Java Programming

The Ponytail Posse - FTC #8808

9/12/15

2015 – 2016 Season Kickoff

Brought to You By

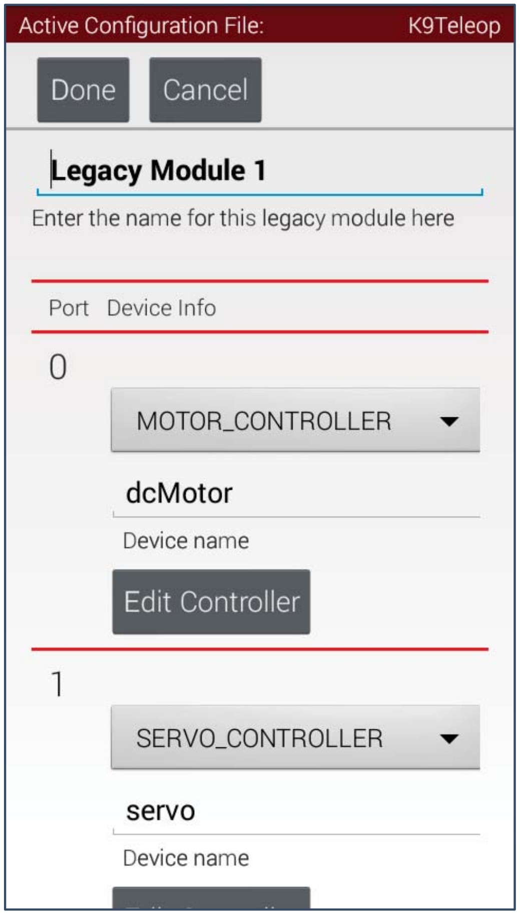
FTC Teams 6705 7190 8712

Outline

- How to make teleop work “out of the box”
- How to make your own teleop
- Autonomous basics
- Source code control
- Resources

