ARCHITENTERPRISES

Team 20

Lab number 7

Inspection

April 15th, 2013

Version 1.0 – Phase 1

Version 2.0 – Phase 2

Version 3.0 – Phase 3

Version 4.0 – Review

By signing below, each group member approves of this document and contributed fairly to its completion.

Raymond Tang, Andrew McMillion, Archit Rupakhetee, Tyler Lenig

On our honors, as students of the University of Virginia, we have neither given nor received unauthorized aid on this assignment.

Raymond Tang, Andrew McMillion, Archit Rupakhetee, Tyler Lenig

ARCHITENTERPRISES

Contents

Inspection Schedule

We decided to spread out the entirety of our inspection activity across the two week period. This schedule is meant to show the main milestones we had for our prototype and when each was completed by.

April 15th

Generate a checklist for each phase in the inspection

Generate a schedule for completing the inspection

Ensure which group members will complete each part of the inspection

Ensure that Phase 1 of the inspection is completed

April 17th

Swap results from Phase 1 with the other team

Begin to complete the rework of our code from the Phase 2 results

Begin to complete Phase 2 of the inspection

April 21st

Swap results from Phase 2 with the other team

Begin to complete the rework of our code from the Phase 2 results

April 25th

Complete the Phase 3 inspection of our partner group's code

Exchange the results of the last phase with our group

Group Member Assignments

Each group member played a crucial role in the completion of this inspection. Throughout the inspection we used both collaboration and solo work to complete it. Below are the lists of what each member completed individually but we all worked together to compile the inspection checklists, the schedule and the Phase 3 inspection.

Tyler Lenig completed the Phase 1 inspection and the compilation of all of the required portions of the inspection report.

Archit Rupakhetee complete the rework of our code after the results of the Phase 3 inspection were provided to us by our partner team.

Raymond Tang completed the Phase 2 inspection.

Andrew McMillion completed the rework of our code after the results of the Phase 1 and Phase 2 inspections were provided to us by our partner team.

Inspection Checklist

Phase 1

- Internal documentation
  - Comments
    - Used sparingly
    - Clarify complicated aspects of code
  - Systematically convey information
    - File information at the top, such as authors, date, affiliations
  - Only required libraries are imported
    - Each method uses camel case and describes an action
    - Variables declared in groups; no lone variable declarations
- Layout
  - All methods with similar functions are grouped together
    - Getter/Setters near each other
  - Coherent code - code is in sensible order
  - Spacing is consistent throughout
    - No extraneous whitespace
  - Indentation is consistent

Phase 2

- Coding practices (single inspector)
  - Each class is capitalized
  - Each line of code completes one cohesive action and is not overly long
  - All mathematical operations use parentheses for clarity
  - Comments
    - Document source code
    - Systematically convey information
      - File information at the top, such as authors, date, affiliations
  - Magic numbers
    - Use symbolic constants
      - Includes variables and methods
      - No numbers that are unexplained
  - Meaningful identifiers
    - No use of one letter identifiers
    - Identifiers should help self-comment the code
      - Use LONG descriptive names
      - Camelcase for methods

- Underscore as separator
- Coding techniques
  - All field declared private
    - Thorough getters and setters to access private fields
  - No use of dangerous coding techniques
  - Each class in a separate file

Phase 3

- Meets specification document.
  - Implements required modes, text macros, symbolic constants
  - Contains every error message
- Relational operators correct.
- Correct use of dynamic storage/outside files
- Appropriate class names/design aspects
  - Appropriate method names
- Implements communication protocol fully
  - Sends acks, sensor data and error message to the GUI
  - Receives and processes properly formed messages
  - Handles checksum generation and validation correctly according to communications protocol.
- Has operation capabilities when disconnected from the Base Station
- Has basic movement capabilities
  - Turn, stop, move forward and move backward on key press, stopping on depress.
- Interface handles and fixes run-time errors
  - Inclusion of breakpoints in code
  - Ability to monitor variable values at run-time

Group Member Efforts

Each group member expended the proper amount of effort when completing their required portion of the inspection.

Tyler Lenig utilized his time effectively throughout the lab weeks and completed his portion to the best of his abilities in the appropriate time frame. He spent approximate 5-8 hours on completion his portion.

Archit Rupakhetee utilized his time effectively throughout the lab weeks and completed his portion to the best of his abilities in the appropriate time frame. He spent approximate 3-5 hours on completion his portion.

Raymond Tang utilized his time effectively throughout the lab weeks and completed his portion to the best of his abilities in the appropriate time frame. He spent approximate 3-5 hours on completion his portion.

Andrew McMillion utilized his time effectively throughout the lab weeks and completed his portion to the best of his abilities in the appropriate time frame. He spent approximate 3-5 hours on completion his portion.

Even though each group member put in time on their own, we also spent approximately 5-8 hours working together to complete Phase 3 and other aspects of the inspection.

Implementation Questions

Below is the list of questions provided to us by our partner group.

1) Why does decode message and encode message and implement command use a string ArrayList?
2) Why do we include getPadding() function in Message Handler?
3) What are the arguments passed in the differential pilot for?
4) Why is the differential pilot passed Motor.B and Motor.C but not Motor.A?
5) Why is the setRotateSpeed of the Pilot originally set to 90?
6) What is COMMAND_TYPE_INDEX and why is it initialized to 0?
7) What is DEFAULT_RADIUS and why is it initialized to 90?

