KNO3 Robotics
PLANT HIGH SCHOOL

A Better Autonomous

Jaxon Brown

August 5, 2017

FIRST RELIC RECOVERY

FIRST TECH CHALLENGE

Why is autonomous important?

- Completing autonomous objectives is worth more
- 45% of points scored at Houston and St. Louis worlds were scored in autonomous
- Possible to continue the extra autonomous incentives

ITS EASY!

Goals

- Dead Reckoning
- Using Sensor Input
  - Color Sensors
  - Light Sensors
  - Encoders
  - Gyro Sensors
- Putting it all together

Dead Reckoning

And why you shouldn’t use it

What is Dead Reckoning?

- Dead Reckoning is when we use no sensor input to drive the robot
- Generally we will go forwards at a given power for a given amount of time
- This works for just-go-forwards autonomous programs
  - But in Velocity Vortex you could score 10-15 points only driving forwards!
- Since turning introduces extra friction with the ground, Dead Reckoning often falls short of useful in turns

Dead Reckoning Forwards

- Create a new LinearOpMode, define your hardware, then set it up at the beginning of runOpMode()
  - Make sure to include a waitForStart() before the rest of your code
- A very easy way to increase the accuracy of your autonomous is to set your motors to run with the built-in speed controlling PID loop.
  - '*motor*.setMode(DcMotor.RunMode.RUN_USING_ENCODER);'
  - Go ahead and do this to all of your drive motors
- Set the power of all motors to 1

```java
@Autonomous(name = "Dead Reckoning Forwards")
public class DeadReckoningForwards extends LinearOpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;

    @Override
    public void runOpMode() throws InterruptedException {
        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);
        frontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);



        waitForStart();



    }
}
```

Setup LinearOpMode with hardware and waitForStart()

```java
@Override
public void runOpMode() throws InterruptedException {
    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);
    frontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    rearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    rearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);



    waitForStart();



    frontLeft.setPower(1);
    rearLeft.setPower(1);
    frontRight.setPower(1);
    rearRight.setPower(1);
}
```

Set motor powers to 1

Dead Reckoning Forwards

- We now need to wait some time, then stop moving. To wait some time, we can use ‘Thread.sleep(*time*)’ where time is the number of milliseconds to wait
- We will use ‘Thread.sleep(1000);’ to wait 1 second
- Finally, stop all of the motors by setting their powers to 0

```
    waitForStart();



    frontLeft.setPower(1);
    rearLeft.setPower(1);
    frontRight.setPower(1);
    rearRight.setPower(1);

    Thread.sleep(1000);

    frontLeft.setPower(0);
    rearLeft.setPower(0);
    frontRight.setPower(0);
    rearRight.setPower(0);
}
```

Wait (sleep) for 1000ms then stop all motors

Dead Reckoning Turn

- You can turn with dead reckoning using the same code as going forwards, except setting the power for motors on one side of the robot to negative numbers
- I would not recommend dead reckoning turns at all, as due to a number of factors they are very inaccurate

```java
@Autonomous(name = "Dead Reckoning Turn")
public class DeadReckoningTurn extends LinearOpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;

    @Override
    public void runOpMode() throws InterruptedException {
        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);
        frontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);



        waitForStart();



        frontLeft.setPower(1);
        rearLeft.setPower(1);
        frontRight.setPower(-1);
        rearRight.setPower(-1);

        Thread.sleep(1000);

        frontLeft.setPower(0);
        rearLeft.setPower(0);
        frontRight.setPower(0);
        rearRight.setPower(0);
    }
}
```

A dead reckoning turn

Servos in Autonomous

- Using servos on your mechanisms in autonomous generally doesn't require the use of sensors
- However, you need to remember that servos take some time to move, so be sure to wait for that in your programs

```java
@Autonomous(name = "Auto Servos")
public class AutoServos extends LinearOpMode {
    private Servo pusher;

    @Override
    public void runOpMode() throws InterruptedException {
        pusher = hardwareMap.servo.get("pusher");
        pusher.setPosition(0);



        waitForStart();



        pusher.setPosition(1);
        Thread.sleep(2000);
        pusher.setPosition(0);
        Thread.sleep(2000);
    }
}
```

