Chapter 1

Robot Code

Robot.java

```java
package frc.robot;

import edu.wpi.first.wpilibj.TimedRobot;
import frc.robot.Mode.Autonomous;
import frc.robot.Mode.Disabled;
import frc.robot.Mode.Simulation;
import frc.robot.Mode.Teleop;
import frc.robot.Mode.Test;
import frc.robot.Mode.Onabot;

public class Robot extends TimedRobot {
  @Override public void robotInit           () { Onabot     .Initialize(); }
  @Override public void robotPeriodic       () { Onabot     .Periodic();   }

  @Override public void autonomousInit      () { Autonomous .Initialize(); }
  @Override public void autonomousPeriodic  () { Autonomous .Periodic();   }

  @Override public void disabledInit        () { Disabled   .Initialize(); }
  @Override public void disabledPeriodic    () { Disabled   .Periodic();   }

  @Override public void teleopInit          () { Teleop     .Initialize(); }
  @Override public void teleopPeriodic      () { Teleop     .Periodic();   }

  @Override public void testInit            () { Test       .Initialize(); }
  @Override public void testPeriodic        () { Test       .Periodic();   }

  @Override public void simulationInit      () { Simulation .Initialize(); }
  @Override public void simulationPeriodic  () { Simulation .Periodic();   }
}
```

Onabot.java

```
1  package frc.robot.Mode;
2
3  import frc.robot.Hardware.AutonChooser;
4  import frc.robot.Hardware.Driver;
5  import frc.robot.Hardware.Elevator;
6  import frc.robot.Hardware.Navigation;
7  import frc.robot.Hardware.Stage;
8  import frc.robot.Hardware.Swerve;
9
10 public class Onabot {
11
12     public static void Initialize () {
13         AutonChooser  .Initialize();
14         Driver        .Initialize();
15         Elevator      .Initialize();
16         Navigation    .Initialize();
17         Swerve        .Initialize();
18
19         // Lidar      .Initialize();
20         // Sonar      .Initialize();
21         // Vision     .Initialize();
22     }
23
24     public static void Periodic () {
25         AutonChooser  .Display();
26         Driver        .Display();
27         Elevator      .Display();
28         Navigation    .Display();
29         Swerve        .Display();
30
31         // Lidar      .Display();
32         // Sonar      .Display();
33         // Vision     .Display();
34
35         Stage.Display();
36     }
37
38 }
```

Autonomous.java

```
1  package frc.robot.Mode;
2
3  import frc.robot.Hardware.AutonChooser;
4  import frc.robot.Hardware.Autopilot;
5  import frc.robot.Hardware.Elevator;
6  import frc.robot.Hardware.Stage;
7  import frc.robot.Hardware.Swerve;
8  import frc.robot.Hardware.Path;
9
10 public class Autonomous {
11
12     public static String
13         AutonChoice = "Do Nothing";
14
15     public static void Initialize () {
16         AutonChoice = AutonChooser.chooser.getSelected();
17         Stage.Initialize();
18     }
19
20     public static void Periodic () {
21         Stage.Begin();
22
23         switch ( AutonChoice ) {
24             case "Nothing" : Path.Nothing(); break;
25             case "Any 1st" : Path.Any_1st(); break;
26             case "Any 2nd" : Path.Any_2nd(); break;
27             case "Lft 1st" : Path.Lft_1st(); break;
28             case "Lft 2nd" : Path.Lft_2nd(); break;
29             case "Ctr 1st" : Path.Ctr_1st(); break;
30             case "Ctr 2nd" : Path.Ctr_2nd(); break;
31             case "Rgt 1st" : Path.Rgt_1st(); break;
32             case "Rgt 2nd" : Path.Rgt_2nd(); break;
33         }
34
35         Stage.Next();
36
37         // EXECUTE COMMANDS
38         Elevator.Periodic();
39         Swerve.UpdateRobotRelative( Autopilot.vx, Autopilot.vy, Autopilot.vt );
40     }
41
42 }
```

Teleop.java

```
1  package frc.robot.Mode;
2
3  import edu.wpi.first.wpilibj.Joystick;
4  import edu.wpi.first.wpilibj.XboxController;
5  import frc.robot.Hardware.Driver;
6  import frc.robot.Hardware.Elevator;
7  import frc.robot.Hardware.Settings;
8
9  public class Teleop {
10
11     public static Joystick       DriveStick;
12     public static XboxController ManipStick;
13
14     public static double
15         Xratio,
16         Yratio,
17         Tratio;
18
19     public static void Initialize () {
20         DriveStick = new Joystick(       Settings.DriveStickID );
21         ManipStick = new XboxController( Settings.ManipStickID );
22     }
23
24     public static void Periodic () {
25         Xratio = -DriveStick.getY();
26         Yratio = -DriveStick.getX();
27         Tratio = -DriveStick.getTwist();
28
29         // UPDATE ALL COMPONENTS
30         Driver   .Periodic();
31         Elevator .Periodic();
32     }
33
34     public static void Display () {
35     }
36 }
```