Making TeleOp Work Out of the Box

- K9TeleOp
- Initial robot configuration



Active Configuration File: K9Teleop

Legacy Module 1

Enter the name for this legacy module here

Port Device Info

0

MOTOR_CONTROLLER

dcMotor

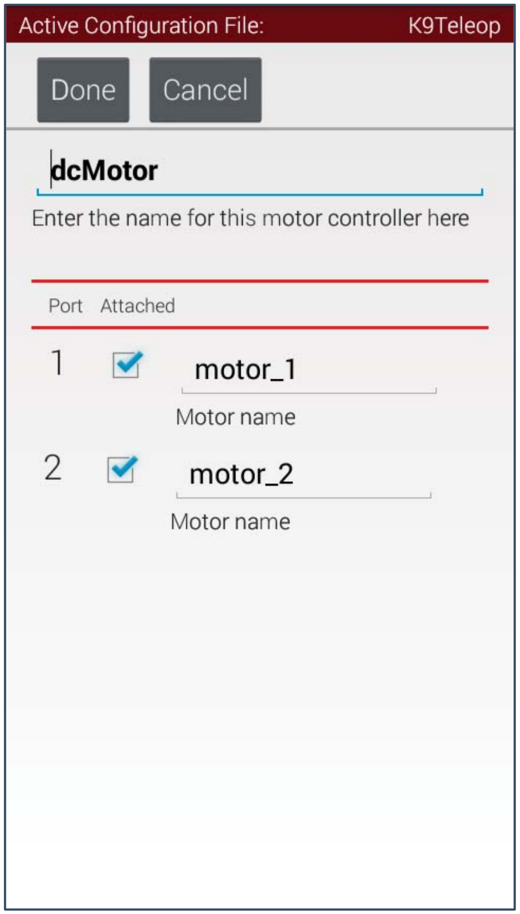
Device name

1

SERVO_CONTROLLER

servo

Device name



Active Configuration File: K9Teleop

dcMotor

Enter the name for this motor controller here

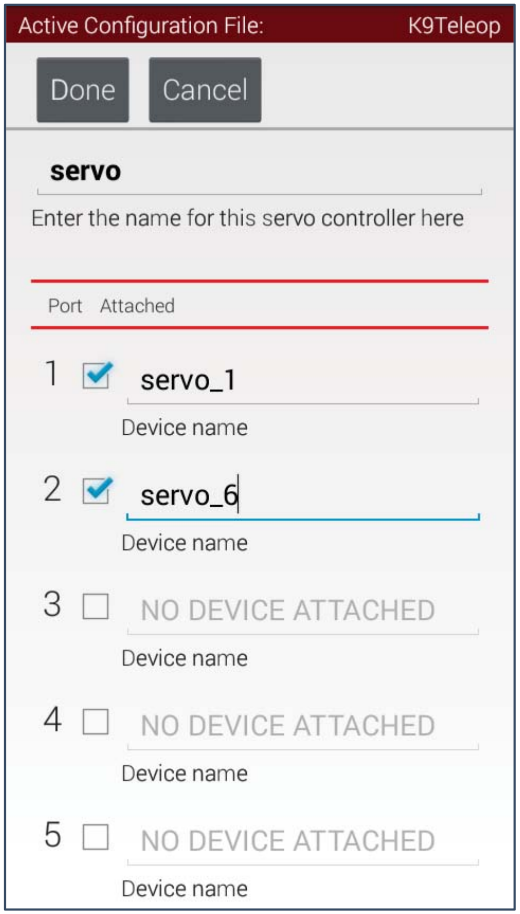
Port Attached

1 ☑ motor_1

Motor name

2 ☑ motor_2

Motor name



Active Configuration File: K9Teleop