Set servo positions and wait in between

Sensors in Autonomous

Color Sensors, Light Sensors, Encoders, and (self-integrating) Gyro Sensors

Reading Sensors in Autonomous

- We read sensors just like we do in Teleop
- Create a new LinearOpMode, don't forget waitForStart()
  - We will create a 4 motor drive train and a ColorSensor called 'colorSensor'
- Create an if statement with condition 'colorSensor.red() > colorSensor.blue()' and in the body set the drive motors to 1
  - Create an else statement where you set the drive train motors to -1
- Use 'Thread.sleep(1000);' to wait for 1 second
- Stop all the motors
- If the sensor sees more red than blue when start is pressed on the Driver Station, the robot will drive forwards for a second. If it sees blue, it will drive backwards

```java
@Autonomous(name = "Sensor Conditional")
public class SensorConditional extends LinearOpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;

    private ColorSensor colorSensor;

    @Override
    public void runOpMode() throws InterruptedException {
        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);
        frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);


        colorSensor = hardwareMap.colorSensor.get("color");
        colorSensor.setI2cAddress(I2cAddr.create8bit(0x6c));



        waitForStart();



    }
}
```

Setup LinearOpMode and hardware; don’t forget to set the I2C Address of the Color Sensor

```
    waitForStart();



    if(colorSensor.red() > colorSensor.blue()) {
        frontLeft.setPower(1);
        rearLeft.setPower(1);
        frontRight.setPower(1);
        rearRight.setPower(1);
    } else {
        frontLeft.setPower(-1);
        rearLeft.setPower(-1);
        frontRight.setPower(-1);
        rearRight.setPower(-1);
    }

    Thread.sleep(1000);

    frontLeft.setPower(0);
    rearLeft.setPower(0);
    frontRight.setPower(0);
    rearRight.setPower(0);
}
```

Create an if/else statement to control forwards/backwards motion then wait before stopping the motors

Forwards until Sensor

- Create a new LinearOpMode, don't forget waitForStart()
  - We will use a drive train and an OpticalDistanceSensor called 'lightSensor'
- Start by setting all of the motors to 1
- Create a while loop with condition 'lightSensor.getLightDetected() < .4' and an empty body
- Stop all motors
- This will drive forwards until the condition is false (eg, the light detected goes above .4)

```java
@Autonomous(name = "Forwards To Sensor")
public class ForwardsToSensor extends LinearOpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;

    private OpticalDistanceSensor lightSensor;

    @Override
    public void runOpMode() throws InterruptedException {
        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);
        frontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);


        lightSensor = hardwareMap.opticalDistanceSensor.get("light");
        lightSensor.enableLed(true);



        waitForStart();



    }
}
```

Setup LinearOpMode and hardware; light sensor (OpticalDistanceSensor) this time

```
    waitForStart();



    frontLeft.setPower(1);
    rearLeft.setPower(1);
    frontRight.setPower(1);
    rearRight.setPower(1);

    while(lightSensor.getLightDetected() < .4) {

    }

    frontLeft.setPower(0);
    rearLeft.setPower(0);
    frontRight.setPower(0);
    rearRight.setPower(0);
}
```

Drive forwards while sensor sees dark surface, then stop

Forwards using Encoders

- Create a new LinearOpMode, don't forget waitForStart()
  - We will use a 4 motor drive train
- Start by setting all of the motors to 1
- Create two integer variables called 'target' and 'current'
  - You don't need to give current a value yet
  - Set target equal to an integer number representing encoder ticks (I'll use 500)
- You can use specs from andymark's website, along with your gear ratios and the circumference of the wheel to predict encoder ticks, however, the best way to do it is usually testing it

```java
@Autonomous(name = "Forwards To Encoder")
public class ForwardsToEncoder extends LinearOpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;

    @Override
    public void runOpMode() throws InterruptedException {
        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);
        frontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);



        waitForStart();



    }
}
```

Create LinearOpMode and setup the hardware

```
    waitForStart();



    frontLeft.setPower(1);
    rearLeft.setPower(1);
    frontRight.setPower(1);
    rearRight.setPower(1);

    int target = 500;
    int current;

}
```

Set all motors to full power and create two integer variables called target and current. Set target equal to 500 and don't give current a value yet

Forwards using Encoders

- Create a do/while loop, with the condition as ‘current < target’
  - Set ‘current’ equal to
    - ‘leftMotor.getCurrentPosition() + rightMotor.getCurrentPosition()’
  - Divide ‘current’ by 2 (to find the average) ‘current /= 2;’
- After the do/while loop, stop the motors
- This will drive forwards, repeatedly updating current with the average encoder position then comparing it to target. When current becomes greater than target, the robot will stop

```
waitForStart();



frontLeft.setPower(1);
rearLeft.setPower(1);
frontRight.setPower(1);
rearRight.setPower(1);

int target = 500;
int current;
do {
    current = frontLeft.getCurrentPosition() + frontRight.getCurrentPosition();
    current /= 2;
} while(current < target);

frontLeft.setPower(0);
rearLeft.setPower(0);
frontRight.setPower(0);
rearRight.setPower(0);
```

