FTC Programming with Android Studio

Patrick R. Michaud
pmichaud@pobox.com
Goals

Introduce FTC robot controller apps
Learn programming basics for FTC robots
Topics

Setup basics
Autonomous and teleop templates
Motor and servo control
Driving logic
Joystick buttons
IR sensor basics
Robot control system overview

Smartphone based
- ZTE Speed
- Motorola Moto G 2nd gen
- Motorola Moto G 3rd gen
- Nexus 5

Controllers for motors, servos, sensors

Programming in Android Studio (Java)
Robot control system basics

Two phones:
- Driver Station
- Robot Controller

Gamepads connect to Driver Station via USB OTG cable

Robot Controller phone connects to Power Distribution Module via USB OTG cable

Core modules connect to Power Distribution Module

Motors, servos, and sensors connect to Core Modules
Setup: Android Studio and ftc_app SDK

(Detailed instructions available on Roboplex.)

1. Install Java SE Development Kit 8
2. Install Android Studio (version 2.2)
3. Unpack ftc_app downloaded from GitHub
4. Import ftc_app into Android Studio
Setup: Configure phone

(Detailed instructions for ZTE speed on Roboplex)

1. Remove SIM card
2. Enable WiFi and Airplane Mode
3. Enable Developer Options
4. Install ZTE Drivers via USB
5. Configure WiFi Direct with team number followed by "-ds" or "-rc"
All teams start with the same basic app framework, called “FtcRobotController”

Teams customize the app with “opmodes” for their robot's specific functionality
Driver Station can place the Robot Controller app into one of several operational modes ("opmodes")

Examples:

- TeleOp
- Autonomous 1
- Autonomous 2
- Stop Robot

Each “opmode” is a separate class instance in the Robot Controller App
Driver Station Demo
Robot Configuration Demo
@TeleOp(name="MyTeleOp", group="Iterative Opmode")
// @Disabled
public class MyTeleOp extends OpMode {

    /* variables common to all methods */

    public void init() {
        /* code to run when opmode entered */
        /* set hardware variables */
    }

    public void loop() {
        /* code run repeatedly during operation
         * once every 25 milliseconds (nominal) */
    }

    public void stop() {
        /* cleanup to run when opmode disabled */
    }
}
In variables section, declare a variable for each motor configured on the robot

```java
private DcMotor leftMotor = null;
private DcMotor rightMotor = null;
```

In init() method, lookup components by name from controller configuration file

```java
leftMotor = hardwareMap.dcMotor.get("leftmotor");
rightMotor = hardwareMap.dcMotor.get("rightmotor");
```

Can reverse motor direction if desired

```java
leftMotor.setDirection(DcMotor.Direction.REVERSE);
```
Java is an *object-oriented language*

Many operations are performed by making *method calls* using the dot notation

```java
leftMotor.setDirection(DcMotor.Direction.REVERSE);
```

Which object we're working with
What we want it to do or get

Method calls can return a result

```java
leftMotor = hardwareMap.dcMotor.get("lmotor");
```
To make a motor move, set its power to be between -1.0 and +1.0 in the loop() method:

```java
leftMotor.setPower(0);       /* turn off motor */
rightMotor.setPower(1.0);    /* full power forward */
leftMotor.setPower(-0.5);    /* 50% power reverse */
```

The motor will continue running at its last commanded speed until given a new setPower value or the OpMode is cancelled.
@TeleOp(name="MyTeleOp", group="Iterative Opmode")
// @Disabled
public class MyTeleOp extends OpMode {

    private DcMotor leftMotor = null;
    private DcMotor rightMotor = null;

    public MyTeleOp() {}

    public void init() {
        leftMotor = hardware.dcmotor.get("leftmotor");
        rightMotor = hardware.dcmotor.get("rightmotor");
        leftMotor.setDirection(DcMotor.Direction.REVERSE);
    }

    public void loop() {
        leftMotor.setPower(0.5);
        rightMotor.setPower(0.5);
    }
}
How to create a new OpMode

Locate “TemplateOpMode_Iterative” or “TemplateOpMode_Linear” in the external.samples folder

Right click and select “Copy”
How to create a new OpMode

In the TeamCode folder, right click and select “Paste”

In the “Copy Class” dialog box, enter a name for the new OpMode

New name: MyTeleOp
Destination package: org.firstinspires.ftc.teamcode
Target destination directory: ...
How to create a new OpMode

In the TeamCode folder, right click and select “Paste”

In the “Copy Class” dialog box, enter a name for the new OpMode
How to create a new OpMode

The new OpMode is created:

```java
@TeleOp(name="Template: Iterative OpMode", group="Iterative Opmode")
@Disabled
public class MyTeleOp extends OpMode {
    private ElapsedTime runtime = new ElapsedTime();
    // private DcMotor leftMotor = null;
    // private DcMotor rightMotor = null;
    /*
    Comment this line (add two slashes in front) to enable the OpMode in the Driver Station menu
```
A simple “move forward” program

```java
@TeleOp(name="MyTeleOp", group="Iterative Opmode")
//@Disabled
public class MyTeleOp extends OpMode {

    private DcMotor leftMotor = null;
    private DcMotor rightMotor = null;

    public MyTeleOp() {}

    public void init() {
        leftMotor = hardware.dcmotor.get("leftmotor");
        rightMotor = hardware.dcmotor.get("rightmotor");
        leftMotor.setDirection(DcMotor.Direction.REVERSE);
    }

    public void loop() {
        leftMotor.setPower(0.5);
        rightMotor.setPower(0.5);
    }
}
```
During the “teleop” phase, drivers use gamepad controllers to command the robot.

**Driver 1**
- `gamepad1.dpad_*`
- `gamepad1.left_stick_x`
- `gamepad1.left_stick_y`
- `gamepad1.right_stick_x`
- `gamepad1.right_stick_y`
- `gamepad1.y`
- `gamepad1.b`
- `gamepad1.x`
- `gamepad1.a`

**Driver 2**
- `gamepad2.dpad_*`
- `gamepad2.left_stick_x`
- `gamepad2.left_stick_y`
- `gamepad2.right_stick_x`
- `gamepad2.right_stick_y`
- `gamepad2.y`
- `gamepad2.b`
- `gamepad2.x`
- `gamepad2.a`
“Tank controls” allow each driving motor to be independently controlled

Driver 1

```java
public void loop() {
    leftMotor.setPower( gamepad1.left_stick_y );
    rightMotor.setPower( gamepad1.right_stick_y );
}
```

gamepad1.left_stick_y (Left wheel)

gamepad1.right_stick_y (Right wheel)
Teleop driving – steering controls

Steering control uses just one stick

Driver 1

gamepad1.left_stick_x
gamepad1.left_stick_y

public void loop() {
  double throttle = gamepad1.left_stick_y;
  double turn     = gamepad1.left_stick_x;

  double leftspeed = throttle - turn;
  double rightspeed = throttle + turn;

  leftMotor.setPower(leftspeed);
  rightMotor.setPower(rightspeed);
}
Teleop driving – steering controls

```java
public void loop() {
    double throttle = gamepad1.left_stick_y;
    double turn = gamepad1.left_stick_x;

    double leftspeed = throttle - turn;
    double rightspeed = throttle + turn;

    leftMotor.setPower(leftspeed);
    rightMotor.setPower(rightspeed);
}
```

Robot will turn when left/right wheels are moving at different speeds

Y-axis is setting forward/backward speed of robot

X-axis is setting the difference between the two motors
Using joystick buttons

Each gamepad has 10 buttons available

Button can be read with “gamepad1.” and the name of the button
Can use `if` statements to perform actions based on button setting:

```java
private DcMotor armMotor = null;

public void init() {
    // ...
    armMotor = hardwareMap.dcMotor.get("armmotor");
}

public void loop() {
    // ...
    if (gamepad1.y) {
        armMotor.setPower(0.1);
    } else {
        armMotor.setPower(0);
    }
}
```

Q: What happens if the `else` part is missing?
Coding style

Always keep opening and closing braces aligned
Indent + align code inside braces
Dualing joysticks

Each robot is given two driver controllers

Many teams use one controller for driving, another for attachment control

We prefer to duplicate all controls on both joysticks
Servo control

Servos are motors that can be told to move (and hold) a specific position.

Servo positions are given as a number from 0 (fully counter-clockwise) to 1 (fully clockwise).
Servo control

To tell the servo to move to a position, use the `setPosition(...)` method:

```java
claw.setPosition(0);    // fully counter-clockwise
claw.setPosition(1);    // fully clockwise
claw.setPosition(0.5);  // move to middle position
```

The servo will attempt to hold the requested position until told to move to a different one.
General pattern for hardware devices

1. Name the device on the robot controller phone

2. Declare a variable in the OpMode
   ```java
   private Servo claw;
   ```

3. Look up the device via `hardwareMap` in `init()`
   ```java
   claw = hardwareMap.servo.get("claw");
   ```

4. Read/write values to the device in `loop()`
   ```java
   claw.setPosition(0.5);
   ```
Make the servo controllable from the gamepad:

In the variables section, add a Servo variable:

```java
Servo claw = null;
```

In the `init()` method, initialize from the hardware map:

```java
claw = hardwareMap.servo.get("claw");
```

In the `loop()` method, control the arm from button A:

```java
if (gamepad1.a) {
    claw.setPosition(0);
} else {
    claw.setPosition(1);
}
```
Autonomous (Linear) OpModes
In the Autonomous portion of the game, there is no driver control (beyond pressing the stop button)

So, how do we get the robot to follow a sequence of commands?
@Autonomous(name="MyAutoOp", group="Autonomous")
// @Disabled
public class MyAutoOp extends OpMode {

    /* variables common to all methods */

    public void runOpMode() throws InterruptedException {
        /* set hardware variables */

        waitForStart();

        /* turn on motors */
        sleep(1000); // drive for 1 second
        /* turn off motors */

    }

}
@Autonomous(name="MyAutoOp", group="Linear Opmode")
//@Disabled
public class MyAutoOp extends LinearOpMode {

    private DcMotor leftMotor = null;
    private DcMotor rightMotor = null;

    @Override
    public void runOpMode() throws InterruptedException {
        telemetry.addData("Status", "Initialized");
        telemetry.update();

        leftMotor = hardwareMap.dcMotor.get("leftmotor");
        rightMotor = hardwareMap.dcMotor.get("rightmotor");
        rightMotor.setDirection(DcMotor.Direction.REVERSE);

        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        if (opModeIsActive()) {
            leftMotor.setPower(0.5);
            rightMotor.setPower(0.5);
            sleep(2000); // wait for two seconds
            leftMotor.setPower(0);
            rightMotor.setPower(0);
        }
    }
}
public void runOpMode() throws InterruptedException {

    leftMotor = hardwareMap.dcMotor.get("leftmotor");
    rightMotor = hardwareMap.dcMotor.get("rightmotor");
    rightMotor.setDirection(DcMotor.Direction.REVERSE);

    // Wait for the game to start (driver presses PLAY)
    waitForStart();

    // move forward for two seconds
    if (opModeIsActive()) {
        leftMotor.setPower(0.5);
        rightMotor.setPower(0.5);
        sleep(2000); // wait for two seconds
        leftMotor.setPower(0);
        rightMotor.setPower(0);
    }

    // move backwards for one second
    if (opModeIsActive()) {
        leftMotor.setPower(-0.5);
        rightMotor.setPower(-0.5);
        sleep(1000); // wait for one second
        leftMotor.setPower(0);
        rightMotor.setPower(0);
    }
}
public void move(double lpower, double rpower, int msec) throws InterruptedException {
    if (opModeIsActive()) {
        leftMotor.setPower(lpower);
        rightMotor.setPower(rpower);
        sleep(msec);
        leftMotor.setPower(0);
        rightMotor.setPower(0);
    }
}

public void runOpMode() throws InterruptedException {
    leftMotor = hardwareMap.dcMotor.get("leftmotor");
    rightMotor = hardwareMap.dcMotor.get("rightmotor");
    rightMotor.setDirection(DcMotor.Direction.REVERSE);

    // Wait for the game to start (driver presses PLAY)
    waitForStart();

    move(0.5, 0.5, 2000); // move forward for two seconds
    move(-0.5, -0.5, 1000); // move backwards for one second
    move(0.5, 0, 1500); // turn to right for 1.5 seconds
}
Sensors
Hardware and Sensor types

In general, everything we put on a robot has a corresponding class for programming in Java:

- DcMotor
- Servo
- TouchSensor
- GyroSensor
- ColorSensor
- UltrasonicSensor
General pattern for hardware devices

1. Name the device on the robot controller phone

2. Declare a variable in the OpMode
   ```java
   private DcMotor motor_1;
   ```

3. Look up the device via `hardwareMap` in `init()`
   ```java
   motor_1 = hardwareMap.dcMotor.get("motor_1");
   ```

4. Read/write values to the device in `loop()`
   ```java
   motor_1.setPower(0.5);
   ```
All of the sensor and hardware types are documented in `doc/javadoc/index.html` in the `ftc_app` directory.
The touch sensor detects presses on its button. It’s ideally used as a limit switch or bumper switch.

OpMode variables:

```java
TouchSensor bumper;
```

init method:

```java
bumper = hardwareMap.touchSensor.get('bumper');
```

loop method:

```java
if (bumper.isPressed()) {
    servo.setPosition(1);
} else {
    servo.setPosition(0);
}
```
Gyro Sensor

The gyro sensor reports how quickly the robot is turning about an axis.

On the z-axis, the gyro sensor will also keep track of how far the robot has turned ("integrating gyro").