwise.

    Note that encoders are sensors, so this will supply a way to achieve more accurate movement. **Provide functions that are *improved versions of the functions that Features #4 and #5 provided*. Keep the code for Features #4 and #5 unchanged** (so that the student can get credit for it). ADD code for this feature.

    Note: This feature is perhaps somewhat too big. Not too hard, just big. ***See your instructor*** for how you can implement this feature without a lot of repetition.

11. [Team-coded, with your instructor] ***Long-running loops can be interrupted.***

IMPORTANT: All features should be designed to re-use existing code, re-use existing GUI widgets, and be re-usable by yet-to-be-implemented code, wherever practical. For example, Feature #12 should use GUI widgets from Feature #10.

12. ***The robot can use its reflectance (line) sensors to follow a curvy black line.*** You must first provide an implementation that uses ***Bang-Bang control*** and then a separate implementation that uses ***PID control***.

13. ***The robot can use its encoders to have the robot go straight per Feature 4 no matter how much stronger one motor is than the other.*** You must first provide an implementation that uses ***Bang-Bang control*** and then a separate implementation that uses ***PID control***.

14. ***The robot can use its camera and front proximity sensor to follow an object.*** For each part of this challenging Feature, you must first provide an implementation that uses ***Bang-Bang control*** and then a separate implementation that uses ***PID control***. First provide an implementation that uses the camera to spin in place as the object moves left or right. Then provide an implementation that uses the front proximity sensor to keep the robot at a constant distance from the object as the object moves away from or toward the robot. Finally, try to integrate the actions of the two sensors.

15. Provide a way for the user to specify a list of x/y coordinates (***waypoints***). Then, given the waypoints, make the robot ***move to each***, one by one, pausing briefly at each waypoint, using Manhatten movement (i.e., all movement is along the x-axis or y-axis).

    Optionally, you can display the movement of the robot on a Canvas on the GUI, either as each waypoint is reached or in "real time" as the robot is moving.

16. The robot can ***"talk" to another*** robot using movements and/or other forms of communication. You will need two robots for this, each running the same program. To "talk" to another robot means that:

    - Your code allows the user to tell one robot that it is the Talker, and to tell the other robot that it is the Listener.
    - Then, the Talker does something to communicate a single "bit" of information to the Listener. Meanwhile, the Listener is listening for that communication and displays the result of it (0 or 1) when the communication occurs.

    Optionally, you can extend this so that the Talker can say more than just 0 or 1.

    Optionally, you can extend this so that after the Listener hears something, the Listener becomes the Talker and the former Talker becomes a Listener (ideally without the user intervening).

17. The robot can ***parse files with songs and play the songs*** using its buzzer. Note: This Feature is great fun, but you receive no more credit for parsing and playing one song than for parsing and playing 100 songs.

    Most forms of this Feature will NOT satisfy the condition (required to earn an A) of being intellectually challenging, that is, requires code that is sophisticated and demonstrates mastery of skills from this course.

18. The robot can move quasi-randomly, using its sensors to ***avoid objects, stop at lines***, and more.

19. The robot can ***display emotion***.

20. The robot can ***compose*** music, and then play its compositions.
21. The robot can ***watch a conductor and play music accordingly***.
22. The robot can ***compose a fictitious bio*** for itself and/or for you.
23. The robot can do ***sophisticated movements***, e.g. trace a regular polygon, parallel park, and more.
24. The robot can ***tweet***! (Via twitter)

    Optionally, the robot can get tweets from other users and respond to them in some slightly intelligent way. (Google for "Eliza".)
25. Use a ***Leap Motion device*** (and accompanying Python software) to control the robot with hand movements.
26. The robot can move through a maze without hitting any walls, where the maze allows the robot to go only one way (left or right) at each intersection.

    Optionally, the maze is a real maze (with options for going both left and right, and with dead ends). Here the robot should attempt to continue at dead ends by backing up and then … [lots of possibilities here].
27. Do ***interesting things*** (beyond those already listed) with the robot's ***camera***.
28. Do ***interesting things*** (beyond those already listed) with the robot's **standard sensors**.
29. Do interesting things with ***additional motors, servos and/or sensors.***
30. Use ***swarm techniques*** and/or distributed algorithms to accomplish interesting things.
31. Use ***parallel algorithms*** (in processes and/or threads, in a single processor or across cores) to accomplish interesting things.
32. Use ***internet communication*** and/or ***files*** to do interesting things.
33. Make the low-level communication more efficient, or otherwise ***augment what the Arduino can do*** on behalf of the Python program that is controlling the Arduino.
34. ***Interact with a different kind of robot***, e.g. a quadcopter or BERO robot.
35. Do something interesting… ***[You suggest what!]***