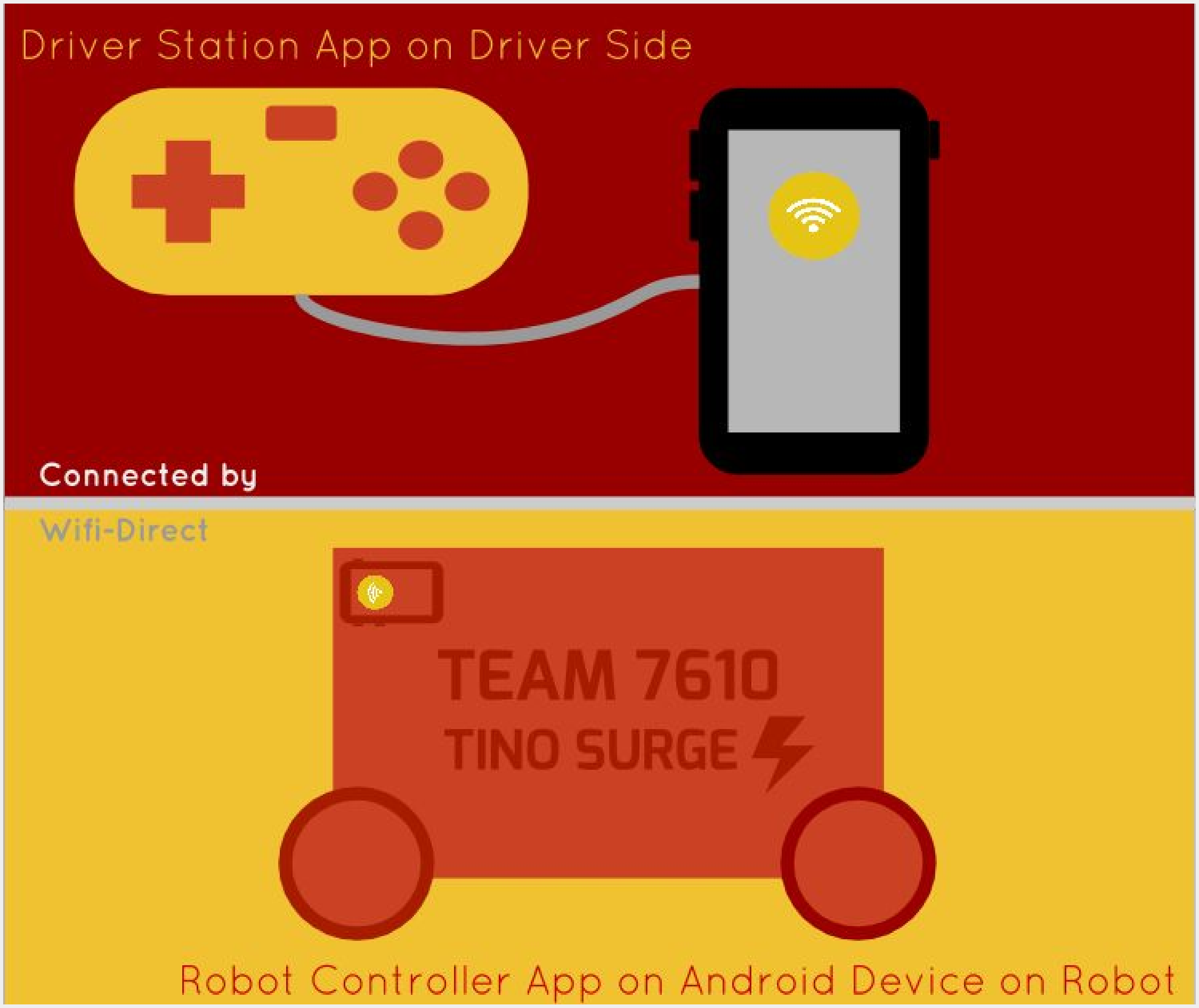
FTC Team 7610 Tino Surge

Driver Controlled Code

Table of Contents

Software and Tele-Op Files

How do Android devices run software?


