# 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 examples of 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 (code to build an .apk for Android Studio) and does not discuss the coding conventions or formats for the source code. It does not cover coding style guides. The document provides an example of the types of requirements and different levels from top level application to OpModes, Classes, and Methods. It is not comprehensive to provide the total set of requirements necessary to develop a complete FTC program capable of scoring all types of points.

## 1.2.Teaching Notes

Since this document is an example, it focuses on providing the different types of requirements and supporting information to help the reader understand why things might be done this way. Additional notes are provided to clarify whether the example given the only valid solution or one of many possible valid options. 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 always have access to the most current information and prevents the author from needless updates. It is important to avoid repetition since when versions exist and an update occurs, 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<br>Maintained in GitHub |

## 2.2.Team References

| Reference | Title | Version |
|---|---|---|
| 4 | 2020-2021 Robot Controls.pdf | Maintained in GitHub |

| Reference | Title | Version |
|---|---|---|
| 5 | UltimateGoalRobotTeam Jave Code | Maintained in GitHub |
| 6 | Robot Team Code Structure *(NOTE: could contain Keynote and Pages files that show code structure diagrams for Classes and Functions)* | Dec. 2020 |
| 7 | Pure Pursuit Algorithm Description Document *(NOTE: doesn’t exist but would include the description of pure pursuit and equations encompassing the PDF with pictures)* | Rev A *(NOTE: often documents use letters for different revisions)* |

*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 statement a requirement vs. just information. “Shall” 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. This supports tracing of requirements between documents and verification of requirements.*

## 3.1. Program Requirements

### 3.1.1.FTC Code

Robot Application software **shall** utilize the FTC Ultimate Goal Java application, libraries and functions provided 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 in Reference 3 they will find the details or actual code for the FTC app.*

### 3.1.2.Class Methods

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

*NOTE: This is more of a code organization requirement than functional requirement, but it’s important for the development of multiple classes. It means that DriveTrain should contain the methods for driving and they should not be defined in an OpMode or separate “driving” class.*

### 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 (how do we assess “maximize” objectively). 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 don’t have to be sequential ([RA007] here following [RA003]). They should be when a document is created 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. This enables the identifiers to always map 1 to 1 to a specific requirement across all versions of the document saving needless rework on documents that reference this one.*

## 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
- Lift the wobble goal above the 12” wall
- Identify the field set-up of a single ring, quad ring stack or no rings

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

*NOTE: Often requirements start out very basic at the top level of the code so they can define the overall functions but then they will get more detailed as one goes into lower levels of code and functional details. Once we have the basic functions we can begin to provide their details.*

### 3.2.2.GamePads

Robot Application software **shall** receive inputs from any of 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 during the 2 minute drive controlled portion of the 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 such as definitions but they aren’t part of the required content of the software.*

### 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]

*NOTE: This is not the only option but this is where our team of Software Engineers define what would be the best and most efficient way to update values during trial and error with the robot. The approach is specified here as a requirement because of a software architecture decision. These SW architecture decisions are key tasks of SW Engineers vs. SW coders who implement them.*

#### 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[2]. 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. Also observe 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 (SEs)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]

*NOTE: Ref[1] is listed to describe to the user where to find out more about TeleOp and Autonomous. The author doesn’t need to repeat all the details here.*

### 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]

*NOTE: Another example of a SE architecture decision being captured as a requirement. There are may ways to construct the robot class code.*

### 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 at a time during 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. Company or Systems Engineering conventions for requirements document formats may dictate whether all dented requirements need a heading or if they can also contain “shalls”.*

#### 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 and per the allowed options in Ref[2]. [RA014]:

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

#### 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 per the PurePursuit Algorithm Description Document Ref[7] 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 based on a functionality decision. In this case the SE writing the requirements wanted a specific approach for driving and a specific algorithm to accomplish it. This might be because of the efficiency or flexibility it provided. The Algorithm Description Document (ADD) would contain all the equations to use for Pure Pursuit. It’s referenced here vs. included to not clutter this document and because other engineers (Control Systems designers perhaps) might develop the algorithms as one part of the total SW that the SE is responsible for.*

