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.

All features must be implemented in a nice ***Graphical User Interface*** (***GUI***) to which all team members must contribute.

- All input and output must be from your GUI (plus any optional external devices you might use, like Wiimotes). There must be ***no input or output from/to a Console window*** (except for debugging purposes).
- The more ***different kinds*** of GUI widgets, the better.
- The more your follow good GUI design principles[1] (and you can ***explain how your GUI does so***), the better.

There are ***green***, ***blue***, ***yellow***, and ***uncolored*** features. Greens and blues are simpler robot ideas. Yellows are more sophisticated robot ideas (that are more challenging to implement). Uncolored features are open-ended opportunities that vary wildly in difficulty.

Each student must implement *one green feature*, *one blue feature* and *one yellow feature*. For a good score, students must *also* implement some (not all!) of the uncolored features.

Many features contain "advanced options". Students who want a high grade will want to implement many of these. However, do not feel obliged to implement *all* of them, since implementing *uncolored* features is another way to earn a high score.

[1] Ben Schneiderman, *Eight Golden Rules of Interface Design*. https://www.cs.umd.edu/users/ben/goldenrules.html.

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

Your grade is based on the features you implement, the process you use in doing so, the quality of your code, and more. See the grading rubric for details.

Features (brief version – see long version for full details):

1. The user can ***connect and disconnect*** to the robot, after specifying whether or not to use the simulator and if not, what port to use for connecting. The program should behave reasonably if the user errs (e.g. by choosing a wrong port, or connecting to an already-connected robot).
2. The ***GUI indicates,*** for ***each Sprint*** and ***each team member***, ***the total hours that the team member worked on that Sprint.*** The data for this item should be ***read from the files*** for *hours-worked* that each team member must maintain, in the *process* folder of the project.
3. Play ***N random notes***, where the user specifies N. The notes must not be "clipped".
4. Be ***tele-operated*** (i.e., remote-controlled, like a remote-control car). The user can make the robot move in any direction (forward/backward, spin left/right) at any speed.
5. ***Move autonomously***, by going a specified ***distance*** in a specified ***direction*** at a specified ***speed***. That is, the user can set the direction (forward, backward, spin left or spin right) and the distance and speed (each in some reasonable units). Then, the user can make the robot go (e.g. by pressing a *Go* button) and the robot should move the specified direction for the specified distance at the specified speed, with some reasonable accuracy.

   An important by-product of this feature is to provide a good set of functions that teammates will use for most of the movements that they ask of the robot.
6. ***Move autonomously***, by going until a specified ***sensor*** reaches a specified ***threshold***. Sensors should include the bump sensors and the 4 cliff sensors, at the least.
7. ***Follow a curvy black line*** using PID control.
8. Move through a sequence of user-specified ***waypoints***.
9. Hold a ***conversation*** with another robot.
10. ***Follow another robot***.
11. ***Sing*** and ***dance*** with a ***light show***.
12. Compose music.
13. Do sophisticated movements, e.g. trace a regular polygon or parallel park as in the video at https://www.youtube.com/watch?v=N4F0-MXK5jM.
14. Do interesting things with its ***internal*** sensors.
15. Do interesting things with ***computer vision*** (using the camera), e.g. finding objects, using semaphores to communicate, or …
16. Do interesting things with ***external motors and/or servos.***
17. Offer ***Rogerian psychotherapy***, ala Eliza (http://en.wikipedia.org/wiki/ELIZA).
18. Use ***swarm techniques*** and/or distributed algorithms to accomplish interesting things.
19. Use ***parallel algorithms*** (in processes and/or threads, in a single processor or across cores) to accomplish interesting things.
20. Use ***internet communication*** and/or ***files*** to do interesting things.
21. ***Compose a fictitious bio*** for itself and/or for you.
22. Use a ***Leap Motion device*** (and accompanying Python software) to control the robot with hand movements.
23. ***Interact with a different kind of robot***, e.g. a quadcopter or BERO robot.

24. Do something interesting… ***[You suggest what!]***

Features – with details:

1. The user can ***connect and disconnect*** to the robot, after specifying whether or not to use the simulator and if not, what port to use for connecting.

