KNO3 Robotics
PLANT HIGH SCHOOL

A Better Teleop

Jaxon Brown
August 5, 2017

FIRST RELIC RECOVERY

FIRST TECH CHALLENGE

Goals

- Discuss the overarching goals of teleop programming
- Create three different drive code options
  - Tank Drive, Arcade Drive, and Car Drive
- Simple mechanism control
  - Using digital and analog gamepad inputs
- Sensors in Teleop
- Put it all together

Goals in Teleop Programs

What should we be thinking about while we create our teleop programs?

Simple Controls

- Drivers need to think about more than driving
  - How much time is left?
  - Where are the scoring elements?
  - Where are other robots?
  - How do I get where I want to go?
- The answer? Simplify controls, not code!
  - More intuitive
  - Instead of individually controlling mechanisms, control the result

Automate Controls

- In order to control the result, we often need automation
- Oftentimes, this requires more complex algorithms
- Today, we will use sensors in teleop to show how easy this *can* be

Drive Code Options

How do we control the drive train?

Tank Drive

- Tank Drive is among the simplest and most common ways to control the drive train of your robot
- Each side of your robot will be controlled separately by two joysticks

Pros

- Simple and Easy
- Gives fine control over motion
- More?

Cons

- Abnormal outside of Robotics
- Hard to use
- More?

Tank Drive

- Create a new OpMode, define your hardware, then set it up in init()
- In loop(), create two double variables, ‘leftSpeed’ and ‘rightSpeed’, setting them equal to ‘gamepad1.left_stick_y’ and ‘gamepad1.right_stick_y’, respectively
- Use ‘*motor*.setPower(leftSpeed)’ to set the power of your drive motors

```java
@TeleOp(name = "Tank Drive")
public class TankDrive extends OpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;


    @Override
    public void init() {
        frontLeft = hardwareMap.dcMotor.get("frontleft");
        rearLeft = hardwareMap.dcMotor.get("rearleft");
        frontRight = hardwareMap.dcMotor.get("frontright");
        rearRight = hardwareMap.dcMotor.get("rearright");

        frontLeft.setDirection(DcMotorSimple.Direction.REVERSE);
        rearLeft.setDirection(DcMotorSimple.Direction.REVERSE);
    }

    @Override
    public void loop() {

    }
}
```

Create a OpMode and setup hardware for a drive train

```java
@Override
public void loop() {
    double leftSpeed = gamepad1.left_stick_y;
    double rightSpeed = gamepad1.right_stick_y;

}
```

Create double variables representing left and right speeds

```java
@Override
public void loop() {
    double leftSpeed = gamepad1.left_stick_y;
    double rightSpeed = gamepad1.right_stick_y;

    frontLeft.setPower(leftSpeed);
    rearLeft.setPower(leftSpeed);
    frontRight.setPower(rightSpeed);
    rearRight.setPower(rightSpeed);
}
```

Set the power of the motors

Arcade Drive

- Arcade Drive is allows you to control your entire drive train on a single joystick

Pros

- Fairly easy to implement
- Pretty intuitive
- More?

Cons

- Lost control over robot motion
- Easy to make mistakes driving
- More?

Arcade Drive

- Create a new OpMode, define your hardware, then set it up in init()
- In loop(), create two double variables, ‘speed’ and ‘turn’, setting them equal to ‘gamepad1.right_stick_y’ and ‘gamepad1.right_stick_x’, respectively
- Set the left motor’s power to ‘speed + turn’
- Set the right motor’s power to ‘speed - turn’

```java
@TeleOp(name = "Normal Arcade Drive")
public class NormalArcadeDrive extends OpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;


    @Override
    public void init() {
        frontLeft = hardwareMap.dcMotor.get("frontleft");
        rearLeft = hardwareMap.dcMotor.get("rearleft");
        frontRight = hardwareMap.dcMotor.get("frontright");
        rearRight = hardwareMap.dcMotor.get("rearright");

        frontLeft.setDirection(DcMotorSimple.Direction.REVERSE);
        rearLeft.setDirection(DcMotorSimple.Direction.REVERSE);
    }

    @Override
    public void loop() {
        double speed = gamepad1.right_stick_y;
        double turn = gamepad1.right_stick_x;
    }
}
```

Setup the OpMode and Hardware; create double variables representing forwards/reverse speed and turning speed

```
@Override
public void loop() {
    double speed = gamepad1.right_stick_y;
    double turn = gamepad1.right_stick_x;

    frontLeft.setPower(speed + turn);
    rearLeft.setPower(speed + turn);
    frontRight.setPower(speed - turn);
    rearRight.setPower(speed - turn);
}
```

Set the power of left motors to ‘speed + turn’ and the right motors to ‘speed – turn’

Car Drive

- Car Drive is similar to arcade drive but mixes up the controls and uses a more complicated system to determine left and right drive speeds

Pros

- Very intuitive controls
- Easy to drive and has fairly fine control
- More?