Create a do/while loop, setting current to the average of one encoder from each side before comparing it to the target. Then stop the motors

Turn to Gyro Position

- Create a new LinearOpMode, don't forget waitForStart()
  - We will use a drive train and an GyroSensor called 'gyroSensor'
- Start by setting the left motors to 1 and the right motors to 0
- Create a while loop with condition 'Math.abs(targetAngle - gyroSensor.getHeading()) > 2' and an empty body
- After the while loop, stop the motors
- When we subtract the heading from the target, we get our error
- We take the absolute value to compare it to a tolerance
- When the error is less than ±2, exit the loop and stop the robot

```java
@Autonomous(name = "Turn To Gyro Pos")
public class TurnToGyroPos extends LinearOpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;

    private GyroSensor gyroSensor;

    @Override
    public void runOpMode() throws InterruptedException {
        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);
        frontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);


        gyroSensor = hardwareMap.gyroSensor.get("gyro");
        gyroSensor.resetZAxisIntegrator();



        waitForStart();



    }
}
```

Create a LinearOpMode and set up a gyro sensor

```
waitForStart();



frontLeft.setPower(1);
rearLeft.setPower(1);
frontRight.setPower(-1);
rearRight.setPower(-1);

double targetAngle = 90;
while(Math.abs(targetAngle - gyroSensor.getHeading()) > 2) {

}

frontLeft.setPower(0);
rearLeft.setPower(0);
frontRight.setPower(0);
rearRight.setPower(0);
```

Turn right, then wait until the difference between the target angle and the current heading is less than 2 before stopping

Putting it all together

Putting it all together

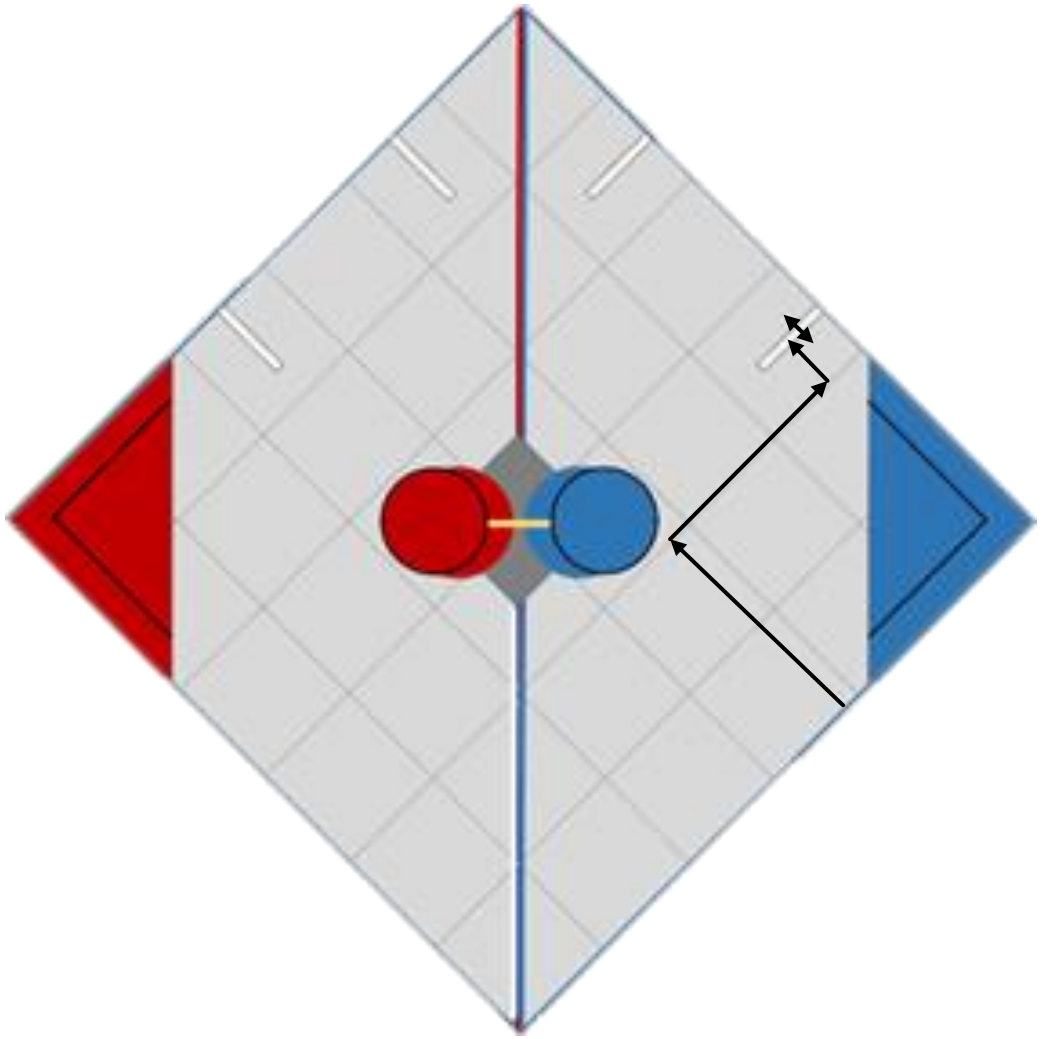
- Create a new LinearOpMode, don't forget waitForStart()
  - We will use a drive train, a Servo called 'pusher', a GyroSensor called 'gyroSensor', a OpticalDistanceSensor called 'lightSensor' and a ColorSensor called 'colorSensor'
- We now have a few building blocks available to us
  - Go forwards n encoder ticks
  - Turn to θ degrees
  - Act differently depending on sensor state
  - Drive until sensor is seen
  - Move servos

```java
@Autonomous(name = "Beacon Auto")
public class BeaconAuto extends LinearOpMode {
    private DcMotor frontLeft;
    private DcMotor rearLeft;
    private DcMotor frontRight;
    private DcMotor rearRight;

    private Servo pusher;

    private GyroSensor gyroSensor;
    private OpticalDistanceSensor lightSensor;
    private ColorSensor colorSensor;

    @Override
    public void runOpMode() throws InterruptedException {
        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);
        frontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        rearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);


        pusher = hardwareMap.servo.get("pusher");
        pusher.setPosition(0);


        gyroSensor = hardwareMap.gyroSensor.get("gyro");
        gyroSensor.resetZAxisIntegrator();

        lightSensor = hardwareMap.opticalDistanceSensor.get("light");
        lightSensor.enableLed(true);

        colorSensor = hardwareMap.colorSensor.get("color");
        colorSensor.setI2cAddress(I2cAddr.create8bit(0x6c));


        waitForStart();
    }
}
```

