1. Introduction

This document is meant to be an example to aid learning the software engineering requirements process. Different industries, companies, and universities may use or teach other formats and methods but the general outline should be useful. The document covers the requirements for robot software for the FIRST Tech Challenge (FTC) Ultimate Goal challenge for academic year 2021/2021.

1.1. Scope and Limitations

This document conveys the functional requirements for the Robot Application software and does not discuss the coding conventions or formats. It does not cover coding style guides.

1.2.Teaching Notes

Throughout this document there will be special “teaching notes” indicated by *NOTE:* and red italic text. These are items that would not be included in an actual requirements document but are provide to add context and clarification to reader of this example. Useful information about the “Whys” or “Hows” of requirements are provide in the notes.

2. Reference Documents

NOTE: Reference documents are documents that contain information that the author wants to utilize but not repeat. The documents referenced allow the user to look-up information and verify it is current. It is important to avoid repetition since when versions update multiple places would need to be updated. Missing an update could create a conflict.

2.1. FTC References

Reference	Title	Version
1	Game Manual Part 1 – Remote Events	11/20/2020
2	Game Manual Part 2 – Remote Events	11/20/2020
3	FTC Ultimate Goal Java Code	Version 6.0 or later Maintained in GitHub

2.2.Team References

Reference	Title	Version
4	2020-2021 Robot Controls.pdf	Maintained in GitHub
5	UltimateGoalRobotTeam Jave Code	Maintained in GitHub

NOTE: There can be many reference categories. Here FTC references are items provided by FTC and Team references are documents created by the team.

3. Requirements

NOTE: Many formats can be used for documenting but they will always involve some numerical identifier (paragraph number, requirement number, requirement label) and a SHALL statement. SHALL is the verb that makes a requirement. Shall the wording the conveys to engineers that something MUST be met in a design or program. Typically conventions dictate that only a single requirement (shall statement) may be contained in any numbered requirement identifier.

3.1. Program Requirements

3.1.1.FTC Code

Robot Application software **shall** utilize the FTC Ultimate Goal Java application, libraries and functions provider in Ref[3]. [RA001]

NOTE: As formatting examples this requirement contains both a paragraph number (3.1.1) and a requirement number [RA001]. Both can be used together or either one can be used. Use of the reference tells the user that TeleOp and Autonomous are defined in Reference 1.

3.1.2.Class Methods

Robot Application Classes **shall** provide contain all of the methods that act on each specific Class. [RA002]

3.1.3.Class Variables

Robot Application Classes **shall** maximize use of internal private variables and constants versus external class variables or constants for use in Class methods. [RA003]

NOTE: Sometimes requirements specify an intent that is not specifically measurable. In general, this is not good practice but in cases like this there may not be a better way.

3.1.4.Robot Source Code

Robot Application software **shall** be maintained and updated in the team Java program repository in Ref[5]. [RA007]

NOTE: identifiers [RA007] don’t have to be sequential. They should be but when a document is revised and new requirements are added the identifiers for existing requirements must stay the same and can’t be renumbered so new larger numbers are inserted in the document. Likewise when a requirement is deleted its identifier should not be re-used.

3.2. Robot Level Requirements

3.2.1.Robot Functions

Robot Application software **shall** provide the means to control robot hardware in order to accomplish the following basic functions:

- collect rings
- convey rings to the shooter
- shoot rings
- Pick-up and release the wobble goal
- Identify the field set-up of a single ring, quad ring stack or no rings

All functions are required to support scoring according to Ref[2]. [RA005]

3.2.2.GamePads

Robot Application software **shall** receive inputs from the allowed GamePads described in Ref[1] and implemented in Ref[3] for control during TeleOp. [RA006]

Note: TeleOp contains both the first 1 minute and 30 seconds as well as the final 30 seconds of End Game.

NOTE: Requirements can have their own notes that are a part of the document like above. These notes are to provide information and clarity but they aren’t part of the required content.

