Setting up Software for New Season (2024 – Into The Deep edition)

Author: Dennis O'Brien
(dco6453@me.com)

Team: FTC-7462 (Not to Scale), Mentor

This page is intentionally left blank

Table of Contents

This page is intentionally left blank

1 Introduction

The full extent of setting up a new environment can be accessed using the FIRST Tech Challenge Software Development Kit link. However, this site does not mention how to layer on the *Road-Runner* and *Dashboard* software, created/maintained by ACME Robotics (Team #8367). ACME makes the SDK a fully functioning environment for developing a highly competitive robot.

One requirement for the 2024 season is that we must use version 10.0 of the *FtcRobotController* software. So, these instructions detail how to add in the capabilities of *Road-Runner* and *Dashboard*.

This year we have been fortunate enough to be able to host 2 teams broken up into a Junior-Senor team (7462) and a Freshman-Sophomore team (8628). With these instructions, each team can now build 2 separate robots using a single code base.

2 Setting up a New Repository

At the beginning of every season FIRST publishes a new version of the *FtcRobotController* software. A bare bones project can be found on GitHub at FIRST-Tech-Challenge.

However, there is a similar piece of software called *Road-Runner*. *Road-Runner* is derived from this year's version of the *FtcRobotController* software and includes additional software needed to build more robust autonomous solutions.

This software, specifically *road-runner-quickstart*, must be **forked**, not cloned, into the *FIRST-4030* repository using the following steps:

1. Before you begin it is helpful to create a local folder on your PC to hold all of the software, documentation, etc. you will need for the season.
2. From the ACMERobotics GitHub site, select the *road-runner-quickstart* repository.
3. Select the *Create a new fork under* the *Fork* pulldown
4. Under the *Create a new fork*, make sure the *Owner* dropdown is set to *FIRST-4030*
5. Set the *Repository name* to a string that is indicative of the competition year (e.g., FTC-<Year>-*<SeasonName>*-*<Team>*)

Once the repository is completed, in the newly created project on GitHub, you need to add in those persons who can access it using the following steps:

6. Select the *Insights* link at the top of the repository
7. Select the *People* button in the left-hand list of options
8. Add all other developers on the coding team with *Write* access
9. From the *FIRST-4030* repository, clone the new project onto your PC by selecting the *<> Code* button and copy the *HTTPS* URL to the clipboard
10. Within Android Studio, use the newly selected URL, and *Git->Clone…* to clone the repository into a separate folder on your PC.
11. If a popup window appears saying *Project update recommended* then select the *upgrade* link and then select *Begin Upgrade*, followed by the *Run selected steps* button.
12. Refer back to the FIRST-Tech-Challenge and scroll down to the *README/Requirements* for the FtcRobotController to find the minimum level of Android Studio needed for the current year.
13. In Android Studio, open the *About Android Studio* under the *Android Studio* menu dropdown to verify you are using the proper level.

3 Connecting Repository to Discord Server

Now that you have a repository it is useful to establish a webhook from the repository to our Discord *#github* channel. (Establishing a webhook results in a notice to everyone in the *#github* channel being notified whenever there is a change to the repository.)

Before you begin, it is assumed that you have the proper privileges (e.g., webhook, moderation, admin, etc.) in Discord to complete the process. (You will know if you have the proper level of privileges if you **can** complete step #1.)

Use the following steps to establish a webhook between our GitHub repository and Discord:

1. In Discord, right-click on *#github* and select *Edit Channel*. This should display a list of options, in a column, on the left-hand side.
2. Select *Integrations* and then *View Webhooks*.
3. Select the *New Webhook* button.
4. Expand the view of the new webhook and change the name of it to something meaningful to this year's work.
5. Select the *Save Changes* button to save the name change.
6. Select the *Copy Webhook URL* button
7. In GitHub, select the newly created repository
8. Select the *Settings* button at the top of the window
9. Select the *Webhooks* button in the left-hand set of buttons
10. Select the *Add webhook* button
11. Log in to the repository, if requested
12. Paste the string into the *Payload URL* field in GitHub
13. Append the string */github* to the *Payload URL*
14. Set the *Content type* to *application/json*
15. Select the *Send me everything* radio button under the *Which events would you like to trigger this webhook*
16. Select the *Add webhook* button
17. Select the *Starred* option in the upper right-hand page which will send a minimalist message to Discord to show that the webhook is working.

4 Adding Software Compiled Date

It is quite advantageous to push code to your robot via Wi-Fi instead of always having to do so using a hard cable. However, there is always a concern that a software push may not have completed successfully.