servo

Enter the name for this servo controller here

Port Attached

1 ☑ servo_1

Device name

2 ☑ servo_6

Device name

3 □ NO DEVICE ATTACHED

Device name

4 □ NO DEVICE ATTACHED

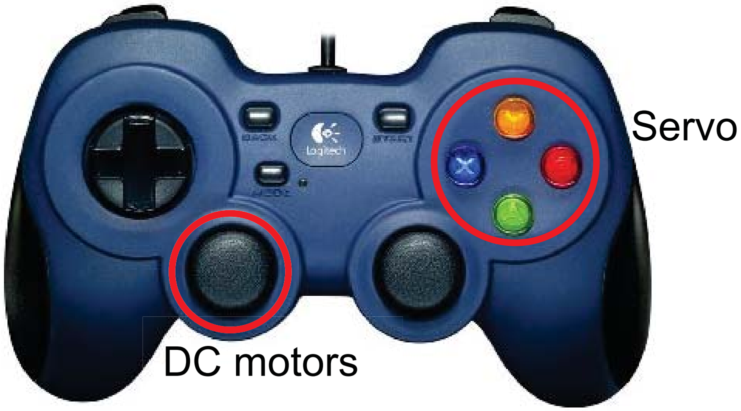
Device name

5 □ NO DEVICE ATTACHED

Device name

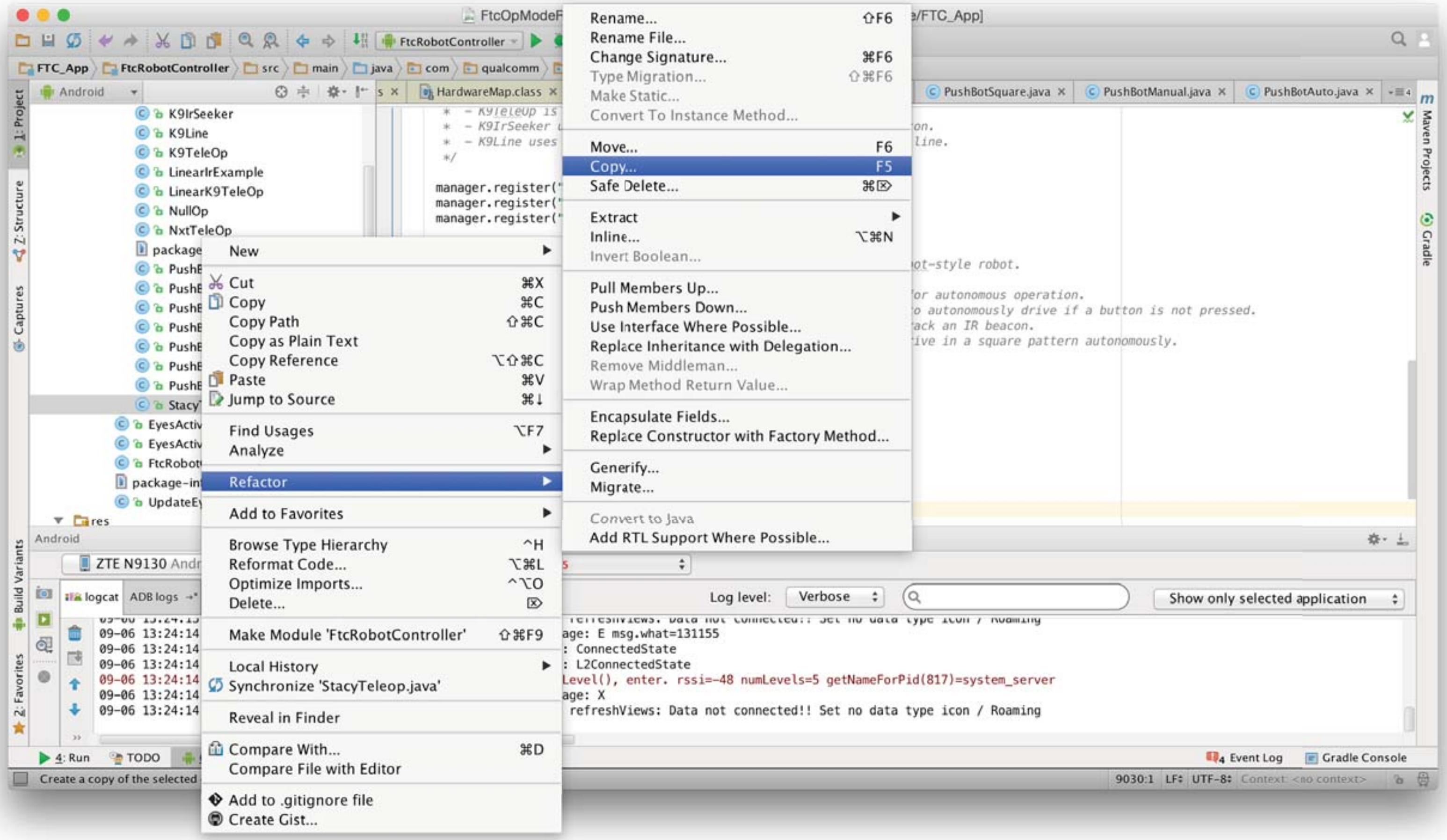
K9TeleOp

- DC motor controls: left toggle controls the wheels
  - Up = forward
  - Down = backward
  - Left = left
  - Right = right
- Servo motor controls: A, B, X, Y buttons
  - X and B = claw
  - Y and A = arm