   The program should behave reasonably if the user errs (e.g. by choosing a wrong port, or connecting to an already-connected robot).

2. The ***GUI indicates,*** for ***each Sprint*** and ***each team member***, ***the total hours that the team member worked on that Sprint.*** The data for this item should be ***read from the files*** for *hours-worked* that each team member must maintain, in the *process* folder of the project.

   Whoever implements this feature determines the format for the *hours-worked* files, and conveys that format to her teammates. Each team member maintains her own file per the format.

3. Play ***N random notes***, where the user specifies *N.* The notes must not be "clipped". Hint: Use the *song_playing* sensor appropriately to avoid clipping.

   Additionally, the use can specify the length of time each note should be played -- either a fixed length of time, or a range from which the time should be chosen at random.

4. Be ***tele-operated*** (i.e., remote-controlled, like a remote-control car). The user can make the robot move in any direction (forward/backward, spin left/right) at any speed, stopping her whenever the user wants. The simplest implementation would use buttons for the movements.

   Advanced options include:

   - Uses easy-to-operate interfaces like keys (without interfering with other features!), gamepads, wiimotes, or other remote-control devices (perhaps wireless).
   - Can move in curves.
   - There is more than one user interface for tele-operation (e.g. buttons and keys).
   - The user can disable/enable tele-operation.

5. ***Move autonomously***, by going a specified ***distance*** in a specified ***direction*** at a specified ***speed***. That is, the user can set the direction (forward, backward, spin left or spin right) and the distance and speed (each in some reasonable units). Then, the user can make the robot go (e.g. by pressing a *Go* button) and the robot should move the specified direction for the specified distance at the specified speed, with some reasonable accuracy.

   An important by-product of this feature is to provide a good set of functions that teammates will use for most of the movements that they ask of the robot.

   Advanced options include:

   - There are multiple implementations (any of which can be chosen by the user), with demonstrated understanding of when and why one is better/worse than another. For example, one implementation is the "time" approach, another is the "distance sensor" approach (which itself is really a collection of approaches parameterized by the time to wait between sensor readings), and a third is the "send a script" approach.
   - There is high accuracy for the best implementations.
   - Can move linearly and angularly (hence along a curve) at the same time, with some reasonable understanding of "distance" and "speed" in that case.
   - The motion can be interrupted by the user.

6. ***Move autonomously***, by going until a specified ***sensor*** reaches a specified ***threshold***. Sensors should include the bump sensors and the 4 cliff sensors, at the least

   In particular, the user can set the speed and which bumpers to use (both, just-left or just-right). Then, the user tells the robot to start, at which point the robot moves until the relevant bumper(s) are pressed.

   Likewise, the user can set the speed, which of the four cliff sensors to use, and a "darkness level." Then, the user tells the robot to start, at which point the robot moves until the specified sensor sees sufficient "darkness".

   Advanced options include:

   - The user can choose from sensors beyond the bump and cliff sensors.
   - The user can choose one or more sensors to be active in determining when to stop. For example, the user might choose the left bump sensor, the right bump sensor, or both.
   - The user can choose from different kinds of sensors (combining bump and cliff sensors, for example), if a variety of ways.
   - The "stopping condition" can be more than just a threshold whose value is exceeded.
   - The best implementations might require multiple sensors mixed in interesting ways, e.g. the infrared hears 100 followed a second later by 200.
   - Perhaps the coolest implementation would allow the user to supply a function definition (written "on the fly") for the stopping condition.

7. ***Follow a curvy black line*** using bang-bang and PID control.

   See the PID video for an explanation of bang-bang and PID control.

   Assume a curvy black line about 2 inches wide, with reasonably gentle curves, using the left front signal (for the left wheel speed) and the right front signal (for the right wheel speed). (You can also use other sensors if you wish.)

   First implement bang-bang control. Then implement ***P (proportional) control***, with the P constants tuned reasonably.

   Warning: The simulator is WAY different from real robots for this feature. Start with the simulator, but realize that real robots require SUBSTANTIAL tuning. (Their IR light sensors vary wildly from robot to robot, and even within a robot!)