Autopilot.java

The Autopilot methods are used in Autonomous mode to set the chassis speed variable found in this class. Values are sent to motor controllers in Autonomous.Periofic().

```
1  package frc.robot.Hardware;
2
3  public class Autopilot {
4
5      public static double LastHeading = 0;
6
7      public static double vx = 0;
8      public static double vy = 0;
9      public static double vt = 0;
10
11 //
12 // HeadingDiff is a simple method that calculates the angle difference
13 // between the current and desired heading. This can be used anywhere.
14 //
15     public static double HeadingDiff ( double SP ) {
16
17         // CALCULATE TURN VALUES
18           double PV = Navigation.GetDirection(); // Current state (Initial)
19                SP = ( SP + 360 ) % 360;          // Ensure SP is between 0 and 360
20                double diff = -( SP - PV );       // Why is this negated? Should setInverted have been used?
21
22           // SMALLEST ANGLE TO SWIVEL: -180 to 180
23           double minTurn = ( diff + 180 ) % 360 - 180;
24         return minTurn;
25     }
26
27 //
28 // This is a simple method for driving somewhat straight without using
29 // a gyroscope. There may be situations where it is good enough.
30 //
31     public static void DriveSortaStraight ( double Vx, double Vy ) {
32         vx = Vx; vy = Vy; vt = 0;
33     }
34
35     // Consider turning this into a pseudo tank drive for purposes of
36     // driving in a straigh line using the gyroscope.
37     public static void DriveStraight ( double Vx, double Vy ) {
38         vx = Vx; vy = Vy; vt = 0;
39     }
40
41 //
42 //
43 //
44     public static void DriveNorth ( double Speed ) {
45         vx = +Speed; vy = 0; vt = 0;
46     }
47
48     public static void DriveSouth ( double Speed ) {
49         vx = -Speed; vy = 0; vt = 0;
50     }
51
52     public static void DriveWest ( double Speed ) {
53         vx = 0; vy = +Speed; vt = 0;
54     }
55
56     public static void DriveEast ( double Speed ) {
57         vx = Speed; vy = -Speed; vt = 0;
58     }
59
60 //
61 //
62 //
63     public static void DriveNorthWest ( double Speed ) {
```

```
64        double radians = Math.toRadians( 45 );
65        double speed   = Speed * Math.cos( radians );
66        vx = +speed; vy = +speed; vt = 0;
67     }
68
69     public static void DriveNorthEast ( double Speed ) {
70        double radians = Math.toRadians( 45 );
71        double speed   = Speed * Math.cos( radians );
72        vx = +speed; vy = -speed; vt = 0;
73     }
74
75     public static void DriveSouthWest ( double Speed ) {
76        double radians = Math.toRadians( 45 );
77        double speed   = Speed * Math.cos( radians );
78        vx = -speed; vy = +speed; vt = 0;
79     }
80
81     public static void DriveSouthEast ( double Speed ) {
82        double radians = Math.toRadians( 45 );
83        double speed   = Speed * Math.cos( radians );
84        vx = -speed; vy = -speed; vt = 0;
85     }
86
87  //
88  // TurnToHeading sets the turn power variable in Autonomous mode to reach
89  // the desired heading using the shortest wheek swivel.
90  //
91     public static void TurnToHeading ( double NewHeading ) {
92        double minTurn = HeadingDiff( NewHeading );
93        double turnMag = Math.abs   ( minTurn );
94        double turnDir = Math.signum( minTurn );
95
96          // MINIMIZE WHEEL SWIVEL: +120 becomes -60
97          if ( turnMag > 90 ) {
98              turnMag  = 180 - turnMag; // Turn smaller angle
99              turnDir  = -1;            // and reverse swivel
100         }
101
102         // DETERMINE POWER USING PSEUDO PID CONTROLLER
103         if      ( turnMag > 20 ) { vt = 0.15; }
104         else if ( turnMag > 10 ) { vt = 0.08; }
105         else if ( turnMag >  1 ) { vt = 0.08; }
106         else                     { vt = 0.00; }
107
108       LastHeading = NewHeading;
109       vx = 0; vy = 0; vt *= turnDir;
110    }
111
112 //
113 // Stop sets the robot speed vector to zero. This is useful only in Autonomous
114 // mode. It should not be used elsewhere.
115 //
116    public static void Stop () {
117       vx = 0; vy = 0; vt = 0;
118    }
119
120 //
121 // These methods rotate the robot at a constant counter-clockwise speed and
122 // clockwise speed respectively. This is only useful in Autonomous mode.
123 //
124    public static void TurnLeftAtSpeed ( double Speed ) {
125       vx = 0; vy = 0; vt = +Speed;
126    }
127
128    public static void TurnRightAtSpeed ( double Speed ) {
129       vx = 0; vy = 0; vt = -Speed;
130    }
131
```

```
132     // public static void AdjustTurnSpeed( double Speed ) {
133     //     // double error = Speed - Navigation.GetTurnSpeed();
134     //     // LastPowerT  += error * 0.0001;
135     // }
136
137     // public static void DriveStraight ( double Speed, double Heading ) {
138
139     // }
140
141 }
```

