CONTROL SYSTEMS

ROBOT CHALLENGE

A guide to setting up your program for the robot challenge activity

The objective of this guide is to assist you in setting up the code for the robot you are building. Since you are building your own, ***custom*** robot, the examples in this guide **might not** perfectly match your robot.

RELEVANT FILES

There are 3-4 files you will need to modify in order to program your robot. These files are:

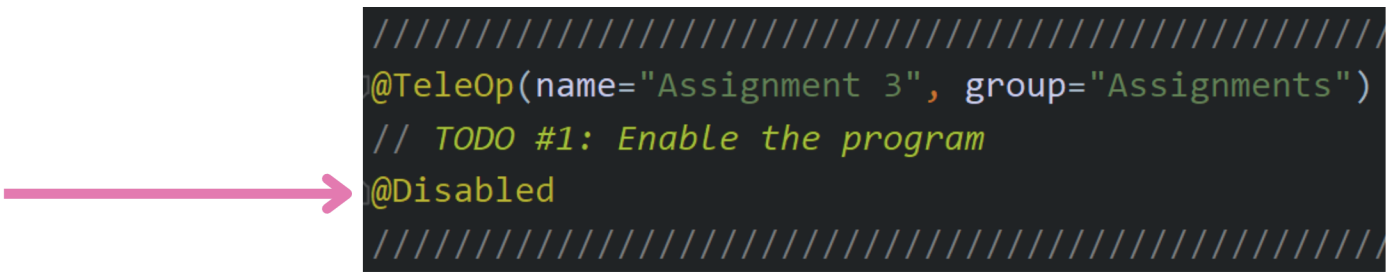
- DriveTrain
- Mechanisms
  - This file is *only* needed if you have more actuators than those in your drivetrain
- AutonomousMode
- TeleOpMode

NEW CONFIG

Your robot is not the testboard! You will need to create a new configuration file on the Driver Station App prior to writing any robot code. If you need a refresher on how to do this, revisit Assignment1a.

DISABLE OLD FILES

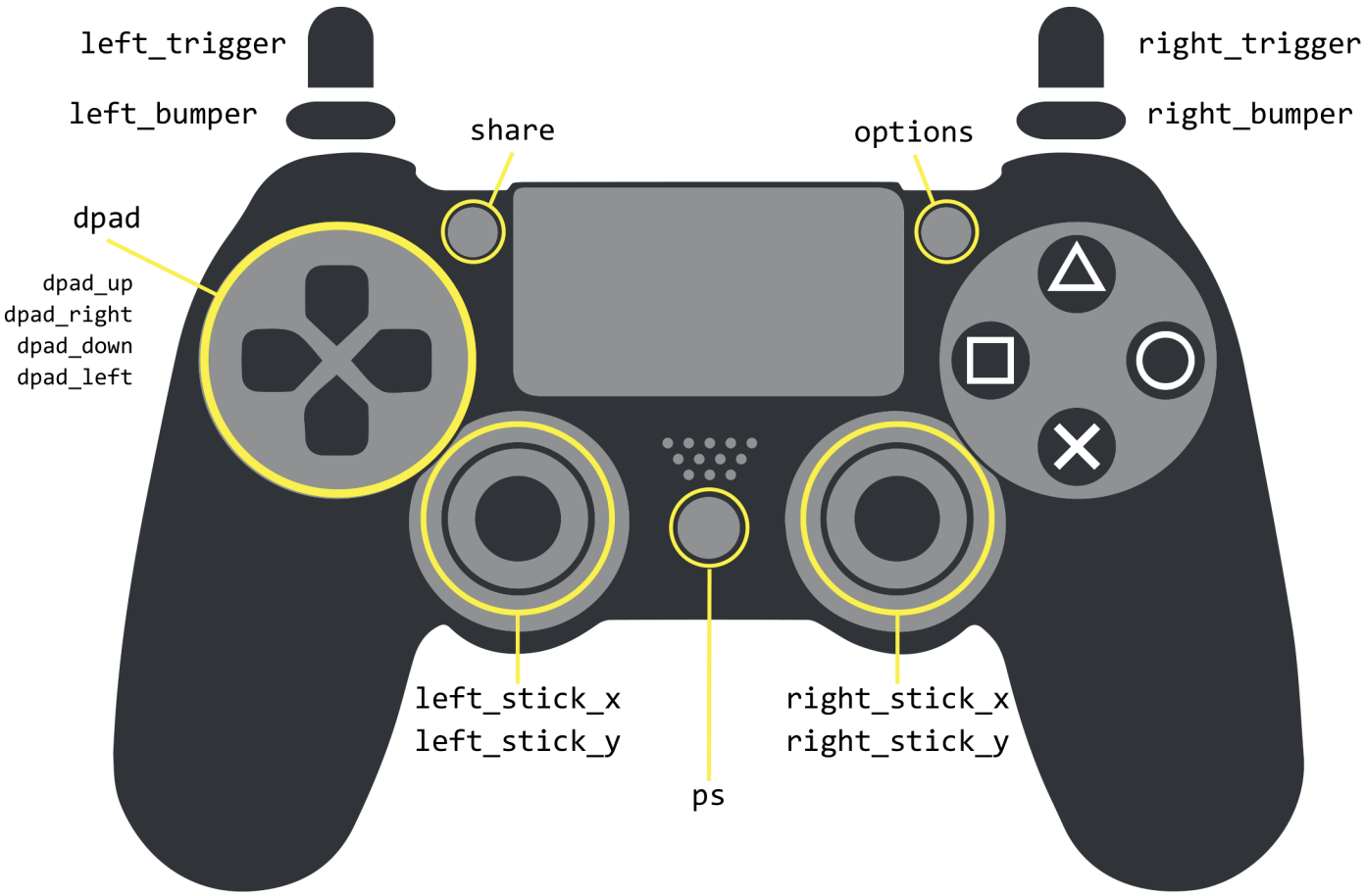
To make using your robot easier, in your program files, disable all files from the previous activities (Assignments 1a, 1b, 2, and 3).



