2262 VR

GAME MANUAL

Table of Contents

Introduction

2262 VR was created by Zimeng Zhang under the Holliston RoboPanthers, FRC team 2262, as an online, game-based introduction to the world of the FIRST Robotics Competition (FRC). This project is designed to immerse new members into the structure of an FRC match—including simple coding, driving a robot, and game rules—through a virtual environment before stepping onto the real field.

Application links:

- Game manual https://team2262.zimengsrealm.com/2262vr/
- Match field: https://team2262.zimengsrealm.com/2262vr/field/
- Code editor: https://team2262.zimengsrealm.com/2262vr/code/
- Match replayer: https://team2262.zimengsrealm.com/2262vr/replayer/

The game manual is a resource for users of 2262 VR where the following details will be outlined:

- a general overview of the game,
- details about the playing field,
- a description of how to play the game,
- rules related to gameplay, conduct, and recommended ranking,
- a description of how to utilize the virtual tool,
- code documentation

Game Overview

In 2262 VR, two competing units are invited to score blocks, score balls, and cover the dot before time runs out. Units earn additional rewards for meeting specific scoring thresholds and for cooperating with their opponent.

During the first 15 seconds of the match, robots are autonomous. Without guidance from their drivers, robots leave their starting zone and score blocks or balls in their scoring areas.

During the remaining 2 minutes and 15 seconds (FRC time), 1 minute and 45 seconds (challenge time), or 1 minute (expert time), drivers control their robots. Robots collect blocks and balls set across the field and score them in their scoring areas. To access balls, robots must push open the containers.

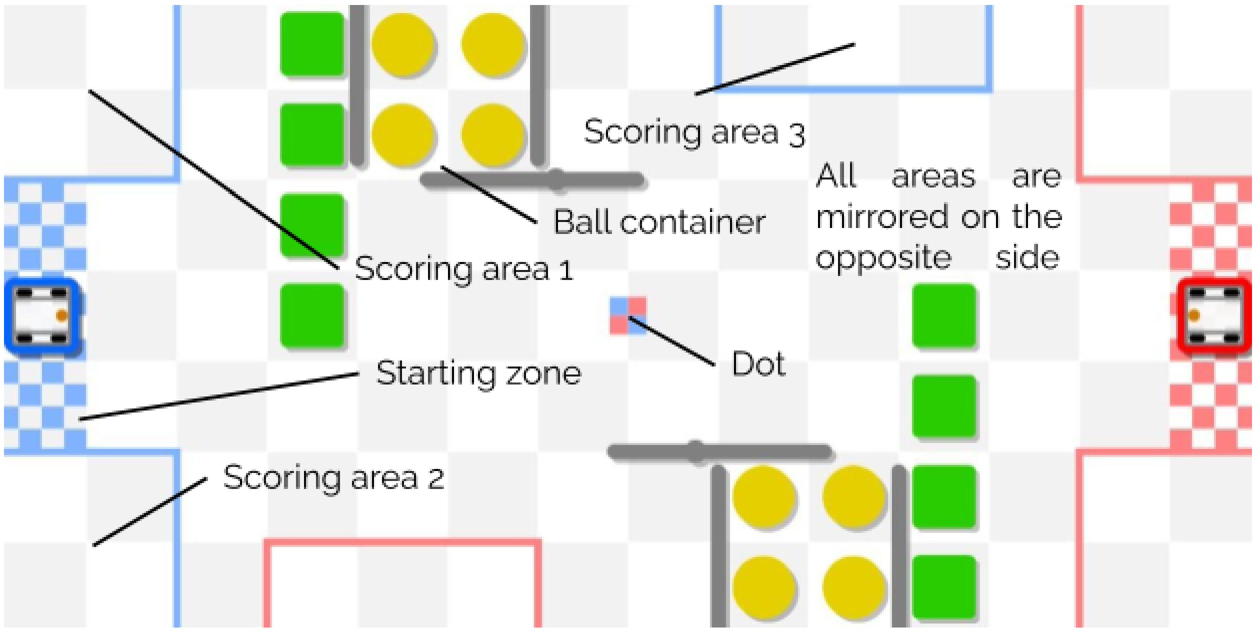
If at least two game pieces are scored in the 3rd scoring area by each unit, both alliances earn a Coopertition Point (which influences their rank in the tournament) and lowers the requirements for a ranking point.