Use the following steps to add the capability of displaying the software compiled time on the Driver Station.

1. Open the *build.gradle* file associated with *TeamCode*
2. At the bottom of the *android* block add the following lines:

```
buildFeatures {
    buildConfig = true
}

buildTypes {
    release {
        buildConfigField "String", "COMPILATION_DATE", "\"${new Date()}\""
    }
    debug {
        buildConfigField "String", "COMPILATION_DATE", "\"${new Date()}\""
    }
}
```

3. Do a *Gradle->Sync* of the project

4. In the *init* portion of each *opMode* add the following lines,

```
if (opModeInInit()) {
    telemetry.addData("Compiled on:", BuildConfig.COMPILATION_DATE);
    telemetry.update();
}
```

5. Rebuild your code

   Note: the BuildConfig class is somehow connected to the Android Studio infrastructure. If you do a *Build->Clean Project* then it will remove some underlying infrastructure bytecode resulting in the BuildConfig class to appear broken. **Don't worry about it.** The bytecode will get rebuilt the next time you do a *Build->Make Project*.

5 Adding Prior Seasons Capability

The repository provided by ACME does not include some files that our team has proven to be useful over the years. Use the following steps to migrate some past history into the new repository.

1. In the *teamcode* folder, create a folder call *OpModes*
2. From a prior year's project,
   - Copy the *gamepad* and *math* folders into the *teamcode* folder
   - Copy *GeneralConstants* and *Pose2dWrapper* and paste them into the *teamcode* folder
   - Copy any useful opmodes into the *OpModes* folder
3. From a prior seasons project, copy the ControlHubAuto opmode and add it to the current project's *OpModes* folder.
4. Copy the 2 classes ControlHub and ControHubNames classes and place them in the teamcode folder.

   Note: ControlHub is a class that connects to the Control Hub of the robot and retrieves its MAC Address. From there it then associates it with the name of the network found in the ControlHubNames class.

Only add in any other files that you know would be useful for the upcoming year. (You can always add in files later, if necessary.)

6 Supporting Multiple Robots

All of the software provide by Acme Robotics assumes there is only a single robot. To support multiple robots *MecanumDrive* must be changed.

If you decide to use odometry pods then you must change either the class *TwoDeadWheelLocalizer* or *ThreeDeadWheelLocalizer*, depending on your robot configuration.

6.1 Changing GeneralConstants

In *GeneralConstants*, add the following line (if it is not already there),

```
public static final String PRIMARY_BOT = "xxxx-RC";
```

where, xxx is the team number of the primary robot (e.g., 7462-RC).

6.2 Changing MecanumDrive

In *MecanumDrive*, locate the class called Params at the top of the file and then use the following steps:

1. After the line that instantiates the PARAMS object, add the following 2 lines:
```
ControlHub controlHub = new ControlHub();

public static String networkName;
```
2. At the end of the *MecanumDrive* class, create the following method,
```
private void setParams() {

  if (networkName.equals(PRIMARY_BOT)) {
  } else {
  }
}
```
3. Copy all of the public double values from the *Params* class at the top of *MecanumDrive* and place them in each part of the *if-else* code of the setParams method.
4. Replace the casting from each variable with the string PARAMS.. (Note the "." At the end of PARAMS)
5. In the original *Params* class, remove all value assignments, leaving only their variable declaration.
6. At the end of the *MecanumDrive* class, create the following method,
```
private void initializeOtherParameters() {
}
```
7. Copy the definitions of MecanumKinematics, TurnConstraint, VelConstraint, and AccelConstraint and place them in the newly created initializeOtherParameters method.
8. Remove the casting from each variable in initializeOtherParameters.
9. In the original assignment of these 4 variables, set each variable to null.
10. Throughout the code there may be some in-line assignments associated with hardware differences (i.e., motor/encoder direction, etc.) that are specific to each robot. In these cases, add the following lines to encapsulate those changes,
```
if (networkName.equals(PRIMARY_BOT)) {
  primary hardware setting;
} else {
  secondary hardware setting;
}
```

11. In the first couple of lines of the *MecanumDrive* **constructor**, add the following lines,

```
networkName = controlHub.getNetworkName();

// set PARAMS based upon the network you are connected to
setParams();
```

6.3 Changing the Localizer

If using odometry pods then open the appropriate localizer, and locate the class called Params at the top of the file and then use the following steps:

