Capstone Team Project: Grading Rubric and How to Submit Your Project

Winter term, 2019-2020

Professor Alangar's and Professor Mutchler's sections

Structure of your Capstone Team Project:

Your Capstone Team Project has two parts:

1. ***Team Part:*** You implement this as a team, but with the work divided up and each person contributing their fair share.
2. ***Individual Part:*** You implement this yourself (NOT as a team), with your code USING (calling) code from the Team Part.

For your individual part, **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 we do supply a Rubric (see below) that specifies the qualities that an A-level project requires.

But ***there are certain types of things that we'd like each student in the class to do, so read on.***

Doing an exceptional job on one or more of the features in the Rubric (below) can mitigate a weak feature or two.

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/9sVpXiT5di0 (robo tracker)

Continues on the next page:

How to turn in your Capstone Team Project:

Due date: The end of the Final Exam time scheduled for this course. This term that is **5 p.m. Thursday of exam week**. ***It is OK to turn the project in early.*** Prior to the due date:

1. **Commit and push your code.** But if doing so will break or overwrite a teammate's project, do NOT commit-and-push, and instead get David Mutchler's help in doing so.
2. Put your code into the ***Drop Box*** that is on Moodle, per the description on that Drop Box, answering the questions in the short quiz associated with that Drop Box, including the question where you state the URL of the video that you make and post to YouTube (see next item).
3. ***Make a video*** that demonstrates your application and ***upload your video to YouTube***, as follows:

a. Make it using someone's smartphone. Keep it to 10 minutes or less, ***preferably 5 minutes or less***. Get a teammate or friend to help you take the video.

b. At the start of the video:

- Video yourself and say your name.
- State your project theme, succinctly and clearly.
- Summarize, succinctly and clearly, what your robot will do to accomplish your project theme.

c. Then show your Tkinter GUI, letting the video stay on it for about 5 seconds. No explanation of it is needed yet.

- If your GUI later displays new information or pops up new windows, show the new information or pop-p window(s) for 5 seconds when they appear.

d. Demo what your robot does, explaining as you go and indicating what sensor(s) it is using, when, for what purpose.

if the robot fails to work as desired in the demo because a sensor reading is flawed/noisy, that is OK. You should ***PRINT all sensor readings*** in the SSH window. If a flawed/noisy sensor reading causes the robot to behave wrongly, show the printed sensor value in your video (pausing there for 5 seconds or so), explain that such a sensor reading does not (or does!) happen ordinarily, and move the robot or objects by hand as needed to continue your demo.

e. At the end of the video, very briefly go through ***each*** of the **8 items in the *Basic* Rubric** and the **6 items in the *Advanced* Rubric** of your Individual Part of the Capstone Team Project. (See below for these rubrics and their items.)

For each rubric item, indicate **what your project did to satisfy that item**, or indicate that it did not satisfy that item. Be especially thorough with the *Intellectual Complexity* item.

Then indicate what letter grade you think that you deserve on the project, per the rubric. If you think that the rubric does not adequately reflect the quality of your project, explain why.

f. **Upload your video to YouTube** and mark it as *Unlisted* (or *Public* if you prefer). **Important: Make sure that it is NOT private!**

Grading Rubric

(What You Should Do in your project):

Your grade on the Capstone Team Project is computed as follows:

1. **40 points, Team Part:** Did your team successfully complete and demonstrate success at the Team Part of your Capstone Project? In particular, did you get checked off on Labs 1a, 1b, 2a, and 2b? (These required completing the DriveSystem, Touch, and ArmAndClaw classes in your libs folder.)

   Your score on this part is 40 of 40 if you did so, with the following exception: If you personally did not contribute your fair share to the Team Part of your Capstone Project, then you must do Labs 3 and 4 to earn these 40 points.