Driver Station App on Driver Side
Connected by
Wifi-Direct
TEAM 7610
TINO SURGE
Robot Controller App on Android Device on Robot

Tele Op Files and Their Structure

Tele Op files enable control of the robot via the gamepad. In this particular file, the joystick controllers change the direction of the robot. Due to the benefit of Java, we can now use one joystick to control the drive train if we want. This program does that. This is a basic file, so feel free to add methods of your own!

```
//state the package that this file is written in
package com.qualcomm.ftcrobotcontroller.opmodes;

//import necessary packages
//import classes that you need to create objects for this program;
import ... //example: import com.qualcomm.robotcore.hardware.DcMotor;
import ...
import ...
import ...

//define your class
public class <classNameOfChoice> extends OpMode {
    public void init() {
        //runs on robot initiation
    }

    public void loop() {
        //the commands to be sent to the robot go here
    }

    public void stop() {
        //what the robot does when it stops, usually empty
    }

}
```

Starting Your File

Code

```
package name;

import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.util.Range;
```

Description

Tell the program which package it is part of, as in, it's file location in the application. After that, you will need to import classes. Creating instances, or objects, of these classes will allow you to manipulate them with various ways. For example, to manipulate DC Motors with your program, you will need objects from the DcMotor class, with each symbolizing a single motor. You create the objects later in this tutorial, but for now, just import the classes you need, as they are blueprints for the objects created. Because we are working with motors and servos, we need to import the DcMotor and Servo classes, using their full package location of course! Because we are writing a custom op-mode, we import the OpMode class, and because we are using a range-clipping method, we will import the Range class.

Class, Objects, and Values

Code

```
//file name is <yourClassName>.java
public class <yourClassName> extends OpMode {

   //we will use these values to set range limitations later
   final static double MIN_SERVO_ONE_RANGE = 0.40;
   final static double MIN_SERVO_ONE_RANGE = 0.60;
   final static double MIN_SERVO_TWO_RANGE = 0.35;
   final static double MIN_SERVO_TWO_RANGE = 0.50;

   //track position of servos
   double servoOnePosition;
   double servoTwoPosition;

   /*objects to manipulate in this program, you can name them
   whatever you want*/
   DcMotor one;
   DcMotor two;
   Servo one;
   Servo two;
}
```

Description

Here we are naming our class and making it extend the code from the OpMode class, which was imported earlier, page 4. Next, we declare some variables, and give them the final keyword, which ensures that their values can't be changed. We don't want to change ranges in this program, as that may surpass hardware limitations. Next, we have created some variables that will store positions of servos. Lastly, we have made objects using the DcMotor and Servo classes. We have one object for each element on the robot to be controlled.

init()

Code

```
public void init() {
    /*using hardwareMap.dcMotor's get(String motorNameInApp)
     method where motorName is what you name the motor in your
     configuration file in the FTC Robot Controller app*/
    motorOne = hardwareMap.dcMotor.get("motor_1");
    motorTwo = hardwareMap.dcMotor.get("motor_2");
    motorTwo.setDirection(DcMotor.Direction.REVERSE);
    /*check your robot and you will notice that one of the
    two back wheels and one of the two front wheels need to
    be reversed in order for the robot's wheels to run in
    the same direction*/

    /*the same process is applied on servos*/
    servoOne = hardwareMap.servo.get("servo_1");
    servoTwo = hardwareMap.servo.get("servo_2");
}
```

Description

The objects have now been created, each created for one element on the robot. We now need to assign them to those elements. Using the get method described in the code comments above, we assigned DcMotor objects to real motors on the robot. The information in the quotations corresponds to the name we give those motors in the configuration file. If you do not give these names in the configuration file and run the program, an error will be returned... We performed the same function on servos.

loop()

Code

```
public void loop() {
    //we will assume that motorOne is right, and motorTwo is left
    //throttle: left_stick_y ranges from -1 to 1, -1 is full up, and 1 is full down
    // direction: left_stick_x ranges from -1, or full left, and 1, or full right

   float throttle = -gamepad1.left_stick_y;
   float direction = gamepad1.left_stick_x;
   float right = throttle - direction;
   float left = throttle + direction;

   //the right and left variables are now clipped using Range.clip, which passes
   //the values for the clipped value, the minimum, and then the maximum
   right = Range.clip(right, -1, 1);
   left = Range.clip(left, -1, 1);

   //using the scaleInput() function, which isn't imported
   // we assign power to the motors based on a value
   // that is scaled based on gamepad1's input

   right = (float)scaleInput(right);
   left = (float)scaleInput(left);

   //assign the power values
   motorOne.setPower(right);
   motorTwo.setPower(left);
```