3.2.3.Parameters

Robot Application software **shall** provide a means to edit parameters used in the code via the GamePads that enables updated values to be used without rebuilding (recompiling) the application code. [RA008]

3.2.3.1. Parameter Creation

Robot Application Parameters **shall** be created in the source code that builds the compiled application code. [RA009]

3.2.3.2.Parameter Class

Robot Application Parameters **shall** be defined as a unique Class Object that can be instantiated and operated on in other classes, OpModes, or methods. [RA010]

3.2.3.3.Parameter Editing

All Robot Application Parameters **shall** be capable of being edited via a TeleOp OpMode using GamePads. [RA011]

Note: This OpMode will be run prior to starting the game sequence described in Ref[1]. The intent of this OpMode is troubleshooting and editing without rebuilding the code.

NOTE: Requirements can be indented into many, many levels. Indents group related requirements. Note that for parameters HashMap isn’t mentioned. The requirement describes what is needed not how to do it at this level. The Software development team has the flexibility to solve the problem any number off ways. Software Engineers may have to provide detailed code block (Class, Method) level requirements in addition to functional requirements if coders need more direction on how to meet the requirements.

3.3. OpMode Requirements

3.3.1.Game Play

Robot Application software **shall** provide applications for two separate modes of game play operation: TeleOp and Autonomous per Ref[1]. [RA004]

3.3.2.Robot Class Construction

Robot Application software **shall** enable construction of the top-level robot hardware class with selectable subassembly parts within any TeleOp or Autonomous class. [RA012]

3.3.3.TeleOp

Robot Application software **shall** contain at least one complete OpMode Class for providing all TeleOp functions. [RA013]

Note: Multiple OpMode Classes are permitted for troubleshooting or different game play scenarios but only one can be loaded for game play

3.3.4.Autonomous

NOTE: 3.3.4 doesn’t contain a requirement it’s just a heading like 3 and 3.3

3.3.4.1. Starting Locations

Robot Application software **shall** contain at least four complete OpMode Classes for providing the Autonomous functions for each of the four starting locations as listed below:

- Blue Outside
- Blue Inside
- Red Inside
- Red Outside

All locations are per the allowed options in Ref[2]. [RA014]

3.3.4.2. Field Set-Up Conditions

Robot Application software Autonomous OpModes **shall** each contain the code to successfully deliver the Wobble Goal for scoring based on the field set-up conditions as listed below:

- No Rings
- Single Ring
- Quad Ring Stack

All field conditions are per the options in Ref[2]. [RA015]

NOTE: By using this reference to the game manual we don’t need to re-iterate the zone A, B, C locations and which one goes with each ring condition. There’s sufficient information for the user to find those details.

3.3.4.3.Pure Pursuit

Robot Application software Autonomous OpModes **shall** each utilize Pure Pursuit algorithms for driving the robot to the wobble goal zones and to a position to shoot rings. [RA015]

NOTE: Here’s a case where the “how” was specified. In this case the Software Engineer writing the requirements wanted a specific algorithm used because of the efficiency or flexibility it provided. Software Engineers need to carefully consider whether all possible solutions need to include this algorithm when specifying it since it becomes limiting to the software designer.

3.3.4.4.Robot Path Determination

Robot Application software Autonomous OpModes **shall** each utilize a common method to set the Pure Pursuit points based on the ring image recognition for driving the robot to the wobble goal zones. [RA034]

3.3.4.4.1.Path Determination Input

Path Determination method **shall** utilize a String input to select the correct path. [RA035]

3.3.4.4.2.Path Determination Output

Path Determination method **shall** populate an ArrayList variable with PursuitPoints as defined in 3.5.1 for the desired robot path for each field set-up scenario. [RA035]

NOTE: 3.5.1 is a paragraph number that defines the PursuitPoint class, but that hasn’t been fully defined here. Since PursuitPoints are an important object for inputting and outputting information they would be covered in requirements at some level.

3.4. HW Class Requirements

3.4.1.Hardware Class Definition

Robot Application software **shall** provide separate Java classes that construct the robot according to the hardware architecture that is comprised of Drive Train, Collector, Conveyor, Shooter, Wobble Goal Arm, and Image Recognition. [RA016]

NOTE: This requirement identifies that the hardware has certain components and each will be controlled by a separate class. The architecture is something that typically requires HW and SW collaboration to define a structure that is functional for programming, building, and testing. If there was a document that described the architecture or captured the decision to make it this way it could be referenced.

