2018 Robot Guide

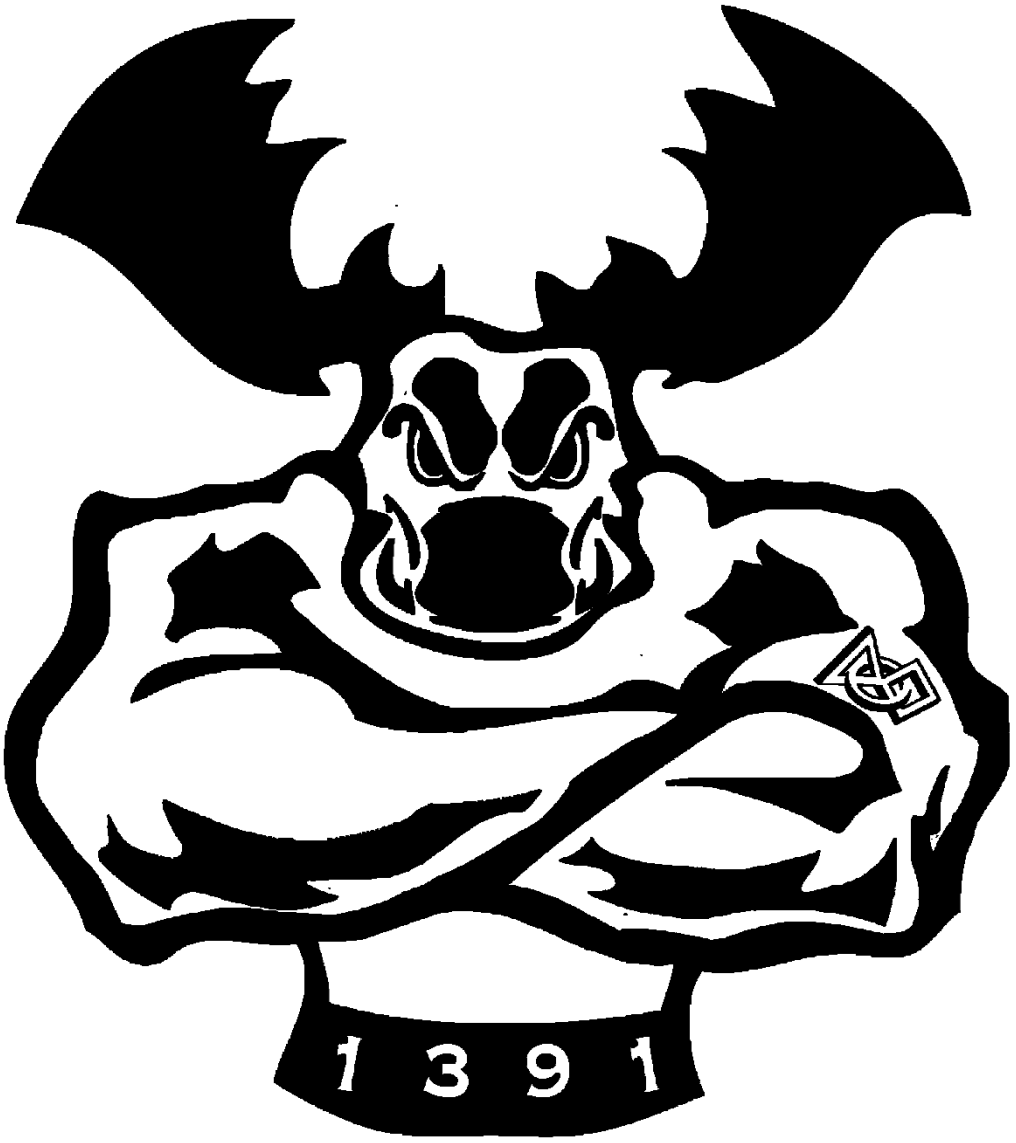


Table of contents

Introduction

This document was written to simplify the usage of the 2018 Metal Moose robot, and to provide documentation and reasoning for the various decisions made when programming, controlling, and designing the robot. It goes over the basics of robot operation (turning on, connecting), driving and operating, programming (mainly where to look for things to change), and other parts of the robot.