USING A CONTROLLER

Know your controller

You will need to write your TeleOp code such that it can be controlled by your gamepad. The following is a layout of the button map for a PS4 controller.



triangle circle cross square

DRIVETRAIN

Open the file called DriveTrain

There are two types of drivetrain you may have built, either a motor-driven one or a continuous rotation servo-driven one. There is a difference in how these are programmed. For the purposes of brevity, in written instructions, these will *all* be referred to as 'motors'.

Follow the code specified for your drivetrain.

TODO 1. Create motor variables

Every motor in your drivetrain needs a variable. Create them with relevant names as modelled below.

```
// Code Segment - Motor
private DcMotor leftMotor;
```

```
// Code Segment - CRServo
private CRServo backLeft;
```

TODO 2. HardwareMap

Every motor in your drivetrain needs to be linked to a motor that you configured on the Control Hub using the phone. Create this link by following the examples modeled below.

```
// Code Segment - Motor
leftMotor = hardwareMap.get(DcMotor.class, "leftMotor");
```

```
// Code Segment - CRServo
backLeft = hardwareMap.get(CRServo.class, "backLeft");
```

DRIVETRAIN

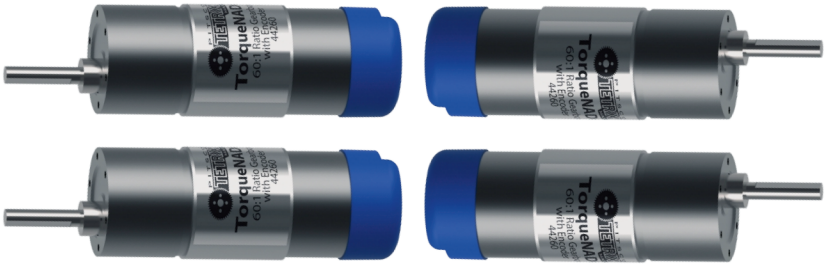
TODO 3. Reverse

Some of the motors in your drive train will need to be reversed.

Here's why:

Imagine this motor naturally spins clockwise when facing it.

Now imagine there are four of them arranged in a configuration like you might see in a robot.