3.4.2.Robot Top-Level Class

Robot Application software **shall** provide a top-level hardware class that constructs the robot hardware from various classes during initialization of OpModes. [RA017]

3.4.2.1.Robot Shutdown

Robot Top-Level Class **shall** provide a public method accessible in all OpModes for shutting down the robot after the OpMode completes or the “Stop” button is pressed on the Driver Station phone. [RA018]

3.4.2.2.Robot Telemetry

Robot Top-Level Class **shall** provide a public method accessible in all OpModes for configuring and outputting telemetry resident in lower level hardware classes to the Driver Station phone. [RA019]

3.4.3.Drive Train Class

Robot Application software **shall** provide a Drive Train class that controls the robot driving motion and orientation on the field as well as navigation during both TeleOp and Autonomous modes. [RA020]

NOTE: Observe that there are general requirements for this class that describe functionality in basic terms and some very detailed items. Depending on conventions for document some items can be combined into fewer requirements.

3.4.3.1.Navigation

Drive Train Class **shall** provide a method for navigating during autonomous utilizing the built-in Inertial Measurement Unit (IMU) from the RevHub. [RA021]

3.4.3.2.Field Orientation

Drive Train Class **shall** conduct navigation with respect to field coordinates as defined below: [RA022]

- Field center location is (0,0)
- Units of distance measurement are inches
- Field Y positive points away from audience
- Field X positive points to the right when viewing the field from the Audience
- Field Theta is positive Counterclockwise
- Field Theta Zero is along Field positive X
- Field Upper Right Corner when viewed from Audience is (72,72)

3.4.3.3.Translation

Drive Train Class **shall** provide a means of driving in translation forward, backward, right, and left. [RA023]

3.4.3.4.Rotation

Drive Train Class **shall** provide a means of rotating either clockwise or counterclockwise about the robot center. [RA024]

3.4.3.5.Combined Motion

Drive Train Class **shall** provide a means of moving in combined translation and rotation motion [RA024]

3.4.3.6.Inputs

Drive Train Class **shall** provide a method for driving in all means as described in 3.4.3 utilizing the GamePad inputs and the GamePad control definitions provided in Ref[4]. [RA025]

3.4.3.7. Autonomous Rotation

Drive Train Class **shall** provide a method to autonomously rotate the robot to any 360 degree orientation. [RA026]

3.4.3.8.Rotation Accuracy

Drive Train Class **shall** provide the commanded orientation to within +/- 1.0 degrees accuracy. [RA027]

NOTE: Accuracy is specified but it may not be able to be met solely by software. This at least defines the IMURotateTol value for exiting the method, but it may require some collaboration with HW to meet.

3.4.3.9.Rotation Input

Drive Train Class Rotation method **shall** receive the commanded angle input in degrees as a type of “double”. [RA028]

NOTE: Requirements should provide very specific details of interfaces (inputs/outputs) because different coders may implement the method using the double and the method or class that calls it.

3.4.4.Collector

Robot Application software **shall** provide a Collector class that enables the robot to collect rings from the field during TeleOp. [RA029]

3.4.5.Conveyor

Robot Application software **shall** provide a Conveyor class that enables the robot to convey rings from the Collector up to the Shooter during TeleOp and Autonomous. [RA030]

3.4.6.Shooter

Robot Application software **shall** provide a Shooter class that enables the robot to shoot rings received from the Conveyor during TeleOp and Autonomous. [RA031]

3.4.7.Wobble Goal Arm (WGA)

Robot Application software **shall** provide a WGA class that enables the robot to grab, raise, and lower the Wobble Goal during TeleOp and Autonomous. [RA032]

3.4.8.Image Recognition

Robot Application software **shall** provide an ImageRecog class that enables the robot to recognize the field settings with single ring, no rings, and quad ring stack during Autonomous. [RA033]

NOTE: Each of the these classes will have detailed requirements like the Drive Train. All of the hardware requirements need

3.5. Utility Class Requirements

3.5.1. Pursuit Points Class

Robot Application software **shall** provide a PursuitPoint class that can contain field location information for use in the Pure Pursuit navigation algorithm and other supporting methods. [RA0XX]

NOTE: Further details would be needed for what the class object is and what it contains and how to construct it so it could be used by others. SW requirements might be divided into multiple documents where functional requirements would be in a top level and then major classes and then requirements for the methods in those classes and finally supporting or utility classes.