Overall Impressions (based on the demo video and submitted code)

Note: Our evaluation is based on what you successfully accomplished in the project, not the idea you originally had (in case you pursued a different idea or were not able to meet all of your milestones).

● Excellent (31 - 40): the project is appropriately challenging for the level of CMPUT 275 and exceeds expectations. At least two challenging aspects of the project are described in the video. Non-trivial algorithms, data structures, communication protocols, and/or object oriented programming concepts make up a significant portion of the code base.
● Good (15 - 30): The project is appropriately challenging for the level of CMPUT 275 and meets expectations. One challenging aspect of the project is described in the video. At least one non-trivial algorithm, data structure, communication protocol, or object oriented programming concept is used in this project.
● Poor to Insufficient (0 - 14): The project is too simple for the level of CMPUT 275 and does not meet expectations. Projects that fall into this category lack complexity. They either did not implement any interesting algorithms, data structures, or OOP concepts, did so in a way that was clearly trivial, or added meaningless complexity (e.g., using Dijkstra's to find straight line distances).

Modularity and Design

● Excellent (16 - 20): The code base is clearly organized and the file and/or directory structure is intuitive. Students always use functions, classes, structs, and/or other data structures appropriately. No section of the code is messy or unclear.
● Good (10 - 15): The code base shows effort towards organization and the file and/or directory structure is clear after some searching. Students mostly use functions, classes, structs, and/or other data structures appropriately. A small portion of the code is messy or unclear.
● Poor to Insufficient (0 - 9): The code base shows minimal effort towards organization and the file and/or directory structure is poor or absent. Students sometimes use functions, classes, structs, and/or other data structures appropriately. A large portion of the code is messy or unclear.

Code Style

● Excellent (16 - 20): Code is fully documented, with file headers in every file and comments describing the general purpose of every function. Comments are meaningful and describe the purpose of the code. Variable and function names are intuitive.
● Good (10 - 15): Code is mostly documented (or excessively over-documented), with file headers in almost every file and comments describing the general purpose of almost every function. Comments are mostly meaningful, but some are trivial or do not add to code.
understanding (e.g., "the following line adds one to x"). Variable and function names are reasonable.
● Poor (0 - 9): Code is under-documented and missing many file headers and/or function comments. Comments are mostly trivial and do not add to code understanding. Variable and function names are poor and make the code hard to read.

README, Running Instructions, Makefile

● Excellent (16 - 20): README contains full running instructions, assumptions, and either Makefile targets (for desktop C++ submissions only) or wiring instructions (for Arduino submissions only). All required components are present, and any extra components (explanation of file structure, photos showing setup, etc.) are complete and helpful. The README is thorough, easy to navigate, and clearly shows student effort.

- Good (10 - 15): README may be missing one or two of the above required components, but is generally complete. The README is sufficient, navigable, and shows student effort.
- Poor (0 - 9): README is missing many of the required components, or the components are incomplete or unhelpful. The README is too short, overly messy, or rambles excessively. It shows little student effort.