Driver.java

```java
package frc.robot.Hardware;

import frc.robot.Driver.RobotRelative;

public class Driver {

    // public static String SelectedDriver = "RobotRel";

    // public static final String kRobotRel = "RobotRelative";
    // public static final String kFieldRel = "FieldRelative";
    // public static final String kAubrey   = "Aubrey";
    // public static final String kNate     = "Nate";
    // public static final String kSteensma = "Steensma";
    // public static final SendableChooser<String> chooser = new SendableChooser<>();

    public static void Initialize () {
        // chooser.setDefaultOption("RobotRel", kRobotRel );
        // chooser.addOption       ("FieldRel", kFieldRel );
        // chooser.addOption       ("Aubrey",   kAubrey   );
        // chooser.addOption       ("Nate",     kNate     );
        // chooser.addOption       ("Steensma", kSteensma );
        // SmartDashboard.putData  ("DRIVER",   chooser   );
    }

    public static void Periodic () {
        RobotRelative.Periodic();
        // switch ( "RobotRel" ) {
        //     case "RobotRel" : RobotRelative  .Periodic();
        //     case "FieldRel" : FieldRelative  .Periodic();
        //     case "Aubrey"   : Aubrey         .Periodic();
        //     case "Nate"     : Nate           .Periodic();
        //     case "Steensma" : Steensma       .Periodic();
        // }
    }

    public static void Display () {
        // SmartDashboard.putString("Driver", chooser.getSelected() );
    }

}
```

Elevator.java

```
1  package frc.robot.Hardware;
2
3  import frc.robot.Elevator.Arm;
4  import frc.robot.Elevator.Claw;
5  import frc.robot.Elevator.Lift;
6  import frc.robot.Elevator.Roller;
7
8  public class Elevator {
9
10     public static void Initialize () {
11         Arm            .Initialize();
12         Claw           .Initialize();
13         Lift           .Initialize();
14         Roller         .Initialize();
15     }
16
17     public static void Periodic () {
18         Arm            .Periodic();
19         Claw           .Periodic();
20         Lift           .Periodic();
21         Roller         .Periodic();
22     }
23
24     public static void Display () {
25         Arm            .Display();
26         Claw           .Display();
27         Lift           .Display();
28         Roller         .Display();
29     }
30
31     public static void Reset () {
32         Arm            .Reset(); // SetHI
33         Claw           .Reset(); // Drop
34         Lift           .Reset(); // SetLO
35         Roller         .Reset(); // Stop
36     }
37
38 //
39 //
40 //
41     public static void Preset1 () {
42         Lift    .SetLO();
43     }
44
45     public static void Preset2 () {
46         Lift    .SetLO();
47     }
48
49     public static void Preset3 () {
50         Lift    .SetHI();
51     }
52
53 }
```