As time runs out, robots finish by covering the dot or returning to their starting zone.

The unit that earns the most points wins the match!

Arena

The arena includes all elements of the game infrastructure that are required to play 2262 VR: the field and scoring elements.



Field

The field is 14 units by 7 units bounded by inward facing walls. Each tile on the field represents 1 square unit.

The field is populated with and surrounded by the following elements:

- 1 starting zone per unit,
- 3 scoring areas per unit,
- 1 ball container per unit,
- 1 dot

Areas, Zones, & Markings

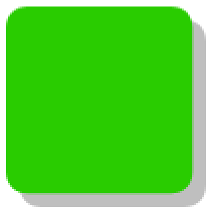
Field areas, zones, and markings of consequence are described below.

- Starting zone: a 1 unit by 3 units area formed by, and including, the unit colored checkered surface and field wall.
- Scoring area 1: a 2 unit by 2 unit area formed by, and including, the unit colored tape and field wall on the same side of the ball container on the unit's side.
- Scoring area 2: a 2 unit by 2 unit area formed by, and including, the unit colored line and field wall on the opposite side of the ball container on the unit's side.
- Scoring area 3: a 3 unit by 1 unit area formed by, and including, the unit colored line and field wall on the same side of the ball container on the unit's side.
- Ball container: a 2 unit by 2 unit area formed by, and including, the field walls and gate on the same side of the unit.
- Dot: a 0.5 unit by 0.5 unit area formed by, and including, a checkered surface of both unit colors.

Scoring Elements

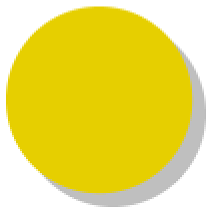
Scoring elements are items that units use to score additional points. There are 2 types of scoring elements used in 2262 VR: Blocks and balls.

Blocks



A block is a 0.75 unit by 0.75 unit green, angular solid with a great amount of friction.

Balls



A ball is a 0.75 unit by 0.75 unit yellow, round solid with a small amount of friction.

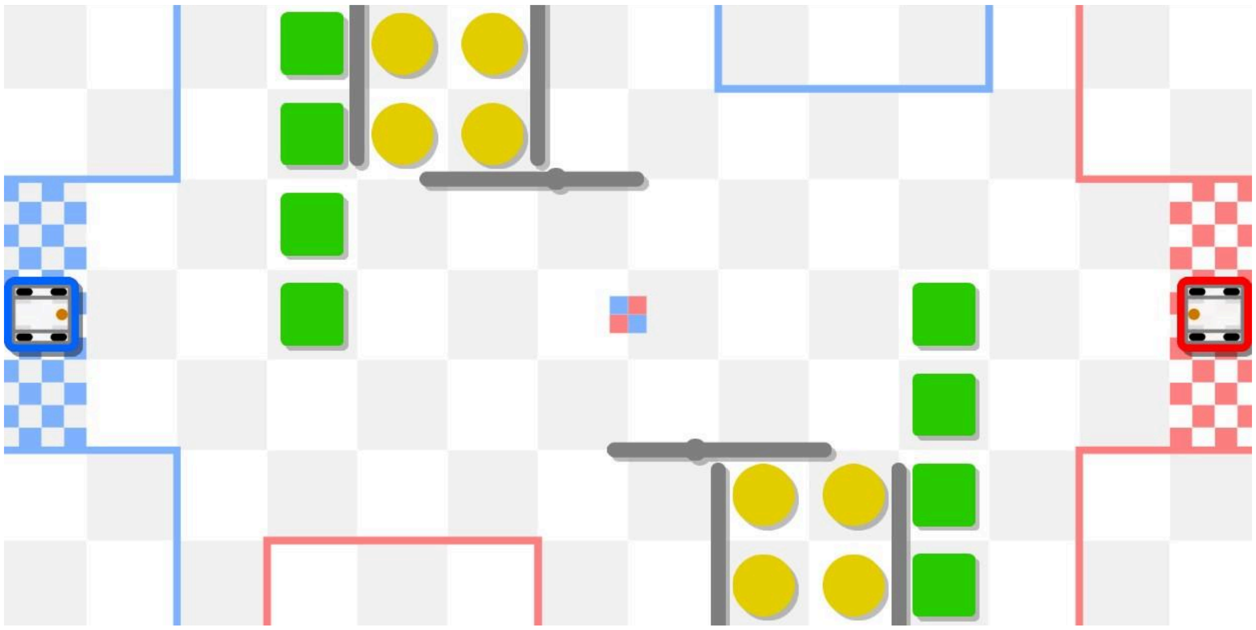
The Field Management System

The Field Management System (FMS) is all the virtual sensors responsible for sensing and controlling the match field. The FMS encompasses all virtual sensors, including robot connections, area sensors, scoring elements, and penalty sensors.