Setup the LinearOpMode and the hardware

Planning Autonomous

- Draw a path!
- Normally you want to optimize something (like speed), but we will optimize simplicity for now



Putting it all together

- Use the code from Forwards with Encoders to go forwards 200 ticks
  - We will add to the value of target ‘(frontLeft.getCurrentPosition() + frontRight.getCurrentPosition()) / 2’ to account for any previous movement before moving forwards
- Always wait some small amount of time between instructions. I’ll use 500ms since we aren’t pressed for time
  - ‘Thread.sleep(500);’
- Use the code from Turn to Gyro Position to turn right 90 degrees

```
waitForStart();


frontLeft.setPower(1);
rearLeft.setPower(1);
frontRight.setPower(1);
rearRight.setPower(1);

int target = 200 + (frontLeft.getCurrentPosition() + frontRight.getCurrentPosition()) / 2;
int current;
do {
    current = frontLeft.getCurrentPosition() + frontRight.getCurrentPosition();
    current /= 2;
} while(current < target);

frontLeft.setPower(0);
rearLeft.setPower(0);
frontRight.setPower(0);
rearRight.setPower(0);


Thread.sleep(500);


frontLeft.setPower(1);
rearLeft.setPower(1);
frontRight.setPower(-1);
rearRight.setPower(-1);

double targetAngle = 90;
while(Math.abs(targetAngle - gyroSensor.getHeading()) > 2) {

}

frontLeft.setPower(0);
rearLeft.setPower(0);
frontRight.setPower(0);
rearRight.setPower(0);


Thread.sleep(500);
```