Answers to Implementation Questions

Below are the answers to the questions provided by our partner group.

1) It is because of readSensor that allows the sensor to arbitrarily add values.
2) To ensure the message is the correct length by adding 0s.
3) They represent the wheel diameter, the track width, the left motor and the right motor.
4) Because B and C are left and right motors, motor A will be used as an arm for the robot.
5) The move arc capability of our robot reduces the need for turning in place so 90 is the most versatile setting.
6) So it would hit the default noOp() case.
7) So with improper values, the robot would still move in the most general case.

Inspection Results

Phase 1

Internal documentation

- Comments
  - *Comments are used sparingly. One consideration would be to add a few comments to explain what each class is used for and what information it hides.*
  - *Some extraneous comments occur in the Debugger and DebuggerShell classes, consider revision or deletion.*
- Systematically convey information
  - *Each file conveys their information in the appropriate fashion with fields at the top and methods below.*
- Only required libraries are imported
  - *Only libraries utilized by the file are imported.*
- Each method uses camel case and describes an action.
  - *This is completed to the fullest extent.*
- Variables declared in groups; no lone variable declarations.
  - *All variables are declared near their instantiation and use.*

Layout

- All methods with similar functions are grouped together
  - *For MessageHandler, consider grouping all of the decoding methods together. For Debugger, consider grouping all command related methods together. All other classes are complete.*
- Coherent code - code is in sensible order
  - *Apart from grouping similar methods together, all of the code is in a sensible, coherent order.*
- Spacing is consistent throughout
  - *No extraneous whitespace is evident. Spacing is consistent throughout, even from file to file.*
- Indentation is consistent
  - *Indentation is consistent and appropriate throughout.*

Phase 2

All files:

Include header information such as Author, Date, License.

Utilize spaces instead of tabs for code indentation.

Driver:
Line 14-24: Magic numbers should be replaced with symbolic constants indicating that these values are the default values. Otherwise, the identifiers should have updated names indicating the symbolic constant they represent and that they are the default values.

Line 34: DifferentialPilot constructor has magic numbers that should be represented by symbolic constants.
Line 35: setRotateSpeed has magic number that should be represented by symbolic constant.

Line 61: Playing sound has numbers that too should be represented by symbolic constants.

Line 73: New lines should be inserted between logical blocks.

Line 131: boolean travel is unclear what it does, perhaps rename travel to is_traveling

line 131: Return statements are unclear as to their effect. Perhaps return SpeedSet = True or Error = False.

Line 178-212: Use of new line should be inserted between the outer if else statement.

line 238: private boolean stop() {} -- Good practice to only set boolean flag after stop command is issued to reduce errors in context switching or early termination causing incorrect flags.

MessagingHandler:
Line 84: Utilize symbolic constant for 256. -- Repeat for it's usage in other class declaration.

Line 94: Utilize MESSAGE_LENGTH symbolic constant declared at top.

Phase 3

Meets specification document.

Implements required SCR aspects (modes, text macros, etc)

Contains every error message

Every error message and appropriate SCR aspects are included.

Relational operators correct.

All relational operators (multiple instances) are correct.

Correct use of dynamic storage/outside files

No outside files are used and are not required.

ArrayLists are used to ensure that storage of run-time variables is possible (dynamic storage).

Appropriate class names/design aspects

Appropriate method names

All methods and class names are appropriate and describe clear aspects of the system.

Implements communication protocol fully

Sends acks, sensor data and error message to the GUI

Returns acks, error messages and sensor data to the GUI when they occur. Consider adding the ability to send all sensors at one time instead of a separate message for each sensor (when polled for all sensors).

Receives and processes properly formed messages

Properly handles well-formed messages and executes their desired function.

Handles checksum generation and validation correctly according to communications protocol.

Generates the appropriate checksum value and appends to the end of the message to validated messages.

Has operation capabilities when disconnected from the Base Station (autonomous control)

No autonomous control has been implemented. The current assumption is that the robot stops and initiate reconnection procedure when it is disconnected. Consider allowing the robot to operate based on sensor input alone until the connection has been reestablished.

Has basic movement capabilities

Line 40 utilizes motor C. Is this intentional? Line 42 sets Motor.C as well, and swing calls motor C. Is the pilot supposed to be Motor A and B by this?

Turn, stop, move forward and move backward on key press, stopping on depress.

Robot has basic movement capabilities and is able to move forward, backward, turn and stop.

Interface handles and fixes run-time errors

Inclusion of breakpoints in code

Breakpoints were not added yet. Consider adding them, perhaps at the beginning of every method or wherever you see appropriate.

Ability to monitor variable values at run-time

Unable to monitor variable values at runtime.

Rework Results

After receiving ample amount of feedback from our partner group, we spent a majority of the week reworking our code between phases and afterward to ensure that our code is as complete as possible. Although we did split up the reworking of code amongst different group members, specifically Andrew McMillion and Archit Rupakhetee, we all played an integral role in reviewing and ensuring the code was as professional as possible.

In terms of lines of code added, we added approximately 50 lines of code. We also had to modify approximately 200 lines of code.