Getting started

ONE and ONLY one person on your team *forks* the starting code from the CSSE 120 Home Page. That person adds her teammates and the instructors as collaborators.

Then the remaining teammates ***CLONE*** the project that their teammate forked and put it into PyCharm as usual.

Sprints and Due Dates

Sprint 1:

From beginning of Session 26 (Wednesday, February 6) to 11:59 p.m. Sunday, February 9. **Demonstrate completed features during class in Session 28, Monday, February 11.**

IMPORTANT: Before implementing any features, with your instructor's help, ***master*** **the starting code that you are given.**

Implement Features 0 through 7.

Sprint 2:

From 11:59 p.m. Sunday, February 9 to 11:59 p.m. Wednesday, February 13. **Demonstrate completed features during class in Session 29, Wednesday, February 13 or in Session 30, Thursday, February 14.**

Implement Features 8 through XX.

Sprint 3:

From 11:59 p.m. Wednesday, February 13 to 6 p.m. Wednesday, February 20. Demonstrate completed features via video.

Implement your self-selected Feature(s).

Features

For Sprint 1:

Feature 0: Each person puts her own name into her two ***mX-...py*** files and also into the other (shared) files. ***Coordinate GIT actions for the shared files,*** as directed by your instructor!

Feature 1: Implement the 4 unimplemented methods specified in the **`ArmAndClaw`** class in the shared **`rosebot.py`** module. ***Trio-programming****, with each person driving for at least one method.*

Test by implementing short, simple, throw-away test functions in your **`mX_run_this_on_robot.py`** module. Each person implements tests for at least one of the 4 methods (with the others NOT testing that method).

Feature 2: Study the **shared_gui.py** file. Then, with help from your instructor, implement the beginning skeleton of your module:

`mX_run_this_on_laptop.py`

Test that your shared GUI DISPLAYS correctly. It will not yet DO anything in response to button presses.

(continues on the next page)

Feature 3: Complete the remaining unimplemented methods in the **shared_gui.py** module.

- Person 1: The methods in the section labelled:

  Handlers for Buttons in the Teleoperation frame.

- Person 2: The methods in the section labelled:

  Handlers for Buttons in the ArmAndClaw frame.

- Person 3: The methods in the section labelled:

  Handlers for Buttons in the Control frame.

Include **BOTH *print* statements** indicating that a message is being sent **AND** the ***send_message*** **calls. Test** by confirming that the ***print*** statements work correctly. At this point, the ***send_message*** calls will be sent to nowhere (and hence do nothing).

Feature 4: Complete the following unimplemented methods from the portion of the **DriveSystem** class labelled:

Methods for driving with no external sensor (just the built-in encoders).

- Person 1:
  - ***go***
  - ***go_straight_for_seconds***
- Person 2:
  - ***stop***
  - ***go_straight_for_inches_using_time***
- Person 3:
  - ***go_straight_for_inches_using_encoder***

Test your portion by implementing short, simple, throw-away test functions of your portion of the above in your **mX_run_this_on_robot.py** module.

Feature 5: With your instructor, implement the shared **shared_gui_delegate_on_robot** module. ***Trio-programming****, with each person driving for at least one method.* Exception: you do not need to do the robot functionality for the ***Quit*** and ***Exit*** buttons yet.

Then, with your instructor, add code to your **mX_run_this_on_robot.py** module to use the code in the **shared_gui_delegate_on_robot** module.

Test by running your GUI on your laptop and your robot mX file on the robot (via SSH). Confirm that your shared GUI controls the robot fully, with all buttons implemented for Quit and Exit.

Feature 6: First, with your teammates, write **on paper** how you would like to **extend your shared GUI to include a *DriveSystem* frame** that allows the user to tell the robot to:

- Go straight for a given number of seconds at a given speed (as in the ***DriveSystem*** method ***go_straight_for_seonds***).
- Go straight for a given number of inches at a given speed using the "time" approach (as in the ***DriveSystem*** method ***go_straight_for_seconds_using_time***).
- Go straight for a given number of inches at a given speed using the "encoder" approach. (as in the ***DriveSystem*** method ***go_straight_for_seconds_using_encoder***).

Then, **trio-program the enclosing frame (but not yet the widgets within the frame)**. ***Test*** to ensure that the frame appears (but has nothing except a title-label on it so far).

Finally, **implement the three items listed above, with each person implementing one of the three items**. ***Test*** that the extension to the GUI works completely (actually making the movements on the robot). It is best if you work sequentially in implementing the three items, not in parallel, to avoid conflicts.

Feature 7: First, with your teammates, write **on paper** how you would like to **extend your shared GUI to include a *SoundSystem* frame** that allows the user to tell the robot to:

- Beep for a given number of times.
- Play a tone at a given frequency for a given duration.
- Speak a given phrase.

