Code a Drivable Robot

Base Project

**OI.java**

Discuss this file:

* used to define joysticks
* used to map buttons to commands.

// Add joystick with port 0.

**RobotMap.java**

Discuss this file:

* used to map ports to variables
* if ports get changed, they can be managed in this one place
* you can also set constants here: variables used throughout the robot project that don’t change

// Figure out which ports the motor controllers are plugged into (PWM or CAN)

// Add relevant motor controllers variables with appropriate port numbers.

**Robot.java**

Discuss this file:

* this is the hub of the robot
* all subsystem instantiations happen here
* items defined in the robot can be accessed with dot syntax like this: Robot.something.something

// If it bothers you, remove m\_ from variable names with find/replace. Find “m\_” replace with nothing.

## DriveSubsystem

**Create new DriveSubsystem**

Discuss this file:

* Subsystems represent hardware on the robot.
* This subsystem contains everything related to the drivetrain
* This includes motor controllers and sensors such as gyro and encoders
* This is just a class. You will need to instantiate a new version of this in Robot.java and assign it to a variable to actually use. In Robot.java, if you name your object driveSubsystem, then you can access the driveSubsystem from anywhere with dot syntax: Robot.driveSubsystem.anyMethodYouCreateInTheSubsystemClass

// instantiate new motor controller objects

// instantiate a new DifferentialDrive object pass motor controllers as arguments

// if using talon SRX motor controllers:

// create constructor function

// point slaves to masters

// add manualDrive() method

**Robot.java**

// Instantiate Drive\_Subsystem as a new object named: driveSubsystem

## ManualDriveCommand

**Create new ManualDriveCommand**

Discuss this file:

* Commands control subsystems
* Commands are typically mapped to buttons in the OI.java file
* We don’t need to map this command to a button because this command will be set as the default command in the DriveSubsystem.
* Discuss how the different methods in this command behave.

// require the driveSubsystem using the requires() method and dot syntax: Robot.driveSubsystem

// in execute

// add move from joystick as Robot.oi.stick.getY() (make sure stick is public in the oi file)

// not that forward on the stick is -1 and reverse is 1. Negate this value to compensate.

// add turn from joystick as Robot.oi.stick.getX() or getTwist() if using the hand stick

// call the manualDrive method or the DifferentialDrive object and pass it the move and turn values into it

//Robot.driveSubsystem.teleopDrive(move, turn)

**DriveSubsytem**

Update default command with a new instantiation of ManualDriveCommand()

STOP AND TEST WHETHER TELEOP WORKS

If it drives in reverse, check the drive station usb io section. Check if moving the joystick on it’s y access moves the green bar right or left. If pushing the stick forward moves the green bar left, then it’s passing a -1 value as forward. This can be negated in the ManualDriveCommand code by saying -Robot.oi.stick.getY().