Arm.java

```
1  package frc.robot.Elevator;
2
3  import com.ctre.phoenix.motorcontrol.ControlMode;
4  import com.ctre.phoenix.motorcontrol.can.TalonSRX;
5
6  import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
7  import frc.robot.Hardware.Settings;
8
9  public class Arm {
10
11     public static TalonSRX
12         Arm;
13
14     public static double
15         angle,
16         direction,
17         displacement = 0,
18         LO           = 0,
19         MD           = 60,
20         HI           = 90,
21         maximum      = -22000,
22
23         PosPV = 0, PosSP = HI, PosER = 0,
24         VelPV = 0, VelSP =  0, VelER = 0,
25
26         kF           = 0,
27         tolerance = 5, // degrees
28
29         Power        = 0;
30
31 //
32 //
33 //
34     public static void Initialize () {
35         Arm = new TalonSRX( Settings.Arm_CANID );
36         Arm  .setInverted( true );
37         Arm  .setSelectedSensorPosition( 90 );
38     }
39
40     public static void Periodic () {
41
42         // POSITION VALUES
43         PosPV = GetPosition();
44         PosER = PosSP - PosPV;
45         VelPV = GetVelocity();
46
47         // VELOCITY SET POINT (Depends on PosER)
48         if ( PosER > 0 ) { VelSP = +1200; }
49         if ( PosER < 0 ) { VelSP = -700; }
50
51         // ADJUST VELOCITY
52         VelER = VelSP - VelPV;
53         Power += 0.00005 * VelER;
54
55         // MAXIMUM POWER
56         if ( Power < -0.90 ) { Power = -0.90; }
57         if ( Power > +0.90 ) { Power = +0.90; }
58
59         // CUT POWER WHEN WITHIN RANGE
60         if ( Math.abs(PosER) < 5 ) { Power = 0; }
61         if ( PosSP == LO & PosPV < LO ) { Power = 0; }
62
63         // SET POWER
64         Arm.set( ControlMode.PercentOutput, Power );
65     }
```

```
66
67      public static void Display () {
68          SmartDashboard.putNumber("Arm Pos PV", GetPosition() );
69          SmartDashboard.putNumber("Arm Pos SP", PosSP         );
70
71          SmartDashboard.putNumber("Arm Vel PV", GetVelocity() );
72          SmartDashboard.putNumber("Arm Vel SP", VelSP         );
73
74          SmartDashboard.putNumber("ARM POWER", Power );
75          SmartDashboard.putNumber("Arm PosER", PosER );
76          SmartDashboard.putNumber("Arm VelER", VelER );
77      }
78
79  //
80  // POSITION        VELOCITY
81  //  0 horizontal  + is movign up
82  // 90 vertical    - is moving down
83      public static double GetPosition () {
84          double PV = Arm.getSelectedSensorPosition();
85          PosPV = 90 - PV / maximum * 90;
86          return PosPV;
87      }
88
89      public static double GetVelocity () {
90          VelPV = Arm.getSelectedSensorVelocity();
91          return VelPV;
92      }
93
94      public static void SetPosition ( double pos ) {
95          PosSP = pos;
96      }
97
98  //
99  //
100 //
101     public static void Reset () {
102         SetHI();
103     }
104
105     public static void SetHI () {
106         Power = 0.35;
107         PosSP = HI;
108     }
109
110     public static void SetMD () {
111         Power = 0.00;
112         PosSP = MD;
113     }
114
115     public static void SetLO () {
116         Power = 0.10;
117         PosSP = LO;
118     }
119
120 }
```

Claw.java

```
1  package frc.robot.Elevator;
2
3  import edu.wpi.first.wpilibj.Compressor;
4  import edu.wpi.first.wpilibj.DoubleSolenoid;
5  import edu.wpi.first.wpilibj.DoubleSolenoid.Value;
6  import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
7  import edu.wpi.first.wpilibj.PneumaticsModuleType;
8
9  public class Claw {
10
11     // This is only needed if we want the ability to turn off the compressor,
12     // change the pressure sensor, or query the compressor status.
13     public static Compressor Comp = new Compressor( 0, PneumaticsModuleType.CTREPCM );
14
15     // Order is forward channel and the reverse channel
16     static DoubleSolenoid Lft = new DoubleSolenoid( PneumaticsModuleType.CTREPCM, 0, 1 );
17     static DoubleSolenoid Rgt = new DoubleSolenoid( PneumaticsModuleType.CTREPCM, 3, 4 );
18     // DoubleSolenoid exampleDoublePH  = new DoubleSolenoid( 9, PneumaticsModuleType.REVPH, 4, 5 );
19
20     public static Value
21         State = Value.kForward;
22
23 //
24 //
25 //
26     public static void Initialize () {
27         Drop();
28     }
29
30     public static void Periodic () {
31         Lft.set( State );
32         Rgt.set( State );
33     }
34
35     public static void Display () {
36         String state = "Off";
37         if ( State == Value.kForward ) { state = "OPEN";  }
38         if ( State == Value.kReverse ) { state = "CLOSE"; }
39         SmartDashboard.putString( "CLAW", state );
40     }
41
42 //
43 //
44 //
45     public static void Drop  () { State = Value.kReverse; }
46     public static void Grab  () { State = Value.kForward; }
47     public static void Stop  () { State = Value.kOff;     }
48
49     public static void Reset () { Drop(); }
50
51     public static void Toggle () {
52         State = State == Value.kForward ? Value.kReverse : Value.kForward;
53     }
54 }
```