#### 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. [RA036]

*NOTE: Path requirements here are examples of details to specify I/O and methods since this information will be shared across Drive Train, OpModes, and utility classes. Similar examples would be needed for lower level requirements for other classes.*

## 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 defined in Ref[6] 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. The reference here is to the documents made that describe the class inheritance and organization of the classes that were made earlier in the year.*

### 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 that takes in the OpMode as an input and returns void which is accessible in all OpModes for configuring and outputting telemetry resident in lower level hardware classes to the Driver Station phone. [RA019]

*NOTE: Depending on the complexity of the project and organization of a company the requirements team, SW design/architecture team, and coding team could all be the same person or different multiple person teams. Writing requirements for each level of the code (classes) allows different people to write code that will be compatible. OpMode coder doesn’t need to know the details of the Drive Train telemetry method just how to call it.*

### 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 or details could be broken out into separate requirements document - one for each class. SW is often described in Configuration Items (CIs) that are the lowest level of source code that can be described as a version controlled unit. For FTC Java classes can definitely meet that definition. CIs (Java classes) might have their own requirements because they can be updated and tracked in GitHub so they can be separate units.*

#### 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)

*NOTE: Some Systems Engineering conventions would not like this list as it defines multiple requirements in a single “shall” and numbered identifier. Others would accept it as it defines a set of related items that all form the definition of Field Coordinates. Still others would write a “Field Coordinates” document to capture all this and then reference it here. Less of a SW issue than a requirements documentation convention.*

#### 3.4.3.3.Driving

##### 3.4.3.3.1.Translation

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

##### 3.4.3.3.2.Rotation

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

##### 3.4.3.3.3.Combined Motion

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

#### 3.4.3.4.Inputs

Drive Train Class **shall** provide a method for driving in all means as described in 3.4.3.3 utilizing the GamePad inputs and the GamePad control definitions provided in Ref[4]. [RA025]

#### 3.4.3.5. Autonomous Rotation

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

#### 3.4.3.6.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.*

#### 3.4.3.7.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 and class or other method 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]

#### 3.4.7.1. Move Wobble Goal Arm Method

WGA class **shall** contain a method named “moveWobbleGoalArm" that moves the Wobble Goal Arm to a target position during Autonomous or TeleOp. [RA037]

##### 3.4.7.1.1.Move Method Inputs

moveWobbleGoalArm method **shall** receive the following input variable types in the order listed:

- BasicOpMode class
- Double for the wobble goal arm target position in degrees
- Double for the allowable accuracy in degrees for the final position to allow exiting the method
- Boolean for indicating whether telemetry is output during the method

[RA038]

##### 3.4.7.1.2.Move Method Outputs

moveWobbleGoalArm method **shall** output a boolean that indicates whether the wobble goal arm has reached a position within the target tolerance. [RA039]

##### 3.4.7.1.3.Drop Method Functions

moveWobbleGoalArm method **shall** provide the following functionality:

- Set the wobble goal arm motor target position based on the input target angle

- Enable the wobble goal arm motor to move to the target position
- Display telemetry from the WobbleArm getTelemetry() method depending on the boolean input
- Display telemetry via the OpMode passed to it
- Calculate the current arm position in degrees
- Compare the current arm position to the target position
- Determine whether the arm is within the accuracy tolerance bounds
- Return a boolean indicating the status of true for arm position within the accuracy bounds and false for arm position not in the accuracy bounds

[RA040]

#### 3.4.7.1.4.Drop Method Usage

moveWobbleGoalArm method **shall** be capable of being called within a loop in either a TeleOp or Autonomous OpMode that may use the boolean output as a means to exit that loop. [RA041]

*NOTE: This moveWobbleGoalArm method set of requirements is an exercise for developing a new code that doesn’t exist. Are the requirements sufficient? Is more information needed?*

### 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.*