How to read

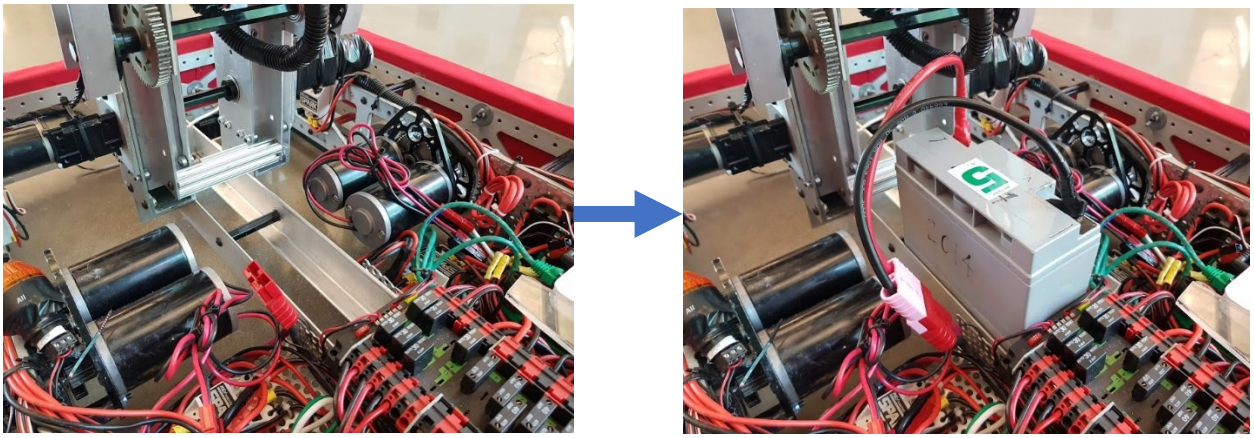
The standard way to read the guide would be from the beginning to end. However, if you feel you have enough knowledge of operating the robot, you can use it as a simple reference guide. One thing that you

should do, even if you feel confident with operating the robot is to **read the notes** that you can find (usually at the ends of the sections) throughout the guide. They are vital for correct robot operation and preventing mistakes that we have made many, many times that we don't want you to make.

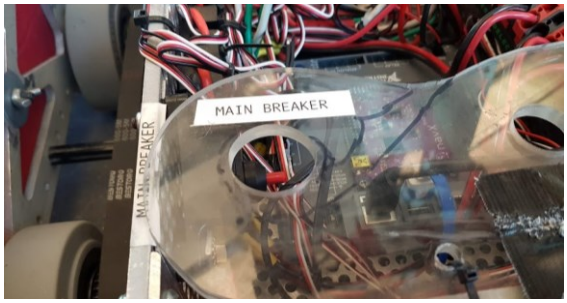
Making the robot come alive

To get the robot working, follow these steps:

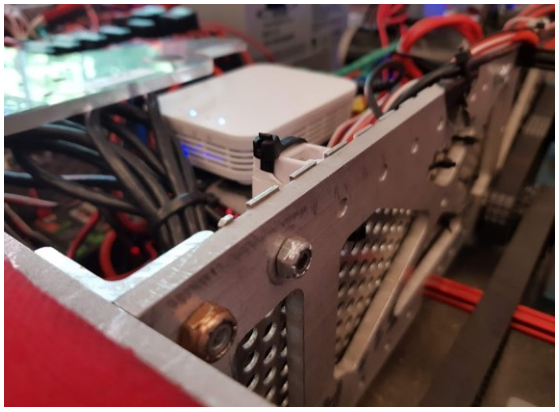
1. **Put in the battery.** The battery goes right in the very center of the robot, in front of the elevator (when looking from the robot's perspective). After making sure that it is properly strapped in place using a double-sided Velcro tape designated for this task, it is time to turn the robot on.