Lift.java

```java
package frc.robot.Elevator;

import com.ctre.phoenix.motorcontrol.VictorSPXControlMode;
import com.ctre.phoenix.motorcontrol.can.VictorSPX;

import edu.wpi.first.wpilibj.Ultrasonic;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import frc.robot.Hardware.Settings;

public class Lift {

    public static VictorSPX
        liftMotorL,
        liftMotorR;

    public static double     SP;
    public static double     power = 0;

    public static Ultrasonic Sonar;

    public static double
        HI = 27.0,
        MD = 10.0,
        LO = 4.5;

    public static double
        kP =  0.2;

    public static double
        direction,
        displacement,
        ratio,
        tolerance;

//
//
//
    public static void Initialize() {
        liftMotorL = new VictorSPX( Settings.LiftL_CANID );
        liftMotorR = new VictorSPX( Settings.LiftR_CANID );

        Sonar = new Ultrasonic(
            Settings.LiftSonar_DIO[0], // Input
            Settings.LiftSonar_DIO[1]  // Output
        );

        Ultrasonic.setAutomaticMode( true );

        SP = LO;
    }

    public static void Periodic() {

        // READ FROM SENSOR
        double PV = GetPosition();

        // CALCULATE DISPLACEMENT AND PSEUDO VELOCITY
        displacement = SP - PV;     // Displacement in inches
        ratio = displacement / 28;

        direction    = Math.signum( displacement ); // +1 for up; -1 for down
        // ratio        = displacement / ( HI - LO );  // Displacement ratio ( 0 to 1 )

        // SIMPLE PID CONTROLLER BASED RATIO
        power = direction * 0.80;
```

```
66
67          if ( Math.abs( displacement ) < 2 ) { power = 0; }
68
69          // SET MOTOR POWER
70          liftMotorL.set( VictorSPXControlMode.PercentOutput, power );
71          liftMotorR.set( VictorSPXControlMode.PercentOutput, power );
72      }
73
74  //
75  //
76  //
77      // public static void increase_power () { power += 0.001; }
78      // public static void decrease_power () { power -= 0.001; }
79
80      public static void Display () {
81          SmartDashboard.putNumber("Lift PV", GetPosition() );
82          SmartDashboard.putNumber("Lift SP", SP             );
83          SmartDashboard.putNumber("Lift Dir", direction     );
84          SmartDashboard.putNumber("Lift Pow", power         );
85      }
86
87  //
88  //
89  //
90      public static double GetPosition () {
91          return Sonar.getRangeInches();
92      }
93
94  //
95  // This intent of these methods is to have presets of where
96  // the lift mechanism is intended to stop.
97  //
98      public static void SetPosition ( double pos ) {
99          SP = pos;
100     }
101
102     public static void Reset () {
103         SetLO();
104     }
105
106     public static void SetHI () {
107         SP = HI;
108     }
109
110     public static void SetMD () {
111         SP = MD;
112     }
113
114     public static void SetLO () {
115         SP = LO;
116     }
117
118 }
```

Roller.java

```
1  package frc.robot.Elevator;
2
3  import com.ctre.phoenix.motorcontrol.ControlMode;
4  import com.ctre.phoenix.motorcontrol.can.VictorSPX;
5
6  import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
7  import frc.robot.Hardware.Settings;
8
9  public class Roller {
10
11     public static VictorSPX
12         Lroller,
13         Rroller;
14
15     public static double
16         Power;
17
18     public static void Initialize () {
19         Lroller = new VictorSPX( Settings.Roller_CANID[0] );
20         Rroller = new VictorSPX( Settings.Roller_CANID[1] );
21
22         Power = 0;
23
24         Lroller.setInverted( false );
25         Rroller.setInverted( true  );
26     }
27
28     public static void Periodic () {
29         Lroller.set( ControlMode.PercentOutput, Power );
30         Rroller.set( ControlMode.PercentOutput, Power );
31     }
32
33     public static void Display () {
34         SmartDashboard.putNumber( "Roller Power", Power );
35     }
36
37 //
38 //
39 //
40     public static void Reset () { Stop();         }
41     public static void Drool () { Power = +0.20; }
42     public static void Spit  () { Power = +1.00; }
43     public static void Suck  () { Power = -0.30; }
44     public static void Stop  () { Power = +0.00; }
45 }
```

EncTalonFX.java

```
1  package frc.robot.Hardware;
2
3  import com.ctre.phoenix.sensors.CANCoder;
4
5  public class EncTalonFX {
6
7      public CANCoder FalconEncoder;
8      public final static  int       kUnitsPerRevolution = 2048;
9
10     public EncTalonFX ( int CanBusID ) {
11         FalconEncoder = new CANCoder( CanBusID );
12     }
13
14 }
```

