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 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.
- Each team member must complete one green feature, one blue feature, and one yellow feature.
  - This is a ***BARE MINIMUM*** and will result in a C for the project.
  - Green and blue features are simpler, yellow are more sophisticated.
  - Uncolored features are open ended and provide room for creativity.
  - Several features contain "Advanced Options" that are not required, but are recommended for a higher grade.
- To receive an A on this project, you must complete the minimum requirements above, some of the uncolored features, and some of the advanced features. There is no set number for this, do the best your team can.
- Use as many different GUI widgets as you can.

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 quality of your individual contributions to the project;
- your attention to detail to the Trello board, SVN, and your code.
- completeness of each feature based on the specification below.

On Friday of 10th week, your instructor will require that each team give a brief (5-10 minute) demo of all of the features that were implemented.

Due date:

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

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

0. 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).
1. 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.
2. Play ***N random notes***, where the user specifies N. The notes must not be "clipped".
3. ***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.
4. ***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.
5. Be ***tele-operated*** (i.e., remote-controlled, like a remote-control car) with a **Roomba remote**. The user can use the d-pad to make the robot move in any direction (forward/backward, spin left/right) at three different speeds. See the next page for **implementation requirements**.
6. ***Follow a curvy black line*** using PID control.
7. Move through a sequence of user-specified ***waypoints***.
8. ***Tele-operate other robots using IR hat.***
9. Robot ***composes*** new music.
10. Do ***sophisticated movements***, e.g. trace a regular polygon, parallel park.
11. Do interesting things with its ***internal*** sensors.
12. Do interesting things with ***external motors and/or servos.***
13. Use ***swarm techniques*** and/or distributed algorithms to accomplish interesting things.
14. Use ***parallel algorithms*** (in processes and/or threads, in a single processor or across cores) to accomplish interesting things.
15. Use ***internet communication*** and/or ***files*** to do interesting things.
16. ***Compose a fictitious bio*** for itself and/or for you.
17. Use a ***Leap Motion device*** (and accompanying Python software) to control the robot with hand movements.
18. ***Interact with a different kind of robot***, e.g. a quadcopter or BERO robot.
19. Do something interesting… ***[You suggest what!]***

Features – with details:

0. 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.

   Features:

   - User enters port number or 'sim' for simulator.
   - Connect to Simulator
   - Connect to Robot
   - Disconnect

   Advanced:

   - Single Connect button that handles both simulator and real robot.
   - Program checks for and handles bad user input (e.g. wrong port, connecting to already-connected robot, without crashing.

1. The ***GUI indicates,*** for ***each Sprint*** and ***each team member***, ***the total hours that the team member worked on that Sprint.*** Features:

   - Reads hours for each team member from files.
   - Total hours for each sprint for each team member displayed.
   - Hours are ***displayed in the GUI***, ***NOT*** the console.
   - Each team member maintains their hours 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.

   Advanced:

   - Can show and hide information.

2. Play ***N random notes***, where the user specifies *N.*

   Features:

   - User specifies number of notes to play
   - **Notes are not "clipped"**. Hint: Use the *song_playing* sensor appropriately to avoid clipping.
   - User specifies the length of time each note should be played.

   Advanced:

   - User may also specify a range of time lengths, from which the time for each note should be chosen at random.
   - Same input area is used for both a single length of time and a range of times.

3. ***Move autonomously***, by going a specified ***distance*** in a specified ***direction*** at a specified ***speed***.

   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.

   Features:

   - Movement Direction options: ***Forward, Backward, Spin Left, Spin Right***
   - User specifies distance (in some reasonable units)
   - User specifies speed (in some reasonable units)
   - User starts motion with a "Go" button.
   - After "Go" is clicked, robot moves in the specified direction for the specified distance at the specified speed with reasonable accuracy.

   Advanced:

   - Motion can be interrupted by the user.
   - There are multiple implementations (any of which can be chosen by the user), such as:
     - Time approach (calculating how long to move based on inputs)
     - Distance sensor approach
     - Another approach you can think of.
   - The best implementation has high accuracy and repeatability.
   - Linear and angular movement (hence along a curve) at the same time, with some reasonable understanding of "distance" and "speed" in that case.

4. ***Move autonomously***, by going until a specified ***sensor*** reaches a specified ***threshold***.

   Features:

   - User sets speed
   - User tells robot to start, and robot moves until the relevant bumper(s) are pressed, or cliff sensors detect a value outside the threshold.
   - Uses at least the following sensors for stopping movement:
     - Bump Sensors
     - All 4 Cliff Sensors
   - User sets which bumpers to use (both, just-left, or just-right)
   - User sets which cliff sensors to use
   - For cliff sensors, user sets threshold "darkness" level for when to stop.

   Advanced:

   - 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.

5. Be ***tele-operated*** (i.e., remote-controlled, like a remote-control car) with a **Roomba remote**.

   Features:

   - The user uses the d-pad to make the robot move in any direction (forward/backward, spin left/right) at three different speeds:
     - **Slow**: 20 cm/s
     - **Medium**: 30 cm/s
     - **Fast**: 40 cm/s
   - If the user lets go of a movement button, the movement stops.
   - Assign the Roomba remote buttons to the following actions:
     - Spot: set to Slow speed
     - Clean: set to Medium speed
     - Max: set to Fast speed
     - *Hold* D-pad up: move forward at the set speed
     - *Hold* D-pad left: spin left in place at the set speed
     - *Hold* D-pad right: spin right in place at the set speed
     - *Hold* D-pad down: move backward at the set speed
   - User may choose to use arrow keys and other keys of your choice instead of Roomba remote to perform the same actions.