2. ***30 points, Individual Part, Basic:*** Does your Individual Part of your Capstone Team Project have the following qualities?

   a. ***Theme:*** Do you have a well-defined theme? Does your code do interesting things with the robot per that theme?

   b. ***Graphical User Interface (GUI):*** Do you have a nice-looking Tkinter/ttk GUI that runs on your laptop? Does it have several buttons, entry boxes and labels?

      Note: Everything that your robot does must be initiated by a Tkinter button or the equivalent.

   c. ***MQTT:*** Do the buttons, entry boxes, etc. on your laptop send information to the robot and cause actions to occur on the robot (by using MQTT)?

   d. ***Teleoperation:*** Can you teleoperate your robot from your GUI or by using the Beacon as a Remote Control (your choice)?

      Here, ***teleoperation*** means that the user can make the robot ***start*** going forward, backward, spinning left (counter-clockwise) or spinning right (clockwise), as well as make the robot stop. This can be accomplished by

      - pressing buttons on the Beacon, or
      - pressing buttons or keys on the GUI, or
      - using a joystick, or
      - moving your hands in the air,
      - or other ways as well.

   (Yes, we have a sensor that can distinguish hand motions!) Your choice; do more than one way to teleoperate if you like!

   e. ***Motion:*** Does your robot ***move*** and ***use its arm-and-claw*** in ways that are meaningful in the context of your theme?

   f. ***Sensors:*** Does your robot react to its environment in some interesting, challenging way, using its sensors to do so? Is the reaction meaningful in the context of your theme?

      Doing just teleoperation is NOT enough. You must use:

      - the wheel encoders, ***and***
      - the touch sensor, ***and***
      - at least one more sensor, ***and***
      - either yet another sensor ***or*** LEDs ***or*** sounds.

      Important: For all the sensors, **if the robot fails to work as desired because a sensor reading is flawed/noisy, that is**

OK. ***PRINT all sensor readings*** in the SSH window so that you can know whether the problem is a flawed/noisy sensor reading.

g. ***Team library:*** Does your code rely on the code in the ***libs*** folder, including the portions that the team wrote in the Team Part of the project?

h. ***Code quality:*** Is your code of reasonable quality? In particular:

- Are the names of your functions, classes, methods and variables meaningful and well-chosen?
- Do you have a brief doc-string for every function and method?
- Did you do **Code ~ Reformat Code** for the final version (to meet some of the PEP 8 standards)?
- Does your code use white space appropriately? (Never more than 2 blank lines between code, and always at least 1 blank line between function and method definitions.)

i. ***Code process:*** Did you do ***frequent***, regular commit and pushes, each with a short ***Commit message*** that is meaningful to teammates. Did you avoiding pushing broken code (that possibly breaks your teammates code)?

3. ***30 points, Individual Part, Advanced:*** Does your Individual Part of your Capstone Team Project have the following additional qualities?

a. ***MQTT:*** Do you use an MQTT client to send meaningful information (in the context of your theme) ***FROM*** the ***robot TO*** the ***laptop***? Does that information get printed or (better) displayed on your GUI?

b. ***Graphical User Interface (GUI):*** Does your Tkinter/ttk GUI use some type(s) of widgets beyond labels, entry boxes, and buttons in ways that are meaningful to your project?

c. ***Sensors:*** Does your robot use ***at least one more sensor than that required for the Basic part*** of your Individual Project? Does your use of sensors include at least one use that is in the context of challenging code? (See next item.)

d. ***Complexity:*** Does your code combine in some new way, beyond what you did in the labs, that involves some genuine intellectual complexity?

As examples of intellectual complexity:

- Moving a specified distance does ***NOT*** have the required intellectual complexity, since it can be achieved simply by calling a method in the library.
- Spinning until the camera sees a desired object, then moving to the object (stopping via the infrared proximity sensor) has SOME intellectual complexity. If you *additionally* have the robot pick up the object and take it to some area recognized by the color sensor, that definitely has adequate intellectual complexity.
- Following a black line via a simple algorithm like the bang-bang algorithm demonstrated in class has SOME intellectual complexity. If you use a more sophisticated

line-following algorithm like PID control, that has adequate intellectual complexity. (See the end of this document for notes about PID control.)