Module.java

```
1  package frc.robot.Hardware;
2
3  import com.ctre.phoenix.motorcontrol.ControlMode;
4  import com.ctre.phoenix.motorcontrol.NeutralMode;
5  import com.ctre.phoenix.motorcontrol.can.WPI_TalonFX;
6
7  import edu.wpi.first.math.controller.PIDController;
8  import edu.wpi.first.math.kinematics.SwerveModuleState;
9  import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
10
11 public class Module {
12
13     public EncTalonFX     SteerEncoder;
14     public int            ModuleNumber;
15     public String         ModuleName;
16     public WPI_TalonFX    DriveMotor;
17     public WPI_TalonFX    SteerMotor;
18     public PIDController SteerPID;
19     public double         TurnDiff     = 0;
20     public double         SpeedPlus    = 0;
21     public double         LastPosition = 0;
22     public boolean        still_turning_flag = false;
23
24     public double         DriveRatio = 0;
25     public double         SteerRatio = 0;
26     public double         minTurn    = 0;
27     public double         reverse    = 0;
28     public double         turnDir    = 0;
29     public double         turnMag    = 0;
30     public double         Power      = 0;
31
32     public double         SteerOffset = 0;
33
34     public Module ( String ModuleName, int ModuleNumber ) {
35
36         // REMEMBER VALUES
37         this.ModuleName   = ModuleName;
38         this.ModuleNumber = ModuleNumber;
39
40         // ID'S FOLLOW A PATTERN BASED ON MODULE NUMBERS
41         int DriveMotorID = ModuleNumber *2 -1;
42         int SteerMotorID = ModuleNumber *2 -0;
43
44         // DEFINE AND CONFIGURE DRIVE MOTOR
45         DriveMotor = new WPI_TalonFX ( DriveMotorID );
46         DriveMotor.setNeutralMode( NeutralMode.Brake );
47
48         // DEFINE AND CONFIGURE STEER MOTOR
49         SteerMotor = new WPI_TalonFX ( SteerMotorID );
50         SteerMotor.setNeutralMode( NeutralMode.Brake );
51
52         // DEFINE STEER ENCODER
53         SteerEncoder = new EncTalonFX ( ModuleNumber );
54     }
55
56     public void Display () {
57         // STEER ENCODER
58         SmartDashboard.putNumber(ModuleName + " PV", this.SteerEncoder.FalconEncoder.getAbsolutePosition() );
59     }
60
61     public void ResetDriveEncoder () {
62         // DriveMotor
63     }
64
65     public double GetDirection () {
```

```
66          return SteerEncoder.FalconEncoder.getAbsolutePosition();
67      }
68
69      public void Update ( SwerveModuleState state ) {
70
71          // CALCULATE DRIVE VALUES
72          DriveRatio = state.speedMetersPerSecond;
73          reverse    = 1;
74
75          // CALCULATE TURN VALUES
76          double SP = state.angle.getDegrees(); // Desired state (Final)
77          double PV = GetDirection();           // Current state (Initial)
78              PV = ( PV + 360 ) % 360;     // Ensure SP is between 0 and 360
79              SP = ( SP + 360 ) % 360;     // Ensure SP is between 0 and 360
80
81          // SMALLEST ANGLE TO SWIVEL: -180 to 180
82          minTurn = ( PV - SP + 180 ) % 360 - 180;
83              turnMag = Math.abs    ( minTurn );
84              turnDir = Math.signum( minTurn );
85
86              // MINIMIZE WHEEL SWIVEL: +120 becomes -60
87              if ( turnMag > +90 ) {
88                  turnMag  = 180 - turnMag; // Turn smaller angle
89                  turnDir  = -1;            // and reverse swivel
90                  reverse  = -1;            // and reverse drive
91              }
92
93          // DETERMINE POWER USING PSEUDO PID CONTROLLER
94          double SteerRatio = turnMag / 200;
95
96          // INCREASE STEERE OFFSET IF NOT MOVING
97          // double CurrentTurnSpeed = SteerMotor.getSelectedSensorVelocity();
98          // if ( CurrentTurnSpeed == 0 & turnMag > 1 ) {
99          //     SteerOffset += 0.0005;
100         // }
101
102         // ONE DEGREEE OFF IS GOOD ENOUGH
103         // if ( turnMag < 5 ) {
104             // SteerOffset = 0;
105             // SteerRatio  = 0.10;
106             // turnDir     = 1;
107         // }
108
109         // SET MOTOR CONTROLLERS
110         DriveMotor.set( ControlMode.PercentOutput, DriveRatio * reverse );
111         SteerMotor.set( ControlMode.PercentOutput, SteerRatio * turnDir );
112     }
113
114 }
```

Navigation.java

```
1  package frc.robot.Hardware;
2
3  import com.kauailabs.navx.frc.AHRS;
4  import edu.wpi.first.wpilibj.SPI;
5  import edu.wpi.first.wpilibj.Ultrasonic;
6  import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
7
8  public class Navigation {
9
10     public static Ultrasonic
11         FrontSonar;
12
13     public static double
14         Offset = 0;
15
16     public static AHRS NavX;
17
18     public static void Initialize () {
19         NavX = new AHRS( SPI.Port.kMXP );
20         NavX.calibrate();
21         Reset();
22
23         // FrontSonar = new Ultrasonic(
24         //      Settings.FrontSonar_DIO[0], // Input
25         //      Settings.FrontSonar_DIO[1]  // Output
26         // );
27
28     }
29
30     public static void Periodic () {}
31
32     public static void Display() {
33         SmartDashboard.putNumber( "Nav-Yaw",     GetDirection() );
34         SmartDashboard.putNumber( "Nav-Pitch",   GetPitch()     );
35     }
36
37 //
38 // SUPPORT METHODS
39 //
40     public static void   Reset     () { NavX.reset(); }
41
42     public static double GetPitch () { return  NavX.getPitch(); } // Forward tilt : - is up
43     public static double GetRoll  () { return  NavX.getRoll();  } // Side-to-side : + is ?
44     public static double GetYaw   () { return -NavX.getYaw();   } // Twist        : + is CCW
45
46     public static double GetDirection () {
47         return GetYaw();
48     }
49 }
```