Go forwards 200 ticks, sleep for .5sec, turn right 90 degrees, then sleep again

Putting it all together

- Use the code from Forwards with Encoders to go forwards 180 ticks
- Use the code from Turn to Gyro Position to turn left 90 degrees back to 0 degrees
  - There is a special case using the gyro turning to 0 degrees. If your tolerance crosses the jump from 360 to 0, you'll need to check both sides manually instead of using an absolute value
  - We will change the condition to 'gyroSensor.getHeading() < 358 && gyroSensor.getHeading() > 2' to account for this

```
frontLeft.setPower(1);
rearLeft.setPower(1);
frontRight.setPower(1);
rearRight.setPower(1);

target = 180 + (frontLeft.getCurrentPosition() + frontRight.getCurrentPosition()) / 2;
do {
    current = frontLeft.getCurrentPosition() + frontRight.getCurrentPosition();
    current /= 2;
} while(current < target);

frontLeft.setPower(0);
rearLeft.setPower(0);
frontRight.setPower(0);
rearRight.setPower(0);


Thread.sleep(500);


frontLeft.setPower(-1);
rearLeft.setPower(-1);
frontRight.setPower(1);
rearRight.setPower(1);

while(gyroSensor.getHeading() < 358 && gyroSensor.getHeading() > 2) {

}

frontLeft.setPower(0);
rearLeft.setPower(0);
frontRight.setPower(0);
rearRight.setPower(0);


Thread.sleep(500);
```

Drive forwards 180 ticks then turn left back to 0 degrees

Putting it all together

- Use the code from Forwards until Sensor to proceed to the white line using the light sensor
- Use the code from Reading Sensors in Autonomous to go forwards or backwards depending on the observed color
- Use the code from Servos in Autonomous to press the button

```
frontLeft.setPower(1);
rearLeft.setPower(1);
frontRight.setPower(1);
rearRight.setPower(1);

while(lightSensor.getLightDetected() < .4) {

}

frontLeft.setPower(0);
rearLeft.setPower(0);
frontRight.setPower(0);
rearRight.setPower(0);


Thread.sleep(500);


if(colorSensor.red() > colorSensor.blue()) {
    frontLeft.setPower(1);
    rearLeft.setPower(1);
    frontRight.setPower(1);
    rearRight.setPower(1);
} else {
    frontLeft.setPower(-1);
    rearLeft.setPower(-1);
    frontRight.setPower(-1);
    rearRight.setPower(-1);
}

Thread.sleep(1000);

frontLeft.setPower(0);
rearLeft.setPower(0);
frontRight.setPower(0);
rearRight.setPower(0);


pusher.setPosition(1);
Thread.sleep(2000);
pusher.setPosition(0);
Thread.sleep(2000);
```

Go forwards until the ground turns white, then compare the color. If it's red, go forwards, otherwise go backwards. Wait a second, stop, then push the button

Advanced Topics

We will briefly cover a few more advanced topics

Cleaning up with methods

- We can use methods with parameters to clean up repeated code. For example, a method for Forwards using Encoders might look like the code on the next slide
- Note that if you need to use Thread.sleep() you'll need to add 'throws InterruptedException' to your method definition

```java
@Override
public void runOpMode() throws InterruptedException {
    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);
    frontLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    rearLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    frontRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    rearRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);


    waitForStart();


    driveUsingEncoders(0.7, 200);
}

private void driveUsingEncoders(double power, int ticks) {
    frontLeft.setPower(power);
    rearLeft.setPower(power);
    frontRight.setPower(power);
    rearRight.setPower(power);

    int target = ticks + (frontLeft.getCurrentPosition() + frontRight.getCurrentPosition()) / 2;
    int current;
    do {
        current = frontLeft.getCurrentPosition() + frontRight.getCurrentPosition();
        current /= 2;
    } while(current < target);

    frontLeft.setPower(0);
    rearLeft.setPower(0);
    frontRight.setPower(0);
    rearRight.setPower(0);
}
```

Using a method to go forwards 200 ticks at .7 power using a method

PID Loops

- Figure these out in autonomous and you'll be set!
- By setting your motor modes to RUN_WITH_ENCODER, you're already using one PID loop
- Another easy one is to control your speed with your encoder error, as this can make your movement more precise and tolerant to battery voltage
- The most useful one is to control motor speeds with gyro error
- All PID loops require many hours in tuning to perfect

Using Inheritance

- You can create autonomous programs without the @Autonomous annotation, and remove the ‘private’ from your hardware definitions
- These programs won’t run on their own, but if you create another class that extends the first class, give *that* one the annotation, don’t include hardware fields, then do ‘super.runOpMode();’ as the first line of this class’ runOpMode(), any code you write after that will execute after the first class’ runOpMode()
- This allows you to create multiple endings to an autonomous program without adding maintenance effort if you need to change something later

KNO3 Robotics
PLANT HIGH SCHOOL

Questions?

Autonomous really isn't that bad, and in a pinch, you can always drive forwards!

FIRST RELIC RECOVERY

FIRST TECH CHALLENGE