The FMS alerts participants to milestones in the match using audio cues detailed in the table below. Please note that audio cues are intended as a courtesy to participants and not intended as official match markers. If there is a discrepancy between an audio cue and the field timer, the field timer is the authority.

Event	Timer Value	Audio Cue
Match start	0:15 (for auto)	"Cavalry Charge"
Auto ends	0:00 (for auto)	"Buzzer"
Teleop begins	2:15 (for FRC time) 1:45 (for challenge time) 1:00 (for expert time)	"3 Bells"
Final 20 seconds	0:20	"Submarine Sonar Ping"
Match end	0:00	"Buzzer"

Game Details



In 2262 VR, 2 units (a unit is one individual) play matches, set up and implemented per the details described below.

Match Overview

Matches run on game duration cycles, which consist of pre-match setup, the 2-minute and 30-second maximum timed match, and the post-match reset.

During the match, robots collect blocks and balls and score in their scoring areas.

Robots conclude the match parking on their starting zone or parking over the entire dot.

Setup

Before each match begins, The field stages scoring elements as depicted in the field above. The block in the vertical center of the field can be preloaded to the robot on its respective side of the field. All scoring elements must start completely outside any scoring area.

Drivers stage their robots so that the robot and their bumpers are completely contained inside their respective starting zones. Preloading robots may extend their arms outside their starting zones.

Gamepad controllers are connected to the device running the virtual match and should be on the respective sides of the robot it controls.

Match Phases

The first phase of each match is 15 seconds long and called the Autonomous Period (auto). During auto, robots operate without any driver control or input. Robots attempt to leave their parking zone, retrieve scoring elements, and score scoring elements. There is a 3-second delay between auto and teleop for preparation.

The second phase of each match is the remaining 2 minutes and 15 seconds (FRC time), 1 minute and 45 seconds (challenge time), or 1 minute (expert time), and is called the Teleoperated Period (teleop). During teleop, drivers remotely operate robots to retrieve and score scoring elements and park their robots.

Scoring

Units are rewarded for accomplishing various actions throughout a match, including leaving their starting zone, scoring blocks or balls in their scoring areas, parking in the starting zone or over the dot, and winning or tying matches.

Rewards are granted either via match points, Coopertition points, or Ranking Points (often abbreviated to RP, which increase the measure used to rank teams in the recommended ranking method).

All scores are assessed and updated throughout the match, except for penalties not detected by the FMS which can be manually inputted by the human operating the device running the match.

Scoring Element Scoring Criteria

A block and a ball are scored if they are completely inside the outer perimeter of the scoring area markers.

Robot Scoring Criteria

To qualify for leave points, a robot must move such that its bumpers no longer overlap its starting zone marker at the end of auto.

To qualify for park points, a robot's bumpers must be partially or completely contained in their starting zone at the end of the match.

To qualify for dot points, a robot's bumpers must entirely contain the dot at the end of the match.

Coopertition Bonus

In matches, if at least 2 scoring elements are scored in each unit's 3rd scoring area, all units earn 1 Coopertition Point, and the threshold for the scoring RP decreases.

Point Values

Condition	Auto match points	Teleop match points	Ranking points	Coopertition points
Leave	10			
Block scored in scoring area 1	7	6		
Block scored in scoring area 2	12	9		
Block scored in scoring area 3	18	12		
Ball scored in scoring area 1	9	8		
Ball scored in scoring area 2	15	12		
Ball scored in scoring area 3	22	16		
Park in the starting zone		10	1	
Park over the dot		20	1	

Condition	Auto match points	Teleop match points	Ranking points	Coopertition points
Auto RP - robot leaves and scores the preloaded block in any scoring area			1	
Scoring RP - If at least 3 scoring elements are scored in both scoring area 1 and 2. If Coopertition achieved, at least 2 scoring elements must be scored in both scoring area 1 and 2.			1	
Win - completing a match with more match points than your opponent			3	
Tie - completing a match with the same number of match points as your opponent			1	

Violations

Unless otherwise noted, all violations are assigned for each instance of a rule violation. All rules throughout the Game Rules section are detected by the FMS.