Path.java

```
1  package frc.robot.Hardware;
2
3  import frc.robot.Elevator.Arm;
4  import frc.robot.Elevator.Lift;
5  import frc.robot.Elevator.Roller;
6
7  public class Path {
8
9      static double counter = 0;
10
11     static double SlowSpeed          = 0.10;
12     static double GoalDistance       = 24.0;
13     static double CommunityDistance = 18*12 + 8;
14
15     static double FastSpeed     = 0.25;
16     static double BalanceSpeed  = 0.15;
17
18     static double InclineAngle  = 10.0;
19     static double BalanceAngle  = 3.0;
20
21     static double ArmTolerance  = 1.0;
22     static double LiftTolerance = 1.0;
23
24     static double DroolDuration = 1.0;
25
26 /*  Do absolutely nothing. This is used as a safety in case
27     things go horribly wrong and Auton modes do not execute
28     as intended.
29 */
30
31     public static void Nothing () {
32         switch ( Stage.StageNumber ) {
33             default:
34                 Stage.Last();
35                 break;
36         }
37     }
38
39 /*
40     The "Balance" path is used when the elevator assembly
41     is no longer working and we become a defense only robot.
42 */
43
44     public static void Balance () {
45         switch ( Stage.StageNumber ) {
46             case 0:
47                 Autopilot.DriveSouth( FastSpeed );
48                 Stage.WaitForIncline( InclineAngle );
49                 break;
50
51             case 1:
52                 Autopilot.DriveSouth( BalanceSpeed );
53                 Stage.WaitForBalance( BalanceAngle );
54                 break;
55
56             default:
57                 Stage.Last();
58                 break;
59         }
60     }
61
62 /*  The "Any" paths are used to simply place a cone or cube
63     without balancing. Useful for when another team balances
64     and we need to stay out of the way.
65 */
```

```
66
67      public static void Any_1st () {
68          switch ( Stage.StageNumber ) {
69              case 0:
70                  Autopilot.DriveNorth( SlowSpeed );
71                  Stage.WaitForDistance( GoalDistance );
72                  break;
73
74              case 1:
75                  Autopilot.DriveSouth( FastSpeed );
76                  Stage.WaitForDistance( CommunityDistance );
77                  break;
78
79              default:
80                  Stage.Last();
81                  break;
82          }
83      }
84
85      public static void Any_2nd () {
86          switch ( Stage.StageNumber ) {
87              case 0:
88                  Arm.SetLO();
89                  Lift.SetHI();
90                  Stage.WaitForArmLift( ArmTolerance, LiftTolerance );
91                  break;
92
93              case 1:
94                  Roller.Drool();
95                  Stage.WaitForDuration( DroolDuration );
96                  break;
97
98              case 2:
99                  Arm.SetHI();
100                 Roller.Stop();
101                 Stage.WaitForArm( 1 );
102                 break;
103
104             case 3:
105                 Lift.SetLO();
106                 Stage.WaitForLift( 1 );
107                 break;
108
109             case 4:
110                 Autopilot.DriveSouth( FastSpeed );
111                 Stage.WaitForDistance( CommunityDistance );
112                 break;
113
114             default:
115                 Stage.Last();
116                 break;
117         }
118     }
119
120 /*
121     The "Center" paths are used when we start in the center of the
122     field and desire to balance.
123 */
124     public static void Ctr_1st () {
125         switch ( Stage.StageNumber ) {
126             case 0:
127                 Autopilot.DriveNorth( SlowSpeed );
128                 Stage.WaitForDistance( GoalDistance );
129                 break;
130
131             case 1:
132                 Autopilot.DriveSouth( FastSpeed );
133                 Stage.WaitForIncline( InclineAngle );
```

```
134                 break;
135
136             case 2:
137                 Autopilot.DriveSouth( BalanceSpeed );
138                 Stage.WaitForBalance( BalanceAngle );
139                 break;
140
141             default:
142                 Stage.Last();
143                 break;
144         }
145     }
146
147     public static void Ctr_2nd () {
148         switch ( Stage.StageNumber ) {
149             case 0:
150                 Elevator.Preset3();
151                 Stage.WaitForArmLift( ArmTolerance, LiftTolerance );
152                 break;
153
154             case 1:
155                 Roller.Drool();
156                 Stage.WaitForDuration( DroolDuration );
157                 break;
158
159             case 3:
160                 Elevator.Preset1();
161                 Stage.WaitForArmLift( ArmTolerance, LiftTolerance );
162                 break;
163
164             case 4:
165                 Autopilot.DriveSouth( FastSpeed );
166                 Stage.WaitForIncline( InclineAngle );
167                 break;
168
169             case 5:
170                 Autopilot.DriveSouth( BalanceSpeed );
171                 Stage.WaitForBalance( BalanceAngle );
172                 break;
173
174             default:
175                 Stage.Last();
176                 break;
177         }
178     }
179
180     public static void Lft_1st () {
181         switch ( Stage.StageNumber ) {
182             case 0:
183                 Autopilot.DriveNorth( SlowSpeed );
184                 Stage.WaitForDistance( GoalDistance );
185                 break;
186
187             case 1:
188                 // Twist robot to put block fully across line.
189
190             case 2:
191                 Elevator.Preset1();
192                 Stage.WaitForArmLift(ArmTolerance, LiftTolerance);
193                 break;
194
195             case 3:
196                 Autopilot.DriveSouth( FastSpeed );
197                 Stage.WaitForDistance( CommunityDistance ); // 18 ft, 4 in
198                 break;
199
200             default:
201                 Stage.Last();
```

```
202                 break;
203             }
204         }
205
206         public static void Lft_2nd () {
207             switch ( Stage.StageNumber ) {
208                 case 0:
209                     Elevator.Preset3();
210                     Stage.WaitForArmLift( ArmTolerance, LiftTolerance );
211                     break;
212
213                 case 1:
214                     Roller.Drool();
215                     Stage.WaitForDuration( DroolDuration );
216                     break;
217
218                 case 2:
219                     Elevator.Preset1();
220                     Stage.WaitForArmLift( ArmTolerance, LiftTolerance );
221                     break;
222
223                 case 3:
224                     Autopilot.DriveSouth( FastSpeed );
225                     Stage.WaitForDistance( CommunityDistance );
226                     break;
227
228                 default:
229                     Stage.Last();
230                     break;
231             }
232         }
233
234 /*
235         The "Right" paths are exactly the same as the left
236         paths. Manipulate the item and then move away a value
237         of CommunityDistance (last set to be 18 feet, 8 inches).
238 */
239
240         public static void Rgt_1st () {
241             Lft_1st();
242         }
243
244         public static void Rgt_2nd () {
245             Lft_2nd();
246         }
247
248 }
```