Then, **trio-program the enclosing frame (but not yet the widgets within the frame)**. ***Test*** to ensure that the frame appears (but has nothing except a title-label on it so far).

Finally, i**mplement the three items listed above, with each person implementing one of the three items.** ***Test*** that the extension to the GUI works completely (actually making sounds on the robot).

It is best if you work sequentially in implementing the three items, not in parallel, to avoid conflicts.

For Sprint 2:

NOTE: For the Sprint 2 features, you will want to watch relevant parts of the videos at:

http://www.rosebotics.org/csse120-ev3/unit?unit=4

Feature 8: Complete the remaining unimplemented methods from the **`DriveSystem`** class:

- Person 1: The methods in the section labelled:

  Methods for driving that use the **`color`** *sensor.*
- Person 2: The methods in the section labelled:

  Methods for driving ... infrared **`proximity`** *sensor.*
- Person 3: The methods in the section labelled:

  Methods for driving that use the **`camera`**. *Note that the display_camera_data() method only needs to display the blob data on the* ***Console*** *(not the GUI)*

Test your portion by adding buttons and entry boxes for the above to your ***DriveSystem*** **frame**, in any way you like. Each person should add the GUI for the methods they implement.

(continues on the next page)

Feature 9: ***Go to an object using the* proximity *sensor and pick the object up,*** as follows:

For this feature, implement it via ***your own mX files***, NOT in shared files. This is an INDIVIDUAL feature, NOT shared.

Assume that there is an object that the robot could pick up that is straight ahead. Then the robot should go to that object, using the ***proximity sensor***, and pick it up. While the robot is going:

- **Person 1:** The robot should beep as it moves, beeping increasingly faster as the robot gets closer to the object. The user should be able to set the initial and rate of increase of the beeping-pace via the GUI.
- **Person 2:** The robot should make tones as it moves, with the tones increasing in frequency as the robot gets closer to the object. The user should be able to set the initial and rate of increase of the frequencies via the GUI.
- **Person 3:** The robot should turn the cycle through left-LED-on / right-LED-on / both-LEDs-on / neither-LED-on, with the cycles occurring increasingly faster as the robot gets closer to the object. The user should be able to set the initial and rate of increase of the LED-cycle-pace via the GUI.

Test your portion by adding buttons and entry boxes for the above to **your OWN frame** for this feature, in any way you like.

Feature 10: ***Use the* camera *to find an object, then go to it and pick it up as in Feature 9,*** as follows:

For this feature, implement it via ***your own mX files***, NOT in shared files. This is an INDIVIDUAL feature, NOT shared.

Train your ***camera*** on the color of an object that the robot could pick up. Then make the robot:

1. Spin until the camera sees the object. The user should be able to set the direction (clockwise or counter-clockwise) and speed of the spin via the GUI.
2. Make the robot point straight to the robot.
3. Calling function(s) from your code for Feature 9, go to the object and pick it up (beeping / tone-making / LED-ing as in your own Feature 9).

Test your portion by adding buttons and entry boxes for the above to **your OWN frame** for this feature, in any way you like.

(continues on the next page)

Feature 11 (No longer required in this project - we leave it here as potential inspiration for someone for sprint 3) : ***Use the* color sensor to follow a dark line*,*** as follows:

For this feature, implement it via ***your own mX files***, NOT in shared files. This is an INDIVIDUAL feature, NOT shared.

Make your robot follow the EDGE of a dark line, as follows:

1. Place your robot so that the color sensor is on the EDGE of a dark line.
2. Have the code measure the original reflected light intensity ***Original***, using the color sensor.
3. Follow the edge of the dark line, as follows:
    - **Person 1:** Use ***bang-bang*** line-following, as described below.
    - **Person 2:** Use the **P of PID control** line-following, as described below.
    - **Person 3:** Use the **P and D of PID control** line-following, as described below.

The user should be able to set the relevant parameters for the type of line-following that is implemented.

Test your portion by adding buttons and entry boxes for the above to **your OWN frame** for this feature, in any way you like. Run on the oval lines on the mats in the room, and also try a more complicated line also in the room.

Bang-bang control means to repeatedly:

- Measure the current reflected light intensity ***Current***.
- If ***Current*** is about the same as ***Original***, go straight.
- Else if ***Current*** is significantly less than ***Original***, veer the appropriate way back toward the line.
- Else veer the other way back toward the line.

The **P of PID control** means to repeatedly:

- Measure the current reflected light intensity ***Current***.
- Let ***Error = Current - Original***.
- Set the left and right wheel speeds to, respectively:

```
B + (Error * K1)
B + (Error * K2)
```

  where one determines effective values for B, K1 and K2 by experimenting on an actual line. Note that K1 and K2 will have opposite signs (one positive, one negative).

The **P and D of PID control** means to repeatedly:

- Measure the current reflected light intensity ***Current***.
- Let ***Error = Current - Original***.
- Let ***Delta = change in error from the previous iteration.***
- Set the left and right wheel speeds to, respectively:

```
B + (Error * K1) + (KD1 * Delta)
B + (Error * K2) + (KD2 * Delta)
```

  where one determines effective values for B, K1, K2, KD1 and KD2 by experimenting on an actual line.

The person implementing the **P and D of PID control** should ask the person implementing the **P of PID control** for good starting values for B, K1 and K2, and use those as your starting point.

For Sprint 3:

First and foremost, we want each of you to be creative, to have fun with this project, and to try some things that you haven't tried yet! So we aren't going to tell you exactly what to do. (But there are certain *types* of things that we'd like **each student** in the class to do, so read on.)

Each of you will write your own code in your own files (ones with m1, m2, etc in the name) - **this sprint is an individual effort**. However, you'll probably want to add more stuff to the *shared* modules (like rosebot), so you'll need to coordinate what goes in there with your teammates.

Passing grade (C grade)

- **Remote control**: Must be able to remote-control (teleoperate) the robot from a GUI on the laptop.
- **TKinter**: Must have a nice-looking TKinter GUI of your own (NOT just the shared GUI you developed in Sprints 1 and 2) with several buttons, entry boxes, text, etc.
- **Sensors**: Must use at least two sensors (at least one of which must be analog) in some way that is different from that of the mini-applications.
- **Motors**: The robot must react in some interesting, challenging way that depends on its environment, per its sensors (NOT just remote control).
- **Team Library**: Your code needs to rely on your team's shared Robot library in some way.
- **Code quality**: Your code is reasonable quality: good choices for names, brief doc-strings for each function/method, did ***Code ~ Reformat Code*** to final version, does not leave behind spurious warning messages, etc.

Excellent (A grade) Same as for the C version, with the following ***additional*** qualities:

- **MQTT**: Must use an mqtt client to send information both from the robot to the PC **and** from the PC to the robot. This info must be integrated into the rest of the program in an interesting way, not just as an add-on feature, and the remote-control does NOT satisfy this item (you need **additional** info to be sent to the robot).
- **TKinter**: Must have a nice-looking TKinter GUI of your own (NOT just the shared GUI you developed in Sprints 1 and 2) with several buttons, entry boxes, text, etc. Includes some type(s) of widget that you didn't learn about in the video to do something interesting.
- **Sensors**: Must use at least three in interesting, challenging ways that fit the project theme (not just an artificial add-on).
- **Creativity**: Your project should be fun and interesting. One potential way to check this box is to have a theme for your project. For example, "My project is a PACMAN game" or "My robot is a firefighter" or "My robot plays Candy Land", then explain how you are using that theme in your project.
- **Complexity**: The functions that you add should be combined in some new way, beyond what you did on Sprints 1 and 2. (In other words, just running *drive_until*

and *beep* in sequence isn't new, so wouldn't meet this requirement.)

- **Code quality**: In addition to the qualities required for the C grade, must also use functional decomposition, with methods and functions no longer than 15 lines and with at least 7 functions or methods in your individual code.

An A project would either include all of these features or all but one of these features with one exceptional feature. A B project does some reasonable amount of the A requirements.

How you turn in your work:

Prior to the beginning of your final exam time:

1. **Commit and push your code.**
2. **Make a video that demonstrates your application,** as follows:
    - Make it using someone's smartphone. **Keep it to 6 minutes or less (and preferably 4 minutes or less if you can).** Get a teammate or friend to help you.
    - Video yourself at the start and say your name as an introduction.
    - State your project goals succinctly and clearly. Make your robot theme clear.
    - Show your Tkinter GUI. It doesn't have to be long, just make it take up most of the frame for a few seconds. If you have pages pop up later, then show them when they pop up.
    - Demo what your robot does, explaining as the robot does so. Be sure to indicate what sensor(s) it is using, when, for what purpose.
    - At the end of the video, ***very*** briefly go through the bullets under Passing Grade and Excellent Grade above, and for each, indicate what your project did to satisfy that one (or indicate that it did not satisfy that one).
    - See any of the following from previous years for examples (but do NOT just repeat what they did, of course!)
        1. **https://youtu.be/3eD4w6C25Xw (path-tracing)**
        2. **https://youtu.be/4Yz6Au0rd-0 (robo-dog)**
        3. **https://youtu.be/TkqSA9_crPI (indiana jones)**
        4. **https://youtu.be/D5D1My0vnbY (pacman)**
        5. **https://youtu.be/I9r8muCzIrQ (broken)**
        6. **https://youtu.be/9sVpXiT5di0** (robo tracker)
3. **Upload your video to YouTube** and mark it as Unlisted (Public is ok to if you like, just **DON'T make it Private**).
4. In the **Team Spreadsheet**, put a link to your video, your robot theme (like PACMAN, Firefighter, Candy Land, etc) and any comments you'd like.