Penalty	Description
Pin foul	a credit of 10 points towards the opponent's match point total for each rounded up second of the pin
Minor foul	a credit of 5 points toward the opponent's match point total
Major foul	a credit of 15 points toward the opponent's match point total and the automatic fulfillment of the scoring RP regardless of whether it is already obtained

Game Rules

1. **Starting the match.** Robots and scoring elements must meet all following match-start requirements:
    a. the robot's bumpers are completely inside its starting zone,
    b. the scoring elements are completely outside all scoring areas

   Violation: The match won't start until all requirements are met.
2. **No pinning.** A robot is pinning if it is preventing the movement of an opponent robot by surrounding its bumpers with its arms.

   Violation: Pin foul.
3. **Don't score for the opponent.** A robot may not deliberately put scoring elements in any of the opponent's scoring areas.

   Violation: Minor foul.
4. **No touching scored elements.** A robot may not contact a scoring element scored in any of the opponent's scoring areas.

   Violation: Minor foul.
5. **No de-scoring.** A robot may not de-score a scoring element scored in any of the opponent's scoring areas. Using another scoring element to de-score an opponent's scoring element will not be detected by the FMS and must be assessed and added by the human operating the device running the match.

   Violation: Major foul.
6. **A-Stop.** A robot may kill their auto by pressing or moving the gamepad.

Using Virtual Tools & Applications

The following tools are provided to enable players to interact with and simulate matches in 2262 VR. Unless otherwise specified, all applications require a supported web browser and stable internet connection.

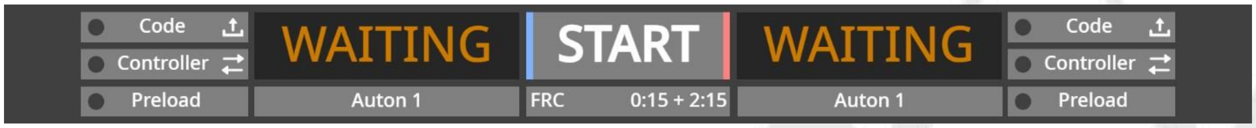
The recommended standings for users using 2262 VR as a mockup competition is determined by the total ranking points of each user at the end of the competition.

Match Field

Link: https://team2262.zimengsrealm.com/2262vr/field/

Purpose: Simulates the competition field for matches and practice.

Controls:



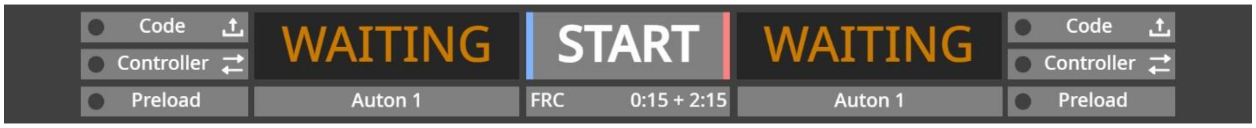
The pre-match dashboard contains the controls for setting up a match.

- Code: Upload code to the respective units.
- Controller: Swap the controllers to the opposite units.
- Preload: Preload a block to the respective units.
- Auton selector: Choose an auton routine to run for the respective units.
- Time toggle: Choose a game time for the duration of the match.

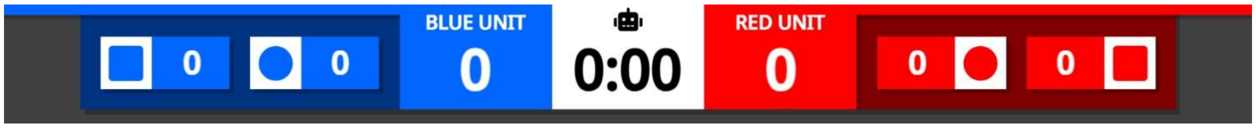
- Add symbol: Award undetected penalty points to the fouled unit.
- Show field / scoreboard: Toggle between showing the field or the scoreboard.
- Download replay: Download the match as a file that can be opened in the replayer.
- X + C keybind: Kill the match.

Condition	Description
⚫ Code ⭱	Code is not uploaded to the robot
🟠 Code ⭱	Code is uploaded but contains errors that may affect performance
🟢 Code ⭱	Code is uploaded and contains no errors
⚫ Controller ⇄	Gamepad is not connected
🟠 Controller ⇄	Gamepad is connected but not currently receiving input
🟢 Controller ⇄	Gamepad is connected and currently receiving input
⚫ Preload	Start without preloaded block
🟢 Preload	Start with preloaded block
Auton 1	Use auton routine 1
Auton 2	Use auton routine 2
Auton 3	Use auton routine 3
FRC 0:15 + 2:15	Play FRC time (0:15 + 2:15)
Challenge 0:15 + 1:45	Play challenge time (0:15 + 1:45)
Expert 0:15 + 1:00	Play expert time (0:15 + 1:00)

Displays:



The pre-match dashboard displays the status of each robot's readiness prior to the match start. The match cannot be started if one or more of the robots display a foul status.



The match dashboard displays the amounts of each scoring element scored by each robot, the scores of each unit, and the time left in the current section of the match.

Condition	Description
WAITING	Waiting status: robot controller is not connected and/or robot code contains errors
READY	Ready status: robot controller is connected and robot code contains no errors
FOUL	Foul status: robot violates one or more match-start requirements
	Blue flag appears for 1 second for each instance of a violation and oscillates vertically at 1 Hz for each second of a pin violation conducted by the blue unit
	Red flag appears for 1 second for each instance of a violation and oscillates vertically at 1 Hz for each second of a pin violation conducted by the red unit
🤝	Coopertition point acquired

Code Editor

Link: https://team2262.zimengsrealm.com/2262vr/code/

Purpose: Allows users to write, test, and deploy robot code in a custom pseudocode developed for 2262 VR.

Controls:

- Left menu: Navigate to different files.
- Top menu: Perform file actions.
- Editor: Use the plus symbol to add lines or press enter when typing. Use the X symbol on each line to delete lines.
- Terminal: View code errors and file action statuses.

Match Replayer

Link: https://team2262.zimengsrealm.com/2262vr/replayer/

Purpose: Review previous matches for strategy and scoring analysis.

Controls:

- Upload: Upload a match replay file.
- Player controls: Play, pause, and seek the replay.
- Show field / scoreboard: Toggle between showing the field or the scoreboard.

2262 VR Code

The 2262 VR code editor is the programming interface for controlling robots during the 2262 VR match. It uses a simplified, text-based code language designed for accessibility and consistency.

The pseudocode language only supports predefined commands for robot actions, such as drive, turn, and move arm. Conditionals, loops, variables, and custom functions are not supported. Autonomous commands execute sequentially in the order they are written. Programs must be deployed or downloaded and opened from the code editor before the start of a match to take effect.

The code editor is not a general-purpose programming environment. Instead, it is intended as a training tool to introduce new users to the structure of basic robot logic, provide a predictable way to test robot actions, and support fair competition by limiting complexity.

Overview

- All commands must appear in valid order and syntax.
- Commands are case-sensitive.
- Units are measured in field units (distance) and degrees (rotation).
- Sequential execution only (no loops or conditionals).

Command Reference

Initialization:

Command	Parameters	Description	Example

`InitPosition(x, y);`	x = horizontal position in field units (float)
	y = vertical position in field units (float)
		Set the robot's starting position in its starting zone.
		Used in auton files only.
		The origin is defined at the bottom left corner of the blue starting zone with positive right and up.
		Positioning for the red unit will be automatically mirrored.
			`InitPosition(0.5, 1.5);`
			
			`// Default start position`

`InitHeading(angle);`	`angle` = initial heading in degrees (float)	Set the robot's starting heading in its starting zone.
		Used in auton files only.
		Zero degrees is defined at 3 o'clock from the center of the blue robot.
		Heading for the red unit will be automatically mirrored.
			`InitHeading(0.0);`
			
			`// Default start heading`

Autonomous movement:

Command	Parameters	Description	Example
`DriveForward(distance);`	`distance` = forward distance in field units (float)	Move the robot forward. Used in auton files only.	`DriveForward(1.0);`
`DriveHeading(distance, angle);`	`distance` = distance in field units (float) `angle` = direction in degrees (float)	Move the robot in a specific direction. Used for swerve only.	`DriveHeading(1.0, 45.0);`
`TurnClockwise(angle);`	`angle` = degrees clockwise (float)	Rotate the robot.	`TurnClockwise(90.0);`
`motorArm.SetPosition (value);`	0 < `value` < 1 (binary)	Set the arm to open or close. Claw must be closed for arm to close.	`motorArm.SetPosition (1);`
`motorClaw.SetPosition (value);`	0 < `value` < 1 (binary)	Set the claw to open or close. Arm must be open for claw to open.	`motorClaw.SetPosition (1);`
`WaitUntilDone();`	none	Wait until all prior commands are executed before starting next.	`WaitUntilDone();`

Motors and actuators:

Command	Parameters	Description	Example

`motorLeft.SetSpeed (value);`	-1 < `value` < 1 (float)	Move the left drive motor to a specific speed.
		Used for tank only.
			`motorLeft.SetSpeed (controller.GetLeftY);`
			
			`// Tank drive controls`

`motorRight.SetSpeed (value);`	-1 < `value` < 1 (float)	Move the right drive motor to a specific speed.
		Used for tank only.
			`motorRight.SetSpeed (controller.GetRightY);`
			
			`// Tank drive controls`

`DriveHeading(distance, angle);`	`distance` = distance in field units (float)
	`angle` = direction in degrees (float)
		Move the robot in a specific direction.
		Used for swerve only.
			`DriveHeading (controller. GetRightMag(), controller. GetRightDir();`
			
			`// Swerve drive controls`

`TurnClockwise(angle);`	`angle` = degrees clockwise (float)	Rotate the robot.	`TurnClockwise (controller. GetLeftX());`
			
			`// Swerve drive controls`

`motorArm.SetPosition (value);`	0 < `value` < 1 (binary)	Set the arm to open or close. Claw must be closed for arm to close.	`motorArm.SetPosition (1);`
`motorClaw.SetPosition (value);`	0 < `value` < 1 (binary)	Set the claw to open or close. Arm must be open for claw to open.	`motorClaw.SetPosition (1);`

Driver control:

Command	Parameters	Description	Example
`controller.GetLeftX()`	none	Returns left joystick X axis value from -1 to 1.	`motorRight.SetSpeed (controller.GetLeftY() - controller.GetLeftX());` `// Arcade drive controls`
`controller.GetLeftY()`	none	Returns left joystick Y axis value from -1 to 1. Used for tank only.	`motorLeft.SetSpeed (controller.GetLeftY() + controller.GetLeftX());` `// Arcade drive controls`
`controller.GetRightX()`	none	Returns left joystick X axis value from -1 to 1.	`motorRight.SetSpeed (controller.GetRightY() - controller. GetRightX());` `// Arcade drive controls`
`controller.GetRightY()`	none	Returns left joystick Y axis value from -1 to 1. Used for tank only.	`motorLeft.SetSpeed (controller.GetRightY() + controller. GetRightX());` `// Arcade drive controls`
`controller.GetLeftMag()`	none	Returns left joystick magnitude from 0 to 1. Used for swerve only.	`DriveHeading (controller. GetLeftMag(), controller. GetLeftDir());` `// Swerve drive controls`
`controller.GetLeftDir()`	none	Returns left joystick direction from 0 to 360. Used for swerve only.	`DriveHeading (controller. GetLeftMag(), controller. GetLeftDir());` `// Swerve drive controls`
`controller. GetRightMag()`	none	Returns right joystick magnitude from 0 to 1. Used for swerve only.	`DriveHeading (controller. GetRightMag(), controller. GetRightDir());` `// Swerve drive controls`

Command	Parameters	Description	Example
`controller.GetRightDir()`	none	Returns right joystick direction from 0 to 360. Used for swerve only.	`DriveHeading(controller.GetRightMag(), controller.GetRightDir());` `// Swerve drive controls`
`controller.AOnPress(command);`	`command` = action to run (command)	Run command when the A button is pressed on the gamepad.	`controller.AOnPress(motorArm.SetPosition(1));`
`controller.BOnPress(command);`	`command` = action to run (command)	Run command when the B button is pressed on the gamepad.	`controller.BOnPress(motorArm.SetPosition(0));`
`controller.XOnPress(command);`	`command` = action to run (command)	Run command when the X button is pressed on the gamepad.	`controller.XOnPress(motorClaw.SetPosition(1));`
`controller.YOnPress(command);`	`command` = action to run (command)	Run command when the Y button is pressed on the gamepad.	`controller.YOnPress(motorClaw.SetPosition(0));`
`controller.UpOnPress(command);`	`command` = action to run (command)	Run command when the D-pad up button is pressed on the gamepad.	`controller.UpOnPress(motorArm.SetPosition(1));`
`controller.DownOnPress(command);`	`command` = action to run (command)	Run command when the D-pad down button is pressed on the gamepad.	`controller.DownOnPress(motorArm.SetPosition(0));`
`controller.LeftOnPress(command);`	`command` = action to run (command)	Run command when the D-pad left button is pressed on the gamepad.	`controller.LeftOnPress(motorClaw.SetPosition(1));`
`controller.RightOnPress(command);`	`command` = action to run (command)	Run command when the D-pad right button is pressed on the gamepad.	`controller.RightOnPress(motorClaw.SetPosition(0));`
`controller.LeftBumperOnPress(command);`	`command` = action to run (command)	Run command when the left bumper button is pressed on the gamepad.	`controller.LeftBumperOnPress(motorArm.SetPosition(1));`
`controller.LeftTrigerOnPress(command);`	`command` = action to run (command)	Run command when the left trigger button is pressed on the gamepad.	`controller.LeftTriggerOnPress(motorArm.SetPosition(0));`
`controller.RightBumperOnPress(command);`	`command` = action to run (command)	Run command when the right bumper button is pressed on the gamepad.	`controller.RightBumperOnPress(motorClaw.SetPosition(1));`
`controller.RightTriggerOnPress(command);`	`command` = action to run (command)	Run command when the right trigger button is pressed on the gamepad.	`controller.RightTriggerOnPress(motorClaw.SetPosition(0));`

Program structure:

Command	Parameters	Description	Example
`include Auton1`	`none`	Include autonomous routine 1 for match execution.	`include Auton1`
`include Auton2`	`none`	Include autonomous routine 2 for match execution.	`include Auton2`
`include Auton3`	`none`	Include autonomous routine 3 for match execution.	`include Auton3`

Set Drivetrain

The drivetrain mode of a robot is determined entirely by the code written in the main program. The FMS auto-detects the drive type from the code. If invalid code is given, it will default to tank.

- Tank Drive: achieved when both left and right motors are directly mapped to the Y-axes of the joysticks.

  Example:

  ```
  motorLeft.SetSpeed(controller.GetLeftY());
  motorRight.SetSpeed(controller.GetRightY());
  ```

- Arcade Drive: achieved when a motor's speed is determined by a combination of forward/backward and steering joystick inputs. Variations exist based on which joystick axes are used.

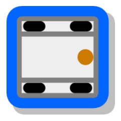
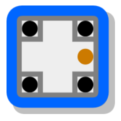
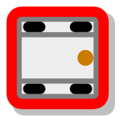
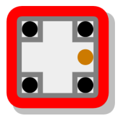
Example (right drive, left steer):

```
motorLeft.SetSpeed(controller.GetRightY() + controller.GetLeftX());
motorRight.SetSpeed(controller.GetRightY() - controller.GetLeftX());
```

- Swerve Drive: achieved when commands use DriveHeading(magnitude, direction) with joystick polar values. A turn command must also be provided.

  Example (right drive, left steer):

```
DriveHeading(controller.GetRightMag(), controller.GetRightDir());
TurnClockwise(controller.GetLeftX());
```

Condition	Description
	Blue robot with tank drivetrain, indicating tank or arcade controls
	Blue robot with swerve drivetrain, indicating swerve controls
	Red robot with tank drivetrain, indicating tank or arcade controls
	Red robot with swerve drivetrain, indicating swerve controls
● ●	The Robot Signal Light (RSL) on each robot indicates the robot's status. Off: Robot does not contain code before match start On and steady: Robot is disabled during the match On and blinking at 2 Hz: Robot is enabled during the match

Data Storage and Deployment

All code execution in 2262 VR follows a standardized deployment pipeline:

- Code Written: Users write pseudocode in the editor.
- Deploy Command: Users deploy to prepare code for practices or the match.
- Cache Encoding: The program is compiled into a simplified instruction set and stored in a temporary cache that is sent to the field.
- FMS Transfer: At match start, the cache is transmitted to the FMS, which distributes commands to the robot.
- Execution: Robots run autonomous routines from the cache sequentially and teleop commands from gamepad mapping.
- Persistence: Programs remain cached until replaced by a new deploy or until a new match is created.
- Local Storage: Users can export code files for offline use and later reload them into the editor and field.