Cons

- More complicated code
- More?

Car Drive

- Create a new OpMode, define your hardware, then set it up in init()
- In loop(), create two double variables, ‘speed’ and ‘turn’, setting them equal to ‘gamepad1.right_stick_y’ and ‘gamepad1.right_stick_x’, respectively
- Set the left motors’ power to
  - ‘((1 - Math.abs(turn)) * speed + (1 - Math.abs(speed)) * turn + turn + speed) / 2’
- Set the right motors’ power to
  - ‘((1 - Math.abs(turn)) * speed - (1 - Math.abs(speed)) * turn - turn + speed) / 2’

```java
@TeleOp(name = "Car Game Drive")
public class CarGameDrive extends OpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;


    @Override
    public void init() {
        frontLeft = hardwareMap.dcMotor.get("frontleft");
        rearLeft = hardwareMap.dcMotor.get("rearleft");
        frontRight = hardwareMap.dcMotor.get("frontright");
        rearRight = hardwareMap.dcMotor.get("rearright");

        frontLeft.setDirection(DcMotorSimple.Direction.REVERSE);
        rearLeft.setDirection(DcMotorSimple.Direction.REVERSE);
    }

    @Override
    public void loop() {
        double speed = gamepad1.right_trigger - gamepad1.left_trigger;
        double turn = gamepad1.left_stick_x;
    }
}
```

Setup the OpMode; create two double variables for speed and turning speed

```java
@Override
public void loop() {
    double speed = gamepad1.right_trigger - gamepad1.left_trigger;
    double turn = gamepad1.left_stick_x;

    double leftSpeed = ((1 - Math.abs(turn)) * speed + (1 - Math.abs(speed)) * turn + turn + speed) / 2;
    double rightSpeed = ((1 - Math.abs(turn)) * speed - (1 - Math.abs(speed)) * turn - turn + speed) / 2;

    frontLeft.setPower(leftSpeed);
    rearLeft.setPower(leftSpeed);
    frontRight.setPower(rightSpeed);
    rearRight.setPower(rightSpeed);
}
```

Set left speeds to
‘((1-Math.abs(turn))*speed+(1-Math.abs(speed))*turn+turn+speed)/2’
Set right speeds to
‘((1-Math.abs(turn))*speed-(1-Math.abs(speed))*turn-turn+speed)/2’

Mechanisms Control

Controlling non-drive motors and servos

Controlling Mechanisms

Digital Controls

- On/Off states
- Especially useful on servos
- Switch between robot states
  - Drive in reverse?

Analog Controls

- Scale of states
- Usually used with motors

Digital Input Mechanism Control

- Create a new OpMode, define your hardware, then set it up in init()
  - We will use a DcMotor called ‘intake’, a servo called ‘servo’, a servo called ‘servo2’, and a continuous rotation called ‘crServo’
- In loop(), create an if statement with condition ‘gamepad2.a’ and body ‘intake.setPower(1);’
  - Add an else statement with body ‘intake.setPower(0);’
- This will make intake run at power 1 when gamepad2 a is pressed, and turn off when gamepad2 a is released

```java
@TeleOp(name = "Control Mechanisms, Digital")
public class ControlMechanismsDigital extends OpMode {
    private DcMotor intake;
    private Servo servo;
    private Servo servo2;
    private CRServo crServo;


    @Override
    public void init() {
        intake = hardwareMap.dcMotor.get("intake");
        servo = hardwareMap.servo.get("servo");
        servo.setPosition(0);
        servo2 = hardwareMap.servo.get("servo2");
        servo2.setPosition(0);
        crServo = hardwareMap.crservo.get("crservo");
    }

    @Override
    public void loop() {
        if(gamepad2.a) {
            intake.setPower(1);
        } else {
            intake.setPower(0);
        }
    }
}
```