1. At the end of the appropriate localizer class, create the following method,

```
private void setParams() {

   if (networkName.equals(PRIMARY_BOT)) {
   } else {
   }
}
```

2. Copy all of the public double values from the *Params* class at the top of the localizer and encapsulate them in each part of the *if-else* code of the method.
3. Replace the casting from each variable with the string PARAMS.. (Note the "." At the end of PARAMS)
4. Assign very generic values to each parameter until you can go thru a full calibration.
5. In the original localizer class, replace all value assignments with null.
6. Throughout the code there may be some in-line assignments associated with hardware differences (i.e., motor/encoder direction, etc.) between 2 robots. In these cases, add the following lines to encapsulate those changes,

```
if (networkName.equals(PRIMARY_BOT)) {
   primary hardware setting;
} else {
   secondary hardware setting;
}
```

7 Limiting OpModes During Competition

The Driver Station can become cluttered with all of the calibration opmodes defined by *RoadRunner*. Use the following steps to mask out unnecessary calibration opmodes.

1. In *GeneralConstants*, add the following line (if it is not there already),
   public static final boolean *ENABLE_CALIBRATION* = true;
2. Open TuningOpModes and navigate down to a series of *manager.register* statements.
3. Encapsulate all of these lines, except for *LocalizationTest*, with an *if (ENABLE_CALIBRATION)* construct.

   Note: Preserving access to *LocalizationTest* makes it possible to easily move the robot around the field outside of any team developed opmodes.

With this in place you can toggle the ENABLE_CALIBRATION value to have the calibration opmodes available, or not.

8 Verifying the SDK via Road-Runner

Presumably all of the above work will result in working robot. The only way to know is to download the project onto a robot and calibrate it using *Road-Runner*.

Begin with opening a web browser to the Road-Runner web site. This is a "cookbook" of steps to follow that are pretty self-explanatory.

If this is the first time that you have used *Road-Runner* then select the *Before You Start!* Button and review the terminology and tips.

Select the High Level Overview header in the left-hand column. It will display the steps you will go thru.

The first step in calibrating a robot is to generate Drive Constants. After that, follow the steps in the order shown on the chart.

9 Preserving Changes Back into GitHub

Your local environment now needs to be rolled up into a comprehensive release and pushed back up to the *FIRST-4030* GitHub repository. All of the files in the project that appear in green have changed since the last commit. Those in red are new to the repository.

Within Android Studio, use the following steps to commit your changes to the team wide GitHub repository:

1. In the left-hand column of the project window, right-click on the *TeamCode* folder.
2. From the dropdown menu select *Git -> Add*.
3. From the dropdown menu select *Git -> Commit Directory...*.
4. A new window will be displayed showing all of the files that will be committed to the local *master* repository.
5. Make sure all of the files are selected. (The gradle files may not be selected but they should be.)
6. Add a brief text string in the *Commit Message* window that explains why the commit is being done.
7. Select the *Commit* button. (**DO NOT** select the *Commit and Push…* button since you are presumably working on a local branch and the "push" will not do what you think it should.)
8. Android Studio will analyze the code one last time and display its results. Assuming **no errors** are generated, then push the *Commit Anyway* button.
9. At this point all of your changes have been committed to your local *InstallDashboard* branch. These changes now need to move to your local *master* branch.
10. From the dropdown menu select *Git -> Branches...*.
11. Select the *master* branch and *Checkout*. (You will see the branch name change in the lower right-hand corner of the IDE.)
12. From the dropdown menu select *Git -> Merge...*.
13. Select the *InstallDashboard* branch.
14. Select the *Merge* button.
15. Move all changes up to the team wide repository by selecting *Git -> Push* button.
16. A new window will be displayed listing all files that will be committed. Assuming every looks right select the *Push* button.

10 Troubleshooting

In building this procedure there were a number of errors/issues that arose that took many hours to track down. Here are those errors and possible fixes for them.

Error / Issue:	When building the software the error "Unsupported class file major version 63" occurred.
Solution:	Check all of the supporting libraries to make sure there versions are compatible. That is, comment out each library and substitute in one of a lower version. Be sure to do a Sync Now before compiling.

Error / Issue:	During checkout of the robot the *Start* and *Stop* button do not stay on when you hit the *Init* button.
Solution1:	Check the values in *DriveConstants* for legitimate numbers.
Solution2:	Make sure that the motor declarations in *MecanumDrive* match those in the robot's config file..
Solution3:	Run the opmode using a Driver Station to see if it throws an exception