Making Your Own TeleOp

1. Reconfigure robot
2. Duplicate K9TeleOp (right click → refactor → copy)



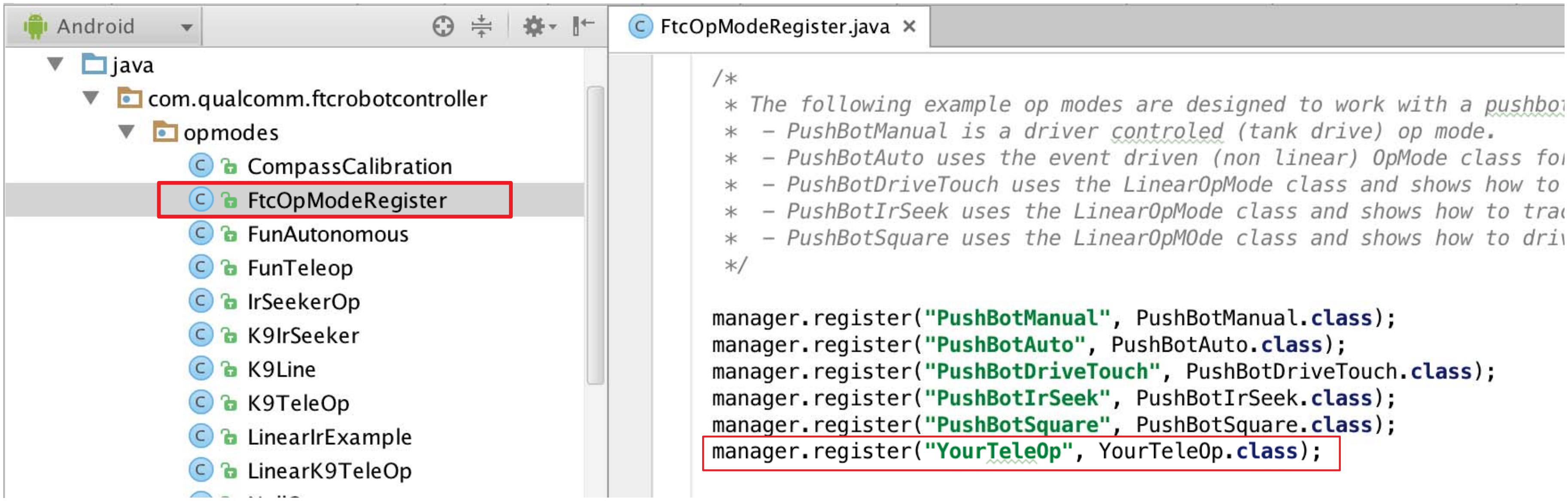
```
 *   - K9TeleOp is
 *   - K9IrSeeker
 *   - K9Line uses
 */

manager.register("
manager.register("
manager.register("
```

```
on.
line.

ot-style robot.

or autonomous operation.
o autonomously drive if a button is not pressed.
ack an IR beacon.
ive in a square pattern autonomously.
```

```
09-06 13:24:14   age: E msg.what=131155
09-06 13:24:14   : ConnectedState
09-06 13:24:14   : L2ConnectedState
09-06 13:24:14   Level(), enter. rssi=-48 numLevels=5 getNameForPid(817)=system_server
09-06 13:24:14   age: X
09-06 13:24:14    refreshViews: Data not connected!! Set no data type icon / Roaming
```

Making Your Own TeleOp

3. Enter your new teleop in the FtcOpModeRegister

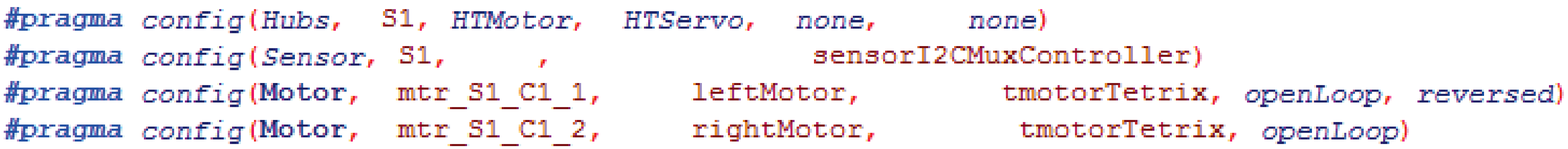


```
/*
 * The following example op modes are designed to work with a pushbo
 *  - PushBotManual is a driver controled (tank drive) op mode.
 *  - PushBotAuto uses the event driven (non linear) OpMode class fo
 *  - PushBotDriveTouch uses the LinearOpMode class and shows how to
 *  - PushBotIrSeek uses the LinearOpMode class and shows how to tra
 *  - PushBotSquare uses the LinearOpMOde class and shows how to dri
 */

manager.register("PushBotManual", PushBotManual.class);
manager.register("PushBotAuto", PushBotAuto.class);
manager.register("PushBotDriveTouch", PushBotDriveTouch.class);
manager.register("PushBotIrSeek", PushBotIrSeek.class);
manager.register("PushBotSquare", PushBotSquare.class);
manager.register("YourTeleOp", YourTeleOp.class);
```

4. Link phone configuration to program through HardwareMap

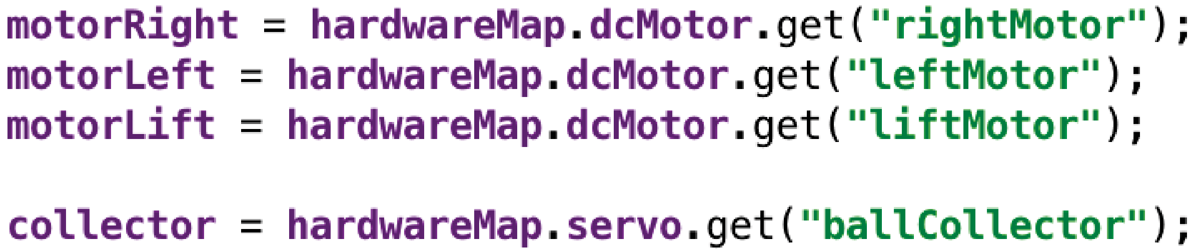
HardwareMap

ROBOTC:



```
#pragma config(Hubs,   S1, HTMotor,   HTServo,   none,      none)
#pragma config(Sensor, S1,      ,                 sensorI2CMuxController)
#pragma config(Motor,  mtr_S1_C1_1,     leftMotor,        tmotorTetrix, openLoop, reversed)
#pragma config(Motor,  mtr_S1_C1_2,     rightMotor,        tmotorTetrix, openLoop)
```

JAVA:



```
motorRight = hardwareMap.dcMotor.get("rightMotor");
motorLeft = hardwareMap.dcMotor.get("leftMotor");
motorLift = hardwareMap.dcMotor.get("liftMotor");

collector = hardwareMap.servo.get("ballCollector");
```

↑
variable you use
for programming

↑
name you put in
the phone

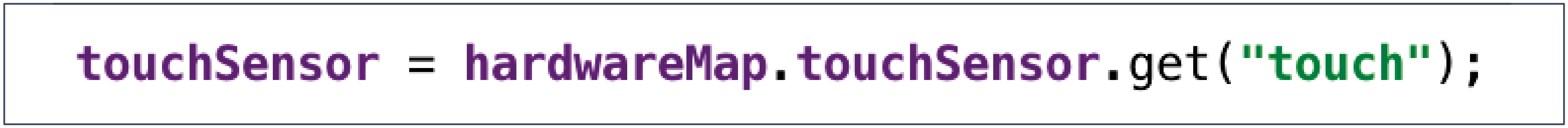
After configuring the hardware on the phone, you need to access it in the program.

HardwareMap is a link that allows the program to access the configuration.

HardwareMap Sensors

- hardwareMap.opticalDistanceSensor
- hardwareMap.touchSensor
- hardwareMap.accelerationSensor
- hardwareMap.compassSensor
- hardwareMap.gyroSensor
- hardwareMap.irSeekerSensor
- hardwareMap.lightSensor
- hardwareMap.ultrasonicSensor
- hardwareMap.voltageSensor

Example:



```
touchSensor = hardwareMap.touchSensor.get("touch");
```

Making Your Own TeleOp

- init()
  - Includes links between hardware configuration and program (hardwareMap)
  - Initializes robot at the beginning
- loop()
  - Contains entire teleop program

ROBOTC	JAVA
`initializeRobot()`	`init()`
`while(true)`	`loop()`

Motor Control

ROBOTC	JAVA

Setting Motor Power
`motor[leftDrive]=100; // -100 to 100`
	Setting Motor Power
	`leftDrive.setPower(1); // -1 to 1`
	`leftDrive.setTargetPosition(4000);`
	`// new! runs to a particular encoder setting!`

Setting Servo Position
`servo[goalGrabber]=128; // 0 to 255`
	Setting Servo Position
	`goalGrabber.setPosition(0.5);`
	`// 0 to 1`

www.bit.ly/cheer4ftc-robotc-java-table

Gamepad Control

ROBOTC	JAVA

Joystick Input
`joystick.joy1_x1, .._y1, .._x2, .._y2 // -128 to 127`
`joy1Btn(1) // 1-12`
`joystick.joy1_TopHat // -1, 0-7`
	Gamepad Input
	`gamepad.left_stick_x, _y, (or right_) // -1 to 1`
	`// y is -1 at top, +1 at bottom!!`
	`gamepad1.left_trigger, _bumper, _stick_button`
	`gamepad1.a, .b, .x, .y, .start, .back`
	`gamepad1.dpad_up, _down, _left, _right`
	`// combine with OR(|) for diagonals`

www.bit.ly/cheer4ftc-robotc-java-table

Variables and Control

ROBOTC	JAVA

Primitive Variables and Control
Basically the same
`int loopCount; float driveSpeed;`
`loopCount=10; driveSpeed=1.0;`
`if (...) { // stuff to do here }`
`while (...) { // stuff to do here }`
	Primitive Variables and Control
	Basically the same
	`int loopCount; float driveSpeed;`
	`loopCount=10; driveSpeed=1.0;`
	`if (...) { // stuff to do here }`
	`while (...) { // stuff to do here }`

www.bit.ly/cheer4ftc-robotc-java-table

Examples

ROBOTC	JAVA

Examples
`motor[leftDrive]=driveSpeed * joystick_joy1_y1;`
`if ( joy1Btn(3) ) {`
`servo[goalGrabber]=255; }`
	Examples
	`leftDrive.setPower(driveSpeed * gamepad1.left_stick_y);`
	`if ( gamepad1.a ) {`
	`goalGrabber.setPosition(1.0); }`

www.bit.ly/cheer4ftc-robotc-java-table

Autonomous

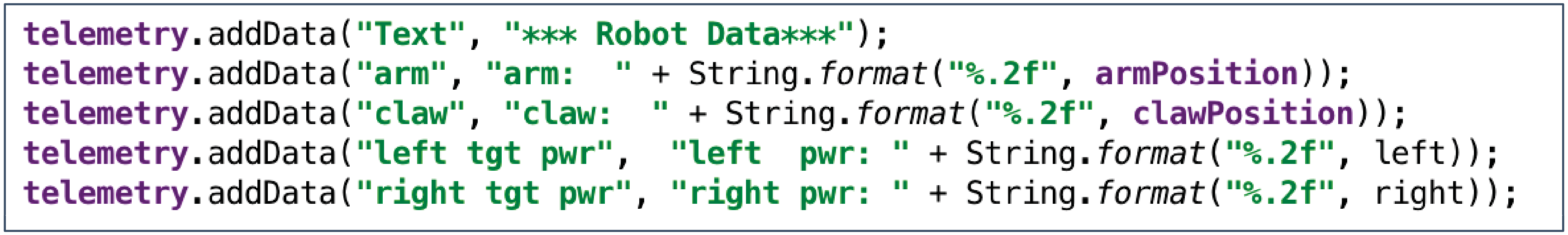
- LinearOpMode

ROBOTC	JAVA
`main()`	`runOpMode()`
`waitForStart()`	`waitForStart()`
`waiting for FCS`	`waiting for the “start” button on the driver station`
`wait (milliseconds)` - not needed to run motor	`sleep (milliseconds)` - needed to run motor
`motor = 0`	`setPowerFloat()`
`while(true)`	`OpModeIsActive()`

You need either “sleep()”, “OpModeIsActive()”, or “waitOneHardwareCycle()” in <u>every step</u> of your program.

Telemetry

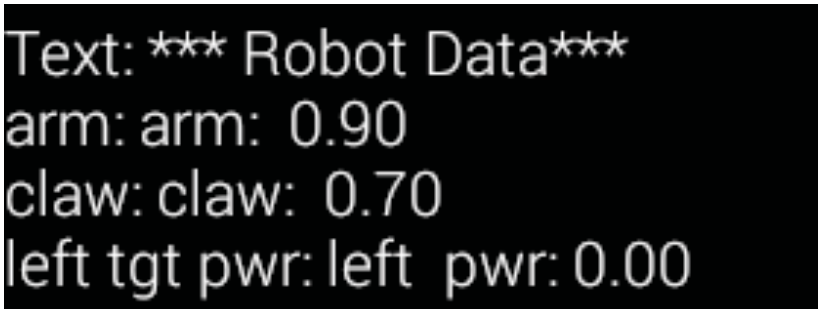
Telemetry data is sent from the robot to the driver station



```
telemetry.addData("Text", "*** Robot Data***");
telemetry.addData("arm", "arm:  " + String.format("%.2f", armPosition));
telemetry.addData("claw", "claw:  " + String.format("%.2f", clawPosition));
telemetry.addData("left tgt pwr",  "left  pwr: " + String.format("%.2f", left));
telemetry.addData("right tgt pwr", "right pwr: " + String.format("%.2f", right));
```

Program in Android Studio

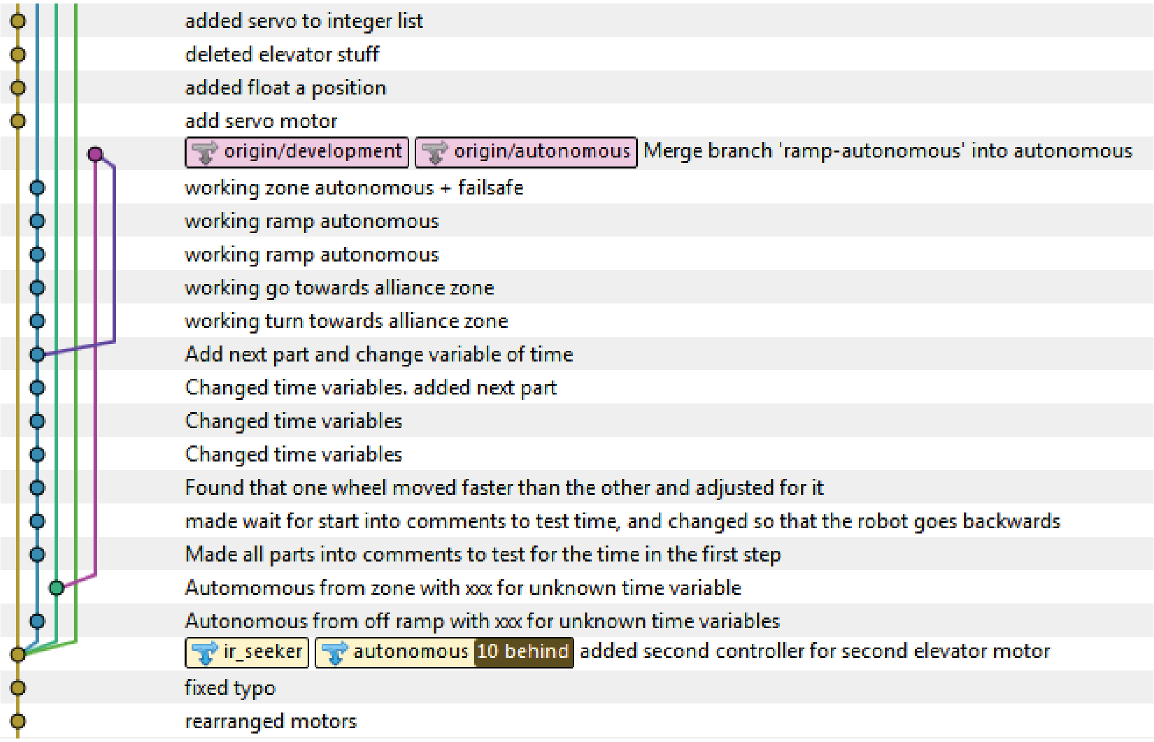
Bottom left corner of driver station



```
Text: *** Robot Data***
arm: arm:  0.90
claw: claw:  0.70
left tgt pwr: left  pwr: 0.00
```

Source Code Control

- Provides a way to experiment with your code
- Check in after every change
- Branch off to try something new
    - Either merge or delete
- SourceTree



added servo to integer list
deleted elevator stuff
added float a position
add servo motor
origin/development origin/autonomous Merge branch 'ramp-autonomous' into autonomous
working zone autonomous + failsafe
working ramp autonomous
working ramp autonomous
working go towards alliance zone
working turn towards alliance zone
Add next part and change variable of time
Changed time variables. added next part
Changed time variables
Changed time variables
Found that one wheel moved faster than the other and adjusted for it
made wait for start into comments to test time, and changed so that the robot goes backwards
Made all parts into comments to test for the time in the first step
Automomous from zone with xxx for unknown time variable
Autonomous from off ramp with xxx for unknown time variables
ir_seeker autonomous 10 behind added second controller for second elevator motor
fixed typo
rearranged motors

Resources

- Android Studio (& more): www.bit.ly/ftc-training-manual
- AppInventor: www.frc-events.usfirst.org/2015/ftcimages
- Java Tutorials:
  - www.ftc.edu.intelitek.com
  - www.docs.oracle.com/javase/tutorial
- RobotC ⟶ Java (Cheer4FTC):
  - www.bit.ly/cheer4ftc-robotc-java-table
  - www.bit.ly/cheer4ftc-robotc-java
- SourceTree: www.sourcetreeapp.com
- **Find this presentation** at www.theponytailposse.com/team-resources

Thanks for listening!

Questions ?