Team Embedded Code Review Worksheet (PD4)

Team Identifier: 2\_C

Team Name (optional):

Date:

Team Members Participating in the Code Review: Koby Fowler, Brock Dykhuis, Charles Zulk, Evan Dunn

**Submit your document as a PDF file in Canvas under the corresponding project assignment.**

Instructions for this assignment:

There are two code review checklists provided (files #3 and #4). As a team, select one of the checklist files for this assignment.

A checklist includes lists of rules that embedded code should adhere to. A particular checklist groups the rules into subsets of rules. A subset of rules is sometimes referred to as a perspective. For this assignment, the subsets are shown in the checklist in each file (file #3 has subsets labeled Reviewer #, and file #4 has subsets labeled Function, Style, etc.).

Each student on a team should select one of the subsets in the team's checklist (e.g., a student might select the Reviewer #1 subset or the Style subset). Then a student picks 2-3 rules listed in their subset. This keeps the workload reasonable for this mock review. Students on the same team may select the same subset if there are more students than subsets in a checklist. However, it would be useful to pick different rules.

Each student keeps their chosen rules in mind when reviewing the team’s code (whatever lines of code were selected). A student should write down any issues found in the code in relation to the rules they are responsible for. You don’t need to improve the code. You are simply reviewing code with particular rules in mind and identifying potential issues.

Note: If you are unsure about what a rule means, write down what you think it means and apply the rule as you defined it.

**Select one of these checklists for the code review:**

\_\_X\_\_ Peer Review Checklist: Embedded C Code (P. Koopman, ECE CMU, 2018)

\_\_\_\_ Embedded System Code Review Checklist (G. Khattak & P. Koopman, July 2012)

These checklists are items 3 and 4 in the Code Review resources for this assignment:

* 1-Better Embedded System SW-Embedded System Code Review Checklist.pdf
* 2-PeerReview-selected-slides-CMU.pdf
* 3-peer\_review\_checklist-CMU.pdf
* 4-code\_review\_checklist\_v1\_010.pdf

1. Select 30-40 lines of code developed for your project.

|  |
| --- |
| if(g == 'm'){ |
|  | int object\_counter = 0; |
|  | object objects[50]; |
|  | const int distance\_threshold = 50; |
|  |  |
|  | int same\_distance = 0; |
|  | int distance = 0; |
|  |  |
|  |  |
|  |  |
|  | cyBOT\_Scan\_t info[181] = { }; |
|  | char line[48];//print format |
|  | int z = 0; |
|  |  |
|  | for(z = 0; z < 48; z++) line[z] = 0; |
|  | sprintf(line, "%-10s %-15s\n\r", "Degrees", "Distance"); |
|  | uart\_sendStr(line); |
|  |  |
|  |  |
|  | info[0].ir = adc\_read(); |
|  | info[0].distance = ping\_getDistance(); |
|  | info[0].angle = i; |
|  |  |
|  | int previous\_ir = info[0].ir; |
|  |  |
|  | i++; |
|  |  |
|  | while (i <= 180) { |
|  | timer\_waitMillis(200); |
|  | servo\_move(i); |
|  |  |
|  | info[i].ir = adc\_read(); |
|  | info[i].distance = ping\_getDistance(); |
|  | info[i].angle = i; |
|  |  |
|  | double period = START\_TIME - END\_TIME; |
|  | count += 1; |
|  |  |
|  | val = 195.52 - (27 \*pow((double)val, 0.246)); |
|  | sprintf(str, "%u", reading); |
|  |  |
|  | // sprintf(str, "\n\r%d\t%f\t%d\n\r", info[i].angle, info[i].distance, info[i].ir); |
|  | // uart\_sendStr(str); |
|  |  |
|  | distance = info[i].ir; |
|  |  |
|  | if(distance > 800) { |
|  | if(distance <= previous\_ir + distance\_threshold && distance >= previous\_ir - distance\_threshold) { |
|  | same\_distance += 1; |
|  |  |
|  |  |
|  | } |
|  | else { |
|  | if(same\_distance > 2) { |
|  | sprintf(line, "%d %-10s\n\r", i - 1, "+ object"); |
|  | uart\_sendStr(line); |
|  |  |
|  | sprintf(line, "%-10d %-10s\n\r", same\_distance, "degree [width]"); |
|  | uart\_sendStr(line); |
|  |  |
|  | objects[object\_counter].degree = i - (same\_distance / 2); |
|  | objects[object\_counter].distance = info[i - (same\_distance / 2)].ir; |
|  | objects[object\_counter].width = (objects[object\_counter].distance \* tan(same\_distance \* 3.14/180));//sin(0.0174533 \* same\_distance) \* objects[object\_counter].distance; |
|  | object\_counter++; |
|  |  |
|  | } |
|  | same\_distance = 0; |
|  |  |
|  | } |
|  |  |
|  | } |
|  | else if(distance <= 800) { |
|  | if(same\_distance > 2) { |
|  | sprintf(line, "%d %-10s\n\r", i - (same\_distance / 2), "[+] object"); |
|  | uart\_sendStr(line); |
|  |  |
|  | sprintf(line, "%-10d %-10s\n\r", same\_distance, "[+] degree [width]"); |
|  | uart\_sendStr(line); |
|  |  |
|  | objects[object\_counter].degree = i - (same\_distance / 2); |
|  | objects[object\_counter].distance = info[i - (same\_distance / 2)].ir; |
|  | objects[object\_counter].width = (objects[object\_counter].distance \* tan(same\_distance \* 3.14/180));//sin(0.0174533 \* same\_distance) \* objects[object\_counter].distance; |
|  | object\_counter++; |
|  |  |
|  | } |
|  | same\_distance = 0; |
|  |  |
|  | } |
|  | previous\_ir = distance; |
|  |  |
|  | i+=1; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. From your selected checklist, identify *perspectives* to be taken by team members. Refer to items 1 and 2 in the Code Review resources to learn about perspectives.

Control flow issues - Brock

Data handling issues - Charles

Style issues - Koby

Data parsing issues - Evan

1. From your selected checklist, identify several specific rules to be applied during the code review. Each team member should **choose two or three rules** and focus on those given the limited time available for a short mock review session.

Koby

-Commenting:  top of file, start of function, code that needs an explanation

-Style is consistent and follows style guidelines

-Proper modularity, module size, use of .h files and #includes

Charles

-No orphans (redundant, dead, commented out, unused code & variables)

-Conditional expressions evaluate to a boolean value; no assignments

-Parentheses used to avoid operator precedence confusion

-All switch statements have a default clause; preferably an error trap

Brock

-Minimum scope for all functions and variables; essentially no globals

-Concurrency issues? (locking, volatile keyword, minimize blocking time)

-Input parameter checking is done (style, completeness)

-Error handling for function returns is appropriate

Evan

-Use const and inline instead of #define; minimize conditional compilation

-Avoid use of magic numbers (constant values embedded in code)

-Use strong typing (includes: sized types, structs for coupled data, const)

-Variables have well chosen names and are initialized at definition

1. Record any issues found related to the specific rules considered.

No issues were found, all rules were helpful during reviewing our code.