2. **Turn the robot on.** The power switch of the robot is located on the right side of the robot (when looking from the robot's perspective), under a protective plastic ~~spare part that looks like a seahorse~~ professionally designed cover by pressing **the side of the breaker** to connect the wires in the breaker. Pressing the red button will turn the robot off, as it disconnects the wires.



3. **Connect the drive station laptop to the robot.** There are two ways to connect to the robot (if not connecting through a field management system). Either connect to the robot **wifi** "1391" that should appear after turning on the robot and waiting a while, or through **tethering** with an ethernet cable (found on the left side of the robot, as seen on the picture on the right). If the robot isn't connecting or the wifi cannot be seen, refer to the "Problems and Solutions" part of this document.



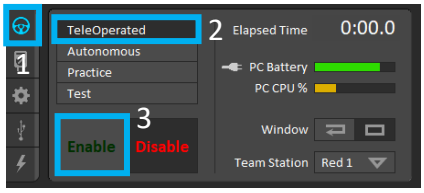
Important note: *due to the networking on the robot, it might take the robot up to 30 seconds for it to fully turn on (and show the 1391 wifi) – be patient!*

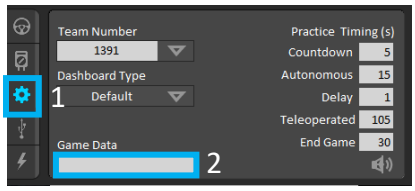
Important note 2: *the Driver Station 1's ethernet port is faulty. That is why the Ethernet to USB-B converter is used instead. Use the ethernet port on the laptop only if necessary, because it might cause disconnects when wiggled around.*

Using the FRC Driver Station

The robot is operated using the **FRC Driver Station**, which can be found on the desktops of either of the driver station laptops.

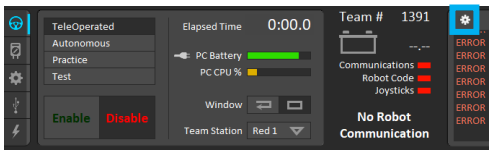
The FRC Driver Station is essentially used to give the robot information about what is happening in the game. To enable either the teleoperated or the autonomous period, click on the appropriate tab in the Driver Station and click **Enable** (as seen on the picture on the right, taking teleoperated as an example).





An important mention for testing the autonomous is altering the game message sent to the robot, because it changes the robot's behavior in autonomous. The message consists of 3 characters (R or L) symbolizing the field layout (RRR, for example, means that both the switches and the scale are on the right side for your alliance).

The driver station can do more cool stuff, like viewing the log files or the console of the RoboRIO (usually to determine what caused problems with the robot, if there were any). Shown on the diagram is the location of the console and the log files.



Important note: *pressing ENTER on the Driver Station laptop will cause the robot to become disabled. Pressing SPACE will cause the robot to become emergency disabled (requiring redeploying the code or restarting the RoboRIO to function again).* ***DO NOT DO EITHER OF THOSE WHEN THE ELEVATOR IS UP*** *and you don't absolutely have to to prevent injury ~~(although I would argue that the robot is more important)~~, because it will come crashing down, which is really bad for the robot and could potentially break something.*

Important note 2: *to correctly execute the desired autonomous sequence, the robot* ***must be restarted*** *before entering the autonomous period. This has to do with how the FRC Driver Station accidentally remembers the previous game data that it sent to the robot, and the way that the robot resets its variables. Blame ~~Tom~~ the programming team for this.*

Operating the robot

Driver Controls

The robot has a multitude of driving controls for either the controller or the joystick. However, the two for the controller are **discontinued**, since they were not used at all throughout the 2018 FRC season. They will still work, but they don't support raising fourbar or switching to backwards mode.

Tank drive functions like the name suggests: each y axis of the two thumbs controls one side of the drivetrain. **Arcade drive** uses only one thumb, and essentially drives the robot to where the thumb is pointing.

