# Capstone Team Project

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_XX*** and ***m0_XX***), so you'll need to coordinate what goes in there with your teammates.

**Passing grade (C grade)**

- **Remote control and MQTT**: 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 Part 1) with several buttons, entry boxes, text, etc.

  EVERY robot action must be able to be initiated by a Tkinter button or the equivalent.
- **Sensors**: Must use at least two sensors (at least one of which must be analog) in some way that is different from that of Part 1.
- **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 code 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**: Your nice-looking TKinter GUI of your own includes some type(s) of widget(s) 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 Part 1, that involves some genuine intellectual complexity. As an example:
    - Moving a set distance does NOT have the required intellectual complexity.
    - Spinning until the camera sees a desired object, then moving to the object (stopping via the infrared distance sensor) has some intellectual complexity.
    - Additionally picking up the object and taking it to a colored area (recognized by the color sensor) has adequate intellectual complexity.

  Another example: Following a curvy black line using PID control would have the required intellectual complexity.

  Another example: Typing notes that play a song is fun but NOT of adequate intellectual complexity. Making the computer compose a song by choosing random notes is getting closer. Making the song have some qualities that real songs might have (see David Mutchler for details) would achieve adequate intellectual complexity.

  Another example: Having your robot draw its motions would have adequate intellectual complexity.

  Another example: Getting your robot to tweet (as in twitter) its motions, combined with something else, would probably have adequate intellectual complexity.
- **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.
- **Code process:** You did frequent, regular commit and pushes, each with a short Commit message that is meaningful to teammates. You avoiding pushing broken code (that possibly breaks your teammates 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 [due date to be determined, about Sunday before exams, maybe a day or two later]:**

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.

## Additional ideas:

Here are some additional ideas for things that you might include in your project:

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

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, do something like one of these:

- 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.
- 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.
- 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.

***Use the* camera *to find an object, then go to it and pick it up as in the previous idea,*** as follows:

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).

***Use the* color sensor to follow a dark line,** as follows:

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.

**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.