- 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.

  Or, reading the notes from a file on the internet and playing them on your robot might have adequate intellectual complexity, depending on details.

- Drawing, on the GUI, the path that the robot is moving has adequate intellectual complexity, especially if at least some of the information for the drawing is via information sent from the robot to the laptop.

- Having your robot tweet (as in twitter) its motions, combined with something else, would probably have adequate intellectual complexity.

  Likewise, having your robot communicate with the Internet (through your laptop) to do something interesting would probably have adequate intellectual complexity.

- Using a novel device like the Leap Motion sensor (for detecting hand motions) would probably have adequate intellectual complexity.

Be creative!!!!

Do something that you find fun and interesting!

Don't fail to do something that is INTERESTING to you just because you are concerned that it might not have adequate intellectual complexity. Instead, ASK if you are unsure whether something has adequate intellectual complexity. Just about ANY creative idea can be made to have adequate intellectual complexity – talk to David Mutchler!

e. ***Code quality:*** In addition to the qualities required for the Basic part of your intellectual project, does your code use *function decomposition* in meaningful ways:

   - Methods and functions are no longer than 15 lines of code?
   - Avoids *duplicated* code by using functions and methods with *parameters* (and calling those functions and methods with different arguments at different places in the code)?

Doing an exceptional job on one or more of the above features can mitigate a weak feature or two.

Additional ideas

Additional ideas for things that you might include in your project:

1. ***Go to an object*** *using the* proximity (or, better, the optional ultrasonic) *sensor and* ***pick the object up****, doing something interesting as it does so, 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*** or optional ***ultrasonic*** sensor, and pick it up. While the robot is going, do something like one or more of these:

   - The robot should beep as it moves, beeping increasingly faster as the robot gets closer to the object.
   - The robot should make tones as it moves, with the tones increasing in frequency as the robot gets closer to the object.
   - 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.

   In each of the above, the user should be able to set the initial and rate of increase of the actions via the GUI.

2. *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:

   a. Make the robot 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.

   b. Make the robot continue spinning until it is pointing straight to the object.

   c. Calling function(s) from your code for the previous item: go to the object and pick it up (beeping / tone-making / LED-ing as in the previous item).

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

   Make your robot follow the EDGE of a dark line using ***bang-bang control*** as follows:

   a. Place your robot so that the color sensor is on the EDGE of a dark line. Have the code measure the ***original*** reflected light intensity at that position (before the robot starts moving). Let's call that reading ***O*** in the following.

   b. ***Repeatedly***:

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

   Stop when the human presses the Touch Sensor.

   Or, even better, make your robot follow the EDGE of a dark line using the ***P*** of ***PID-control*** as follows:

a. As in bang-bang, place your robot so that the color sensor is on the EDGE of a dark line and have the code measure the ***original*** reflected light intensity, which I will call ***O*** in the following.

b. ***Repeatedly***:

- Measure the ***current*** reflected light intensity ***C***.
- Let `Error = C – O`
- Set the left and right wheel speeds to, respectively:

```
B + (Error * KP1)
B + (Error * KP2)
```

where one determines effective values for ***B***, ***KP1*** and ***KP2*** by experimenting on an actual line. Note that ***KP1*** and ***KP2*** will have opposite signs (one positive, one negative).

Or, still better, add ***D*** and ***I*** to the ***PID*** control, like this:

In addition to doing P-control, also compute, each time through the loop:

- Let ***`Delta = change in error`*** from the previous iteration.
- Let ***`Sum = sum of errors`*** from the previous iterations (perhaps weighting more recent iterations more highly).

Then:

- Set the left and right wheel speeds to, respectively:

```
B + (Error * KP1) + (Delta * KD1) + (Sum * KI1)
B + (Error * KP2) + (Delta * KD2) + (Sum * KI2)
```

where all the constants are determined empirically.

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