The best usage for these two would be to check, whether the next year's driver would be more comfortable with them.

To actually use any of those, you have to change the variable **driveMode** to either 0 (tank drive) or 1 (arcade drive).

A note to be made is that while it hasn't been tested, the arcade drive should work with the Logitech joystick too (although it is mainly designed for the controller).

The main mode of driving used throughout the 2018 seaso is a variant of the joystick arcade drive. The controls can b seen on the diagram on the right. A note to be made is th Fourbar down just needs a single press for the fourbar to down, but the Fourbar up needs to be held (it goes up for t duration of the hold and then it will ~~hopefully~~ stay up).

To choose the desired one, select the appropriate radio button on SmartDashboard **before the teleoperated period begins.**

Operator Controls

The operator controls various aspects of the robot (such as the collector, hanger, and elevator) using a joystick. There are no other driving modes. To rebind the controls, either refer to the programming portion of this document (or use a scripting language like AutoHotkey to rebind the keys on the keyboard if you are confident in your programming abilities – that might be easier).

Pressing the **override button** gives the operator full control over the elevator speed, because normally, it is throttled by a polynomial function to not come crashing down or flying up. This is helpful when the elevator is not functioning properly (likely because of friction, encoder fault…) and we *really* need to win the match. **DO NOT PRESS THIS FOR NORMAL USE**, as the elevator will not be held up and will come crashing down if not held up manually. If you can avoid it, don't use this button at all.

Important note: *To intake, preferably use the* ***Full intake*** *button. The Variable Intake is just there because why not. Full Intake automatically unclamps the collector, allowing easy cube intake (Variable Intake does not).*

Important note 2: *make sure that the light next to the mode button on the controller is not on. If it is, press the mode to turn it off (having it on changes the mode of the controller to one for which the controls are not correctly bound).*

Tips and Tricks

Here are a few things that the drive team noticed when controlling the robot during the 2018 season.

Fourbar doesn't go up when driving backwards too quickly. This is caused by the counter force applied to the fourbar caused by the robot moving in an opposite direction. It's better to raise the fourbar when the robot is driving forwards, not backwards.

Climbing with the robot is a relatively easy task, when the appropriate movements are used. To hook the arm on the rung, the robot must drive backwards to the left side of the platform. From there, the operator extends the arm and raises the elevator. Once elevating the elevator with the arm raised hits the bottom of the rung, the driver (while pressing the reverse direction button) drives backwards and to the right at the same time, then forward and to the left (and repeat this motion a few times). Once in, the operator can either start winching, or simply elevate down (when in time trouble – this will cause the robot to slowly slide down at the end of the match, which might not be great for other robots climbing – it might hit them).

Programming

Class design

The robot uses the **command-based programming** paradigm. It contains **subsystems** that essentially serve as an interface between the commands and the hardware, and **commands** that interact with the hardware using the subsystems.

Subsystems

As mentioned previously, subsystems serve as the interface between the commands and the hardware. They contain SpeedController, Encoder, Gyro and PID objects to send commands using PWM to the hardware components.

Most of the subsystems have a **default command** that is called whenever the subsystem is idle. That is the place where to look for how the subsystem performs during the teleoperated period. All the default commands have *"DefaultControl"* at the end of their name, to distinguish them from other commands.

Subsystems are where you should look like if you wanted to add functions like "setMotorSpeed()" for a part on the robot. Here is the list of the subsystems that the robot has and what they do:

Name	Description
Clamp	Controls the clamp on the elevator (to better hold cubes).
Collector	Controls the collector wheels (intakes and outtakes cubes).
Drivetrain	Controls the drivetrain (drives the robot).
Elevator	Controls the elevator (raises the elevator up, lowers the elevator down).
Fourbar	Controls the fourbar (the weird four-bars thingy that goes up and down).
Hanger	Controls the winch for climbing (not used too much, elevator is fine for climb).
HangerArm	Controls the arm that hooks onto the rung for climbing.
VisionSystemClient	Controls the client that talks to the RPi on the robot (for vision).