Settings.java

```
1  package frc.robot.Hardware;
2
3  public class Settings {
4
5      // AUTON STAGES
6      public static int
7          MAX_NUMBER_OF_STAGES = 20;
8
9      // CONTROLLER PORTS
10     public static int
11         DriveStickID = 0,
12         ManipStickID = 1;
13
14     // MAXIMUM SWERVE MODULE SPEEDS
15     public static double
16         MAX_DRIVE_RATIO = 1.00;
17
18     // MODULE ASSIGNMENTS
19     public static int
20         FL_moduleNumber = 1,
21         FR_moduleNumber = 4,
22         RL_moduleNumber = 5,
23         RR_moduleNumber = 3;
24
25     // MODULE LOCATIONS
26     public static double
27         FLx =  1, FLy =  1,
28         FRx =  1, FRy = -1,
29         RLx = -1, RLy =  1,
30         RRx = -1, RRy = -1;
31
32     // FRONT SONAR
33     public static int[]
34         FrontSonar_DIO = { 3, 2 };
35
36     // CLICKS PER INCH
37     public static double
38         IN_PER_CLICK = ( Math.PI * 4 ) / EncTalonFX.kUnitsPerRevolution;
39
40 //
41 // ELEVATOR ASSIGNMENTS
42 //
43     // ARM
44     public static int
45         Arm_CANID = 15;
46
47     // ROLLER (Left, Right)
48     public static int[]
49         Roller_CANID = { 30, 35 };
50
51     // LIFT
52     public static int
53         LiftR_CANID = 20,
54         LiftL_CANID = 21;
55
56     public static int[]
57         LiftSonar_DIO = { 0, 1 };
58
59 }
```

Stage.java

```
1  package frc.robot.Hardware;
2
3  import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
4  import frc.robot.Elevator.Arm;
5  import frc.robot.Elevator.Lift;
6
7  public class Stage {
8
9     public static double   AutonStartTime;
10    public static double   AutonFinalTime;
11    public static double   StageStartTime;
12
13    public static int      StageNumber;
14    public static boolean  ReadyToAdvance;
15    public static double[] StageDistance = new double[ Settings.MAX_NUMBER_OF_STAGES ];
16    public static double[] StageTime     = new double[ Settings.MAX_NUMBER_OF_STAGES ];
17
18    public static double
19       NegTilt = 0,
20       PosTilt = 0;
21
22    public static boolean
23       DoneWithAuton = true;
24
25    public static void Initialize () {
26
27       AutonStartTime = System.currentTimeMillis();
28       DoneWithAuton  = false;
29       StageStartTime = AutonStartTime;
30       StageNumber    = 0;
31
32       for ( int i = 0; i < 5; i++ ) {
33          StageDistance[i] = 0;
34          StageTime[i]     = 0;
35       }
36    }
37
38    public static void Display () {
39       SmartDashboard.putNumber("Robot-Stage Number",    StageNumber               );
40       SmartDashboard.putNumