Programming and Control Systems
July 9, 2015

Goals

Introduce FTC robot controller apps

Learn programming basics for FTC robots

Disclaimers

* I received my controllers two days ago
  - I'm just now learning things myself

* The published robot controller app is still “beta”
  - It will certainly be updated over the course of the season
  - Some hardware components not yet documented
    - or perhaps not even supported yet

Topics

Setup basics

Autonomous and teleop templates

Motor and servo control

Driving logic

Joystick buttons

IR sensor basics

Setup overview

Check the programming section of “Team Resources” on the FTC website

As of July 9, 2015, they were still under development

Draft version at github.com/ftctechnh/ftc_app

Use “Download ZIP” to obtain copy

Follow instructions in FTCTrainingManual PDF

Setup: Install Java Development Kit (JDK)

Available from
oracle.com/technetwork/java/javase/downloads

Setup: Install + Configure Android Studio

http://developer.android.com/sdk

Run Android SDK Manager

- Add “Google USB Driver” to windows machines

Setup: Prepare Android device

Remove SIM card

Turn on phone

Enable airplane mode

Enable Wifi

Enable developer options

Enable USB debugging

Setup: Connect Android device to computer

Connect ZTE Speed to computer via USB cable

Windows:

- Set USB connection to “driver install”
- Phone will be mounted as a USB drive
- Run AutoRun.exe to install drivers

Set phone to “Charge only”

Setup: Driver Station

Google Play Store

Search for “ftc driver”

Connect joysticks via USB + hub

Setup: Robot controller

Install a Robot Controller App

- Create a configuration file for robot
- Names in configuration used for hardware lookup

Setup: Pair controllers

Initiate pairing from driver station app

Now we can finally(!)
start programming

Programming process basics

All teams start with the same basic app framework

They customize the app with “opmodes” for robot's functionality

OpModes

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

Basic OpMode Java code

```
public class MyTeleOp extends OpMode {

    /* variables common to all methods */

    public MyTeleOp() { }

    public void start() {
        /* 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 */
    }
}
```

Programming - motors

In variables section, declare a variable for each motor configured on the robot

```
DcMotor leftMotor;
DcMotor rightMotor;
```

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

```
leftMotor = hardwareMap.dcMotor.get("left");
rightMotor = hardwareMap.dcMotor.get("right");
```

Can reverse motor direction if desired

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

Java method calls

Java is an *object-oriented language*

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

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

Which object we're working with

What we want the object to do

Method calls can return a result

```
leftMotor = hardwareMap.dcMotor.get("left");
```

Programming - motors

To make a motor move, set its power to be between -1.0 and +1.0 in the loop() method:

```
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 disabled

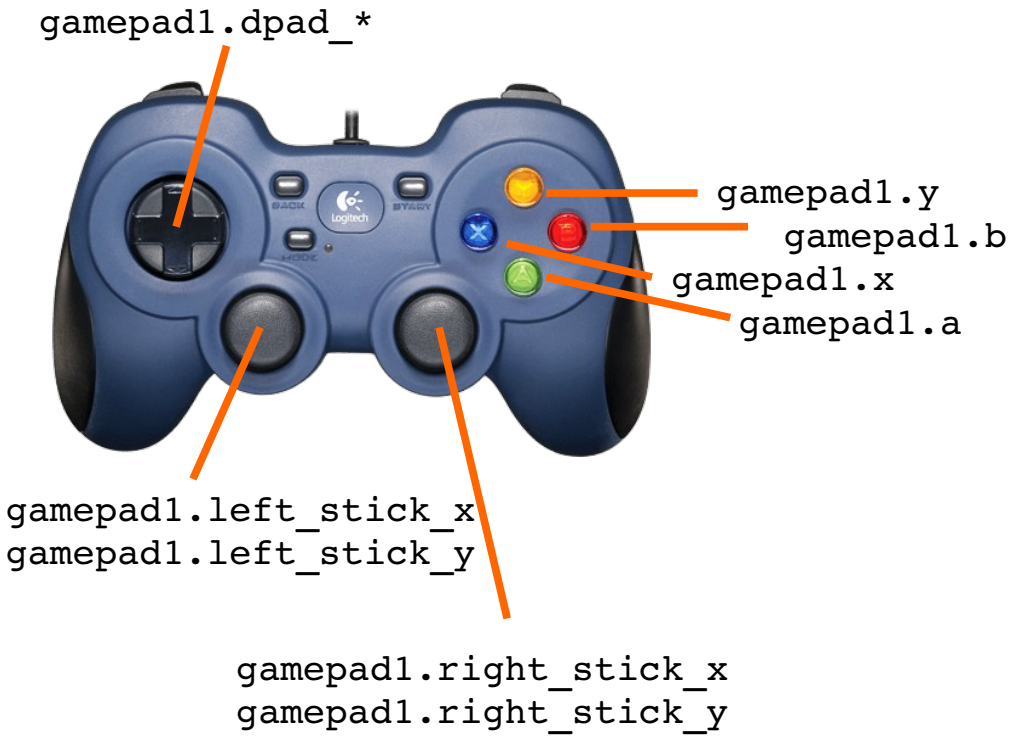
A simple “move forward” program

```
public class MyTeleOp extends OpMode {

    DcMotor leftMotor;
    DcMotor rightMotor;

    public MyTeleOp() { }

    public void start() {
        leftMotor = hardware.dcmotor.get(“left”);
        rightMotor = hardware.dcmotor.get(“right”);
        leftMotor.setDirection(DcMotor.Direction.REVERSE);
    }

    public void loop() {
        leftMotor.setPower(0.5);
        rightMotor.setPower(0.5);
    }

}
```

Teleop – joystick controls

During the “teleop” phase, drivers use joystick controllers to command the robot

Driver 1



Driver 2

Teleop driving – tank controls

"Tank controls" allow each driving motor to be independently controlled

Driver 1

gamepad1.left_stick_y
(Left wheel)

gamepad1.right_stick_y
(Right wheel)

```
public void loop() {

    leftMotor.setPower(gamepad1.left_stick_y);
    rightMotor.setPower(gamepad1.right_stick_y);

}
```

Teleop driving – steering controls

Steering control uses just one stick

Driver 1

gamepad1.left_stick_x
gamepad1.left_stick_y

```
public void loop() {

    float throttle = gamepad1.left_stick_y;
    float turn = gamepad1.left_stick_x;

    float leftspeed = throttle – turn;
    float rightspeed = throttle + turn;

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

}
```

Teleop driving – steering controls

```
public void loop() {

    float throttle = gamepad1.left_stick_y;
    float turn = gamepad1.left_stick_x;

    float leftspeed = throttle – turn;
    float 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

Teleop driving – steering controls

```
public void loop() {

    float throttle = gamepad1.left_stick_y;
    float turn = gamepad1.left_stick_x;

    float leftspeed = throttle – turn;
    float rightspeed = throttle + turn;

    leftspeed = Range.clip(leftspeed, -1, 1);
    rightspeed = Range.clip(rightspeed, -1, 1);

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

}
```

We should make sure that our power stays in the Range -1.0 to +1.0, though

Joystick deadzone detection

The new system now handles this for us!

When the joysticks are physically close to 0.0, the system automatically sets them to exactly 0.0

This can be changed by the program if desired:

```
gamepad1.setJoystickDeadzone(0.1);
```

Using joystick buttons

Each gamepad has 10 buttons available

Button can be read with “gamepad1.” and the name of the button

Using joystick buttons

Can use `if` statements to perform actions based on button setting:

```
public void loop() {

    if (gamepad1.x) {
        leftMotor.setPower(1);
    }
    else {
        leftMotor.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)

Programming - servos

In variables section, declare a variable for each motor configured on the robot

```
Servo arm;
```

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

```
arm = hardwareMap.servo.get("arm");
```

It's also a good idea to initialize the servo

```
arm.setPosition(0.5);   /* move servo to middle */
```

Servo control

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

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

The servo will attempt to hold the requested position until told to move to a different one

I haven't found a way to set the speed of the servo movement (as we could in RobotC)

Programming – IR seeker

In variables section, declare a variable for each IR seeker

```
IrSeekerSensor irSeeker;
```

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

```
irSeeker = hardwareMap.irSeekerSensor.get(“irseeker);
```

Reading IR Seeker values

To read the angle from the IR Seeker:

```
float bearing = irSeeker.getAngle();
```

Result is an estimate of the angle to the beacon

- negative values: beacon is to the left
- positive values: beacon is to the right
- zero: beacon is directly ahead