Description

We use Range.clip(), some logic, and the methods described in the SDK docs as well as the comments in the code above to perform some calculations and scales and finally assign some power to the motors on the robot. For more detail on scaleInput(), keep reading! This op-mode is not done yet...

loop()

Code

```
    //if certain buttons are pressed, increase/decrease the servo position by some
    //number, in this case 0.05

    if (gamepad1.a) { //if a button
        servoOnePosition += 0.05;
    }

    if (gamepad1.y) { //if y button
        servoOnePosition -= 0.05;
    }

    if (gamepad1.x) { //if x button
        servoTwoPosition += 0.05;
    }

    if (gamepad1.b) { //if b button
        servoTwoPosition -= 0.05;
    }

    servoTwoPosition = Range.clip(servoTwoPosition, MIN_TWO_RANGE, MAX_TWO_RANGE);
    servoOnePosition = Range.clip(servoOnePosition, MIN_ONE_RANGE, MAX_ONE_RANGE);

    servoOne.setPosition(servoOnePosition);
    servoTwo.setPosition(servoTwoPosition);

} //end of loop()
```

Description

Using some conditionals, we have programmed the servos to increase/decrease based on different commands. We also put some limitations on the servo positions. We then used the setPosition() method to set the power to servos.

stop() and telemetry data

Code

```
/*this is how you report telemetry data*/
telemetry.addData("Data Title","Actual Data");
/*if you wanted to report the motor power for motorOne*/
telemtry.addData("motorOne power: ",String.format("%.2f", left); //100ths place

/*this is what a stop() method looks like in thsi op-mode*/
public void stop() {
   //does nothing in the stop() method...robot has stopped in this case
}
```

Description

Let's begin with the telemetry data. This is reported using calling the telemetry.addData() method. Two arguments are passed, both being data types that can be printed. The first serves as the name of the data that will be displayed as the data "header". The data itself, as in the number of rotations, or power of motor, is passed as the second argument for appropriate printing. The stop() method in this op-mode does nothing. Whatever runs in the stop() method is run once the robot controller has stopped communicating with the driver station.

the extra Surge ⚡: dScale()

Code

```
double scaleInput(double dVal)  { //think of dVal as the input from the gamepad
   double[] scaleArray = { 0.0, 0.05, 0.09, 0.10, 0.12, 0.15, 0.18, 0.24,
           0.30, 0.36, 0.43, 0.50, 0.60, 0.72, 0.85, 1.00, 1.00 };
   /*above is an array of values the dVal can be scaled to*/

   /*the code in the next 6 lines assigns an index from the array to the value*/
   int index = (int) (dVal * 16.0);
   if (index < 0) {
       index = -index;
   } else if (index > 16) {
       index = 16;
   }

   /*the code below assigns the value of the index to the variable dScale*/
   double dScale = 0.0;
   if (dVal < 0) {
       dScale = -scaleArray[index];
   } else {
       dScale = scaleArray[index];
   }

   /*dScale is returned*/
   return dScale;
}
```

Description

Let's begin with the telemetry data. This is reported using calling the telemetry.addData() method. Two arguments are passed, both being data types that can be printed. The first serves as the name of the data that will be displayed as the data "header". The data itself, as in the number of rotations, or power of motor, is passed as the second argument for appropriate printing. The stop() method in this op-mode does nothing. Whatever runs in the stop() method is run once the robot controller has stopped communicating with the driver station.

Credits

PROJECT MANAGER

Deep Sethi

SOFTWARE LEAD

Shashank Mahesh

AUTONOMOUS MANAGER

Anik Gupta

TELE-OP MANAGER

Abhinav Gokhale

DEV TEAM

Rajath Rao

Varun Shenoy

Andrew Liang

Benjamin Liang

Mokshith Voodarla

Eugene Kim