Setup OpMode; create intake control if/else statement using gamepad.a

Digital Input Mechanism Control

- Add another if statement with ‘gamepad2.b’ as the condition and ‘servo.setPosition(1);’ as the body
  - Add an else statement with body ‘servo.setPosition(0);’
- This will move servo to position 1 as long as you hold down b, otherwise it will move back towards 0.
- Add another if statement with ‘gamepad2.x’ as the condition and ‘servo2.setPosition(1);’ as the body
- Add another if statement with ‘gamepad2.y’ as the condition and ‘servo2.setPosition(0);’ as the body
- This will move servo2 to position 1 when you press x, and it will stay there until you press y, then it will move back to 0

```
@Override
public void loop() {
    if(gamepad2.a) {
        intake.setPower(1);
    } else {
        intake.setPower(0);
    }


    if(gamepad2.b) {
        servo.setPosition(1);
    } else {
        servo.setPosition(0);
    }
}
```

Create servo control if/else statement using gamepad.b

```
if(gamepad2.x) {
    servo2.setPosition(1);
}
if(gamepad2.y) {
    servo2.setPosition(0);
}
```

Create servo2 control if statements using gamepad2.x and y

Digital Input Mechanism Control

- Add another if/else statement like the first one, but using ‘left_bumper’ instead of ‘a’ and ‘crServo’ instead of ‘intake’
- This will make crServo move at 100% power as long as you hold left_bumper

```
if(gamepad2.left_bumper) {
    crServo.setPower(1);
} else {
    crServo.setPower(0);
}
```

Create crServo control statement with gamepad2.left_bumper

Analog Input Mechanism Control

- Create a new OpMode, define your hardware, then set it up in init()
  - We will use a DcMotor called ‘intake’, a servo called ‘servo’, and a continuous rotation called ‘crServo’
- In loop(), create a double variable called ‘intakeSpeed’ and set it equal to ‘gamepad2.right_trigger’. You can then set the intake power to intakeSpeed. The intake will speed up the more you depress the right trigger
- We can do the same with a continuous rotation servo

```
private CRServo crServo;


@Override
public void init() {
    intake = hardwareMap.dcMotor.get("intake");
    servo = hardwareMap.servo.get("servo");
    servo.setPosition(0.5);
    crServo = hardwareMap.crservo.get("crservo");
}

@Override
public void loop() {
    double intakeSpeed = gamepad2.right_trigger;
    intake.setPower(intakeSpeed);
```

Create OpMode; create speed/power variables and store gamepad input, then set the power of the hardware

Analog Input Mechanism Control

- Create a double variable called ‘servoPosition’ and set it equal to ‘gamepad2.right_stick_y / 2 + 1’
- Here we want to move the servo to a position depending on where the joystick is positioned
- The ‘/ 2 + 1’ divides ‘gamepad2.right_stick_y’ by two and adds 1. Since ‘gamepad2.right_stick_y’ returns a value [-1, 1], we divide our bounds by 2, yielding [-0.5, 0.5], then add 1, [0, 1], resulting in bounds that match the [0, 1] we need to provide setPosition()

```java
@Override
public void loop() {
    double intakeSpeed = gamepad2.right_trigger;
    intake.setPower(intakeSpeed);

    double crServoPower = gamepad2.left_stick_y;
    crServo.setPower(crServoPower);

    double servoPosition = gamepad2.right_stick_y / 2 + 1;
    servo.setPosition(servoPosition);
}
```

To control a servo with a joystick, adjust the bound using ‘/ 2 + 1’

Sensors in Teleop

Using sensors to automate your Teleop

Sensors in Teleop

- Create a new OpMode, define your hardware, then set it up in init()
  - We will use a DcMotor called ‘intake’, and two ColorSensors named ‘intakeSensor1’ and ‘intakeSensor2’
- In loop(), create a double variable called ‘intakeSpeed’ and set it equal to ‘gamepad2.right_trigger - gamepad2.left_trigger’
- Skip a few lines, and set the power of ‘intake’ to ‘intakeSpeed’

```java
@TeleOp(name = "Sensors In Teleop")
public class SensorsInTeleop extends OpMode {
    private DcMotor intake;
    private ColorSensor intakeSensor1;
    private ColorSensor intakeSensor2;


    @Override
    public void init() {
        intake = hardwareMap.dcMotor.get("intake");

        intakeSensor1 = hardwareMap.colorSensor.get("intakeSensor1");
        intakeSensor2 = hardwareMap.colorSensor.get("intakeSensor2");

        intakeSensor1.setI2cAddress(I2cAddr.create8bit(0x5c));
        intakeSensor2.setI2cAddress(I2cAddr.create8bit(0x6c));
    }

    @Override
    public void loop() {
        double intakeSpeed = gamepad2.right_trigger - gamepad2.left_trigger;



        intake.setPower(intakeSpeed);
    }
}
```