   Advanced:

   - Make the **P** button at the top of the remote change the robot into **sound mode**.
   - ***While in Sound mode***, pressing the remote buttons does not move the robot, but instead causes the robot to play different sound tones. Pressing the P button *again* goes back to movement mode.
   - Make the **red pause button** on the remote toggle tele-operation.
     - With this option, the robot should always listen for the IR signal, and when the user presses the red button, the robot will begin tele-operation.
     - When the user presses the red button again, the robot should stop tele-operation and allow other features to execute normally.
     - NOTE: This feature requires threading that runs in the background: http://stackoverflow.com/questions/2846653/python-multithreading-for-dummies

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

See the PID videos (http://www.rose-hulman.edu/class/csse/csse120/VideoFiles/PID_control/PID_control.html and http://www.rose-hulman.edu/class/csse/csse120/VideoFiles/PID_control2/PID_control2.html) 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.

Features:

- Can select Bang-Bang control
- Can select P control
- Robot follows large curved line on mat (will show in class)

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!) You **must** provide an interface to calibrate the darkness of the lines under current lighting conditions. The human may place the robot in positions as desired. You must not hard-code the darkness of the lines into the program.

Advanced options include:

- Use I and D (the rest of PID). Make a line where they help.
- User may choose to P or PI, PD, or PID control.
- The user can set all the PID constants at run-time.
    - Ideally this could be done 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.

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

Features:

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

Advanced options include:

- There is an easy 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.

8. Connect an IR "hat" to the robot and **tele-operate other robots**.

   The robot can wear an IR transmitter ("hat") accessory that broadcasts in all directions.

   Features:

   - Program the robot to broadcast the same IR signals as the Roomba remote when specific keys are pressed.
   - Robot emitting IR should move when keys are pressed (you should have implemented movement based on keys in #6, updating that code to start transmitting IR is a good solution.)
   - Then gather several robots and run them in tele-operated mode USING KEYS (you'll need to use multiple instances of your application, or multiple computers for this). If you use the remote to control the robot emitting IR, you'll get interference between the remote and the robot's signal.

   The robot wearing the IR hat should transmit signals to control the other robots and lead a **line dance**.

   Advanced:

   - Use a **single instance** of your application to control all of the robots.
   - Have the robot read dance moves from a **file and emit them to other robots**.
   - Have the robot lead a dancing ***and singing*** party. The robot should transmit same signal as the **P** button on the Roomba remote during the dance to make the robots start singing.

   IMPORTANT: You get NO credit for time spent typing in long sequences of dance moves, 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.

9. Compose music.

   Features:

   - Composes Music – This means that the robot creates its own songs. This **DOES NOT** mean that you read existing music from a file or a list of notes. There is **NO CREDIT** for this if the robot plays the same song each time.
   - Uses principles of music theory to pick notes to play.

   Advanced:

   - Compose dances to go with music.
   - Compose light show with robot's LEDs to go with music.

10. 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.

    Polygon Features:

    - Allow user to input number of polygon points
    - Allow user to specify perimeter length, interior width, or other descriptor for size of polygon.
    - Calculate the angle and lines needed for the polygon.
    - Robot drives the polygon.

    Advanced Polygon Features:

    - User can choose if the robot makes left turns or right turns.
    - User can choose if the robot traces the polygon forward or backward.

    Other:

    - Additional credit will be given for other types of sophisticated movements that you can think of, such as parallel parking the robot.

11. Do interesting things with its ***internal*** sensors.

    Features:

    - For this feature, you choose what you do with the sensors, some examples are below.
    - Example 1: There is a sensor that, perhaps surprisingly, can tell when a robot is "stuck" even when the robot is attempting to move BACKWARDS.
    - Example 2: Students have previously implemented a game with the robot in which you attempt to keep the robot on the white line mat. The robot drives forward until it finds the carpet, and if it does, you lose. If you tap a bump sensor, the robot turns and moves in another direction and faster. You win if you keep the robot on the mat for a specified number of hits.

12. Do interesting things with ***external motors and/or servos.***
13. Use ***swarm techniques*** and/or distributed algorithms to accomplish interesting things.
14. Use ***parallel algorithms*** (in processes and/or threads, in a single processor or across cores) to accomplish interesting things.
15. Use ***internet communication*** and/or ***files*** to do interesting things.
16. ***Compose a fictitious bio*** for itself and/or for you.
17. Use a ***Leap Motion device*** (and accompanying Python software) to control the robot with hand movements.
18. ***Interact with a different kind of robot***, e.g. a quadcopter or BERO robot.
19. ***Create a web-based tool that allows data from Trello to get to your project.*** For example, the tool could let you record your hours-worked on Trello cards and it get displayed when your project runs, with the calculations done in "real time." This might be a good feature for someone who is ambitious and interested in web development to try. The URL of the Python wrapper for Trello that you would use is at : https://pythonhosted.org/trello/trello.html. See David Lam for details.
20. Do something interesting… **[You suggest what!]**