Commands

Commands use subsystems to perform various tasks – driving straight, driving for an amount of time, or just simply driving. They are managed by the scheduler that takes care of the order in which they are executed (or even if they are executed in parallel with each other!). There are, however, various ways by which they could be issued (autonomous, the operator, the driver…)

~~The commands' names are arguably descriptive, but~~ here is the list of what they do and how they're issued ~~anyway~~.

Name	Description	Issued By
CollectorIntake	Intakes at a set intake speed.	Autonomous/Operator button
CollectorManualControl	Intakes at a speed dependent on the operator's.	Subsystem (default command)
CollectorOuttake	Intakes at a set outtake speed.	Autonomous/Operator button
DrivetrainDriveDistance	Drives by a distance.	Autonomous
DrivetrainDriveTime	Drives for a certain time.	Autonomous
DrivetrainManualControl	Drives the robot at a speed dependent on the driver's input.	Subsystem (default command)
DrivetrainTimeout	Times out the drivetrain for a certain amount of time.	Autonomous
DrivetrainTurnByAngle	Turns the robot by an angle.	Autonomous
DrivetrainTurnDrive	Turns the robot by an angle and drives at the same time.	Autonomous
DrivetrainTurnToCube	Turns to the nearest cube (closest cube to the camera center).	Autonomous
ElevatorManualControl	Elevates the elevator at a speed dependent on the operator's input.	Subsystem (default command)
ElevatorToHeight	Elevates the elevator to a certain height.	Autonomous
FourbarLower	Lowers the fourbar.	Autonomous/Driver button
FourbarManualControl	Controls the fourbar (holds it up or down, pushes it up or down…).	Subsystem (default command)
FourbarRaise	Raises the fourbar.	Autonomous
HangerArmManualControl	Raises the arm at a speed dependent on the operator's input.	Subsystem (default command)
HangerManualControl	Winches at a speed dependent on the operator's input.	Subsystem (default command)

VisionMonitor	Makes sure that the vision is updated periodically.	Subsystem (default command)
AutonomousCommandGroup	Parses the command strings into actual robot commands.	Autonomous
ClampIn	Clamps in the collector.	Autonomous/Teleoperated (done automatically)
ClampManualControl	Controls the clamp – holds it in either direction, or automatically clamps/unclamps.	Subsystem (default command)
ClampOut	Clamps out the collector.	Autonomous/Teleoperated (done automatically)

Autonomous

The autonomous works in essentially two parts. The first one (the **pre-processing**) happens in the autonomousInit() method of Robot.java. This is where the robot decides, depending on input from the field and form the SmartDashboard, which command sequence to send to the processing phase.

The **processing** happens inside AutonomousCommandGroup. After the AutonomousCommandGroup is created (with the command String as the constructor parameter), it recursively parses the command string into actual robot commands using regular expressions. The scheduler then executes the commands from the AutonomousCommandGroup, until it either has no more commands to execute (and then crashes, but that's fine), or the autonomous period is over (at which point the robot is force-switched to teleoperated).

The autonomous configurations that the robot uses can be found in RobotMap.java (refer to variables **chunks** and **chunkLayout**).

For the full documentation of the parsing language (specifics of optional parameters and detailed descriptions of the commands), refer to the documentation file in the GitHub repository (**AUTONOMOUS_DOCUMENTATION.md**).

RobotMap

RobotMap.java is one of the most important classes on the robot. It stores (pretty much) all the variables used to control the robot. This is where you should be looking, if you want to change things like lengths of the commands, speed controller ports, PID coefficients, autonomous sequences… pretty much **any variable that controls the robot.**

While this is might not seem like the greatest practice with regards to the rule of encapsulation in Java (and the FRC practice – it is recommended to only store variables related to hardware), it is the easiest solution for debugging the robot and changing variables on-the-go.