Create OpMode; run the intake at the difference of the right and left triggers

Sensors in Teleop

- Create an if statement with body ‘intakeSpeed = 0;’
- We will now check the color sensors. If for either of them a condition is true, we want to stop the intake
- We will use the condition *red-3 >= blue*
- Our condition is then ‘intakeSensor1.red() - 3 >= intakeSensor1.blue() || intakeSensor2.red() - 3 >= intakeSensor2.blue()’

```java
@Override
public void loop() {
    double intakeSpeed = gamepad2.right_trigger - gamepad2.left_trigger;


    if(intakeSensor1.red() - 3 >= intakeSensor1.blue() || intakeSensor2.red() - 3 >= intakeSensor2.blue()) {
        intakeSpeed = 0;
    }


    intake.setPower(intakeSpeed);
}
```

Create an if statement to detect red balls and stop the intake

Sensors in Teleop

- However, while this will stop the intake if the sensors see red, it won't allow you to reverse the collector either.
- Remember that && comes before || in the Java order of operations
- We will put our existing condition in parentheses, then add '&& intakeSpeed > 0' after the parentheses containing our previous condition.

```java
@Override
public void loop() {
    double intakeSpeed = gamepad2.right_trigger - gamepad2.left_trigger;


    if((intakeSensor1.red() - 3 >= intakeSensor1.blue() || intakeSensor2.red() - 3 >= intakeSensor2.blue()) && intakeSpeed > 0) {
        intakeSpeed = 0;
    }


    intake.setPower(intakeSpeed);
}
```

Add a condition for ‘intakeSpeed > 0’ so that it won’t stop you from reversing the intake

Putting it all together

Putting it all together

- Create a new OpMode, define your hardware, then set it up in init()
  - We will use a 4 motor drive, a DcMotor called ‘intake’, two ColorSensors named ‘intakeSensor1’ and ‘intakeSensor2’, two DcMotors called ‘shooterLeft’ and ‘shooterRight’, and a Servo called ‘indexer’
- Start by choosing your drive code. I’ll use Car Drive. Include the code for Car Drive at the top of loop().
- Skip a few lines for readability.

```java
@TeleOp(name = "Robot Teleop")
public class RobotTeleop extends OpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;

    private DcMotor intake;
    private ColorSensor intakeSensor1;
    private ColorSensor intakeSensor2;

    private DcMotor shooterLeft;
    private DcMotor shooterRight;
    private Servo indexer;


    @Override
    public void init() {
        frontLeft = hardwareMap.dcMotor.get("frontleft");
        rearLeft = hardwareMap.dcMotor.get("rearleft");
        frontRight = hardwareMap.dcMotor.get("frontright");
        rearRight = hardwareMap.dcMotor.get("rearright");

        rearLeft.setDirection(DcMotorSimple.Direction.REVERSE);
        frontRight.setDirection(DcMotorSimple.Direction.REVERSE);


        intake = hardwareMap.dcMotor.get("intake");
        intakeSensor1 = hardwareMap.colorSensor.get("intakeSensor1");
        intakeSensor2 = hardwareMap.colorSensor.get("intakeSensor2");

        intakeSensor1.setI2cAddress(I2cAddr.create8bit(0x5c));
        intakeSensor2.setI2cAddress(I2cAddr.create8bit(0x6c));


        shooterLeft = hardwareMap.dcMotor.get("shooterLeft");
        shooterRight = hardwareMap.dcMotor.get("shooterRight");
        indexer = hardwareMap.servo.get("indexer");
        indexer.setPosition(0);

        shooterLeft.setDirection(DcMotorSimple.Direction.REVERSE);
        shooterLeft.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);
        shooterRight.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);
    }

    @Override
    public void loop() {

    }
}
```

Setup the OpMode

```
@Override
public void loop() {
    double speed = gamepad1.right_trigger - gamepad1.left_trigger;
    double turn = gamepad1.left_stick_x;
    double leftSpeed = ((1 - Math.abs(turn)) * speed + (1 - Math.abs(speed)) * turn + turn + speed) / 2;
    double rightSpeed = ((1 - Math.abs(turn)) * speed - (1 - Math.abs(speed)) * turn - turn + speed) / 2;

    frontLeft.setPower(leftSpeed);
    rearLeft.setPower(leftSpeed);
    frontRight.setPower(rightSpeed);
    rearRight.setPower(rightSpeed);



}
```