   Advanced options include:

   - Auto-calibrates the darkness of the lines under current lighting conditions by the human placing the robot in positions as desired (with no changes to the program needed for this process).
   - Use I and D (the rest of PID). Make a line where they help.
   - The user can set all the parameters at run-time, ideally even while the robot is doing line-following.
   - The line-following can be interrupted by the user.
   - Uses additional sensors to enable following more challenging lines.
   - Can follow a curvy wall, using a "bump and bounce" algorithm that is akin to bang-bang control.
   - Can follow a curvy wall, using the wall sensor but (ideally) the same PID code as for line following.

8. Move through a sequence of user-specified ***waypoints***.

That is, the user can enter a sequence of (x, y) coordinates and tells the robot to go. Then, the robot moves to each, one after the other. (The origin of the coordinate system is where the robot began the sequence of moves.)

Advanced options include:

- There is a nice way to enter coordinates (e.g. by clicking on a map displayed in a window).
- The path of the robot is shown on a window as the robot moves.
- The movement can be interrupted by the user.
- Coordinates can come from a file.
- The robot can move around obstacles as it moves from waypoint to waypoint.
- User can control speeds as well (perhaps via pre-specification, perhaps via tele-operation, perhaps both).
- The robot remembers paths on which it is tele-operated and then can reproduce the paths autonomously.
- The robot keeps track of its position through ALL its movements (even those produced by teammate's code) and can reproduce them from any point the user specifies.

9. Hold a ***conversation*** with another robot.

Note: This description uses IR, but other sensors might be able to be used in a similar way. *Check with your instructor before beginning to implement this feature!*

As a stepping-stone to conversation, first implement the following:

- Pressing a button makes the robot start (and continue indefinitely) sending a signal specified in an entry box. This action must not "block".
- Pressing another button makes the robot stop sending that signal.
- Pressing a third button makes the robot start listening and then display what signal it hears, once it hears something. This action is permitted to "block".

Then implement a simple protocol per the Protocol video that allows two robots to "converse" by sending and hearing signals.

Advanced options:

- The listening and sending can be interrupted by the user.
- Uses codes to send letters, words and entire phrases, per a coding system read from a file.
- Encrypts and decrypts (perhaps as simple as Caesar's cipher, or as complicated as a public key encryption system).
- Does a more advanced protocol, e.g. handshaking to identify itself.

10. ***Follow another robot***.

    Uses the camera, or uses IR emitters with black tape over the IR sensor for directionality, or the other robot sends codes to indicate directionality, or

11. ***Sing*** and ***dance*** with a ***light show***.

    IMPORTANT: You get NO credit for time spent typing in long sequences of notes, or long periods of time transliterating a song to the Create's MIDI note system. Focus on the interesting things you can do via the programming.

    Options include:

    - Songs of more than 16 notes.
    - Plays MIDI from a file.
    - Does the light show while dancing and singing, perhaps choreographed.

12. Compose music. Composes music – randomly, or with principles from music theory. Plays the music. Option: do likewise for dances and/or light shows.

13. Do sophisticated movements, e.g. trace a regular polygon or parallel park as in the video at https://www.youtube.com/watch?v=N4F0-MXK5jM.

14. Do interesting things with its ***internal*** sensors. One simple example: there is a sensor that, perhaps surprisingly, can tell when a robot is "stuck" even when the robot is attempting to move BACKWARDS.

15. Do interesting things with ***computer vision*** (using the camera), e.g. finding objects, using semaphores to communicate, or …

16. Do interesting things with ***external motors and/or servos.***

17. Offer ***Rogerian psychotherapy***, ala Eliza (http://en.wikipedia.org/wiki/ELIZA).

18. Use ***swarm techniques*** and/or distributed algorithms to accomplish interesting things.

19. Use ***parallel algorithms*** (in processes and/or threads, in a single processor or across cores) to accomplish interesting things.

20. Use ***internet communication*** and/or ***files*** to do interesting things.

21. ***Compose a fictitious bio*** for itself and/or for you.

22. Use a ***Leap Motion device*** (and accompanying Python software) to control the robot with hand movements.

23. ***Interact with a different kind of robot***, e.g. a quadcopter or BERO robot.

24. Do something interesting… ***[You suggest what!]***