If all of these motors naturally spin clockwise when given a power of 1, would they all be helping push the robot in the same direction?

No! One side would be pushing the robot forwards and the other backwards.

Reversing the motor direction lets us give all motors a power of 1, but travel in the direction that makes sense for the robot's "forwards".

You should change the motors in your code such that the direction you want to travel is forwards when given power of 1.

All motors default to being set to forwards, so you only need to perform this action for the motors needing reversed.

The method for setting direction is the same for motors and servos.

```
// Code Segment

leftMotor.setDirection(REVERSE);
```

DRIVETRAIN

TODO 4. Zero Power Behavior

Motors have a set behavior for what they do when you don't given them any power. For example, when you've been pressing on the joystick to go and then let go, letting the motor go back to zero. There are two modes:

- **Coast**: The motor will stop providing current, but allow momentum to continue.
- **Brake**: The motor will use current to stop momentum quickly.

All motors default to Coast behavior. If you **want** to change to brake behavior, you may do so. The method for setting zero power behavior is the same for both motors and servos.

```
// Code Segment
leftMotor.setZeroPowerBehavior(BRAKE);
```

TODO 5. setDrivePower

To make controlling your robot simple, you need to create a method that takes in power values and sends them to your motors

The example below shows you how to setup the code. This method will be the same for both motors and servos.

```
// Code Segment
public void setDrivePower(double leftPow, double rightPow){
    leftMotor.setPower(leftPow); }
}
```

You will need one of these for each motor you are using, renamed appropriately.

DRIVETRAIN

TODO 6. stopDriving

Sometimes, we make methods that aren't strictly necessary, however, they make future work we do easier.

The stopDriving method is something we can call that will turn power off to the motors. This is a method that will likely prove useful to your autonomous programs.

```
// Code Segment
public void stopDriving(){
    setDrivePower(_,_);
}
```

Parts of this code have been obscured. Think about what values you would need to input in order to have the motors stop spinning.

MECHANISMS

Depending on the design of your robot, you may have devices that need to be setup in the mechanisms file. Any devices not part of your drivetrain should go into this file.

Setup for these devices will follow the same pattern as the drivetrain. Use the above examples to help you.

If you do not have any other actuators you can skip this.

SENSORS

If you have any sensors you plan to use, such as a limit switch, you should set those up in the file they correspond to.

For example, a potentiometer telling you when your arm has rotated 90 degrees would go under mechanisms.

AUTONOMOUS

TODO 1. Enable the program

The first change you are going to make to your program is to comment out a section of code near the top of the program.

Each program can be enabled/disabled with a simple command. This will hide/show the program on the phone's lists as desired, allowing you to only see the programs you want to run at any given time.

Remember, that to comment out something, add **//** to the beginning of the line.

Your program should look like the following:

```
//////////////////////////////////////////////////
@Autonomous(name= "Autonomous", group="Auto")
//@Disabled
//////////////////////////////////////////////////
```

TODO 2. Write your autonomous code

The code you write in this section will be largely unique to you and the needs of your robot. Experiment and try different things and see what happens.

The code example below will drive the robot forwards for one second.

```
// Code Segment
drivetrain.setDrivePower(1, 1);
delay(1000);
drivetrain.stopDriving();
```

TELEOP

TODO 1. Enable the program

The first change you are going to make to your program is to comment out a section of code near the top of the program.

Each program can be enabled/disabled with a simple command. This will hide/show the program on the phone's lists as desired, allowing you to only see the programs you want to run at any given time.

Remember, that to comment out something, add **//** to the beginning of the line.

Your program should look like the following:

```
//////////////////////////////////////////
@TeleOp(name="TeleOpMode", group="teleop")
@Disabled
//////////////////////////////////////////
```

TODO 2. Write your teleop code

The code you write in this section will be largely unique to you and the needs of your robot. Experiment and try different things and see what happens.

Here are two code examples to help you get started.

```
// Code Segment
```

```
drivetrain.setDrivePower(gamepad1.          , gamepad1.          );

if (gamepad1.triangle) {
    mechanisms.setArmMotorPower(1);
}
```