Insert drive code

Putting it all together

- Insert your intake code. You can use the digital mechanism control code, the analog mechanism control code, the sensors in teleop control code, or any alternative. I'll use the sensors in teleop code.
- Skip a few lines then create an if statement with condition 'gamepad2.x' and in the body set 'shooterLeft.setPower(1);' and 'shooterRight.setPower(1);'
  - Create an else statement and set shooterLeft and shooterRight to 0.
- Skip a few more lines and create another if statement with 'gamepad2.y' as the condition and 'indexer.setPosition(0.5);' as the body
  - Create an else statement and set the position of indexer to 0

```java
@Override
public void loop() {
    double speed = gamepad1.right_trigger - gamepad1.left_trigger;
    double turn = gamepad1.left_stick_x;
    double leftSpeed = ((1 - Math.abs(turn)) * speed + (1 - Math.abs(speed)) * turn + turn + speed) / 2;
    double rightSpeed = ((1 - Math.abs(turn)) * speed - (1 - Math.abs(speed)) * turn - turn + speed) / 2;

    frontLeft.setPower(leftSpeed);
    rearLeft.setPower(leftSpeed);
    frontRight.setPower(rightSpeed);
    rearRight.setPower(rightSpeed);



    double intakeSpeed = gamepad2.right_trigger - gamepad2.left_trigger;
    if(intakeSensor1.red() - 3 >= intakeSensor1.blue() || intakeSensor2.red() - 3 >= intakeSensor2.blue()) {
        intakeSpeed = 0;
    }
    intake.setPower(intakeSpeed);



}
```

Insert intake code

```java
@Override
public void loop() {
    double speed = gamepad1.right_trigger - gamepad1.left_trigger;
    double turn = gamepad1.left_stick_x;
    double leftSpeed = ((1 - Math.abs(turn)) * speed + (1 - Math.abs(speed)) * turn + turn + speed) / 2;
    double rightSpeed = ((1 - Math.abs(turn)) * speed - (1 - Math.abs(speed)) * turn - turn + speed) / 2;

    frontLeft.setPower(leftSpeed);
    rearLeft.setPower(leftSpeed);
    frontRight.setPower(rightSpeed);
    rearRight.setPower(rightSpeed);



    double intakeSpeed = gamepad2.right_trigger - gamepad2.left_trigger;
    if(intakeSensor1.red() - 3 >= intakeSensor1.blue() || intakeSensor2.red() - 3 >= intakeSensor2.blue()) {
        intakeSpeed = 0;
    }
    intake.setPower(intakeSpeed);



    if(gamepad2.x) {
        shooterLeft.setPower(1);
        shooterRight.setPower(1);
    } else {
        shooterLeft.setPower(0);
        shooterRight.setPower(0);
    }
}
```

Use digital mechanism control to control the shooter

```java
@Override
public void loop() {
    double speed = gamepad1.right_trigger - gamepad1.left_trigger;
    double turn = gamepad1.left_stick_x;
    double leftSpeed = ((1 - Math.abs(turn)) * speed + (1 - Math.abs(speed)) * turn + turn + speed) / 2;
    double rightSpeed = ((1 - Math.abs(turn)) * speed - (1 - Math.abs(speed)) * turn - turn + speed) / 2;

    frontLeft.setPower(leftSpeed);
    rearLeft.setPower(leftSpeed);
    frontRight.setPower(rightSpeed);
    rearRight.setPower(rightSpeed);



    double intakeSpeed = gamepad2.right_trigger - gamepad2.left_trigger;
    if(intakeSensor1.red() - 3 >= intakeSensor1.blue() || intakeSensor2.red() - 3 >= intakeSensor2.blue()) {
        intakeSpeed = 0;
    }
    intake.setPower(intakeSpeed);



    if(gamepad2.x) {
        shooterLeft.setPower(1);
        shooterRight.setPower(1);
    } else {
        shooterLeft.setPower(0);
        shooterRight.setPower(0);
    }


    if(gamepad2.y) {
        indexer.setPosition(0.5);
    } else {
        indexer.setPosition(0);
    }
}
```

Use digital mechanism control to control the indexer

Advanced Topics

We will briefly cover these more advanced topics

Threading in your teleop program

- We can actually create multiple concurrent threads in Java
- I used one to feed balls into the shooter while holding down a button
- When using threads make certain that they will terminate with the program or are short-lived
  - If your threads don't close you could possibly crash your app!
- You may also consider using threads for other automated tasks

PID Loops in Teleop

- If you choose to implement a PID loop in teleop, be careful with how you manage timing
- PID loops need to at least know how much time has elapsed between calls, or ideally always fire at a consistent rate
- The loop() method doesn't guarantee anything with respect to timing
  - Often fires more than once per millisecond, which might be faster than your sensors can give you useful data

Questions?

For further research, Google ‘Java Threading’ and ‘PID Loop’