Changing code

After changing the code on the driver station (using Eclipse), you need to re-deploy the code to the robot. You can do this by either going to Run – Run as – WPILib Java Deploy, or using a shortcut by pressing **ALT + R, S, 1.**

Problems and Solutions

Autonomous being stupid

Make sure to reset the robot before it enters the autonomous period. This is usually the culprit. If you don't do this, the results will be unexpected.

The second likely scenario is that the **SmartDashboard is not properly communicating**. Check out the "SmartDashboard not working properly" part of this chapter of the document. The reason for why the autonomous not work well in this case is because it can't correctly read the values form the SmartDashboard and therefore usually uses the default autonomous sequence (just driving forward) or other weird things.

Another cause for this is having the wrong autonomous selected. Check the SmartDashboard's chunks and board layouts and make sure that they match up with what you want to do.

One of the causes could be the wrong syntax of the language. Make sure that the commands are only separated by spaces, not commas. Make sure that the abbreviations that you are trying to call actually exist. Check for any mistypes and non-ASCII characters (Java is not a huge fan of UTF-8).

The last resort would be checking, whether the encoder and the gyro are producing correct values ~~and making snarky remarks towards the engineering team when doing so~~ by first checking the connections to the encoder and the gyro on the robot and then putting System.out.println(); function calls into the teleopPeriodic method to print out the encoder and the gyro values to see, if they are working.

SmartDashboard not working correctly

If you don't see any commands in the subsystems on the SmartDashboard and only see dashes (---), then you are having this problem. This bug usually occurs after restarting the FRC Driver Station (specifically the SmartDashboard) or editing SmartDashboard's XML file. I don't know the reason behind it, but my opinion is that there is something fishy going on with the FRC Driver Station wrongly remembering the previous XML SmartDashboard file, so it can't match the actual values with the current SmartDashboard's XML.

There only found reliable solution is to reset the SmartDashboard pressing CTRL+N, or just restart the robot until it works. The former should be employed at the competition, or when there is not much time to fix the problems, but if you have enough patience (which I didn't really have by the end of the season, by which point it was re-done about 7 times), you can try restarting either the laptop or the robot and pray to god for it to work.

Robot not connected to the driver station

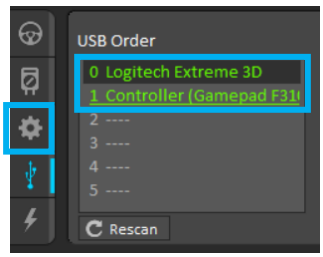
Wireless not working

Try to re-format the radio (follow one of the guides on the internet to do so). Make sure that the radio is powered (that there is a steady blue leftmost light), that it is correctly connected to the switch at the very bottom of the "networking pyramid" (it's a stack of Radio – Raspberry Pi – Switch in the robot), and that the ethernet going into the radio is connected to the ethernet port on the radio that's closer to the power source.

Tethered not working

Make sure that you are not connecting the ethernet cable into the ethernet port on the 1st driver station (the port is faulty and might not function correctly) – use the Ethernet to USB adapter instead. Check, whether the packages are being transmitted to the driver station (there should be lights blinking on the converter) – if not, there is a bad connection (likely in the tethering adapter in the robot itself – it was causing some troubles with falling out and with the connection being loose).

Something is very wrong with the controls



Make sure that the **mode button on the controller is off.** If the green light is on, simply press the "mode" button down and it should be fine.

Check, if the controllers are in the right order on the driver station (driver being in position 0 (*Logitech Extreme 3D*), operator being in position 1 (*Controller (Gamepad …)*), as illustrated by the picture on the right). If they are not, simply drag the upper one down under the bottom one and they should automatically switch positions. ~~Make sure that you are also not accidentally sitting on the controller, the robot's behavior will be unpredictable in that case.~~

Vision (TTC command) not working

~~You're screwed.~~ Check, whether the Raspberry Pi is turned on. If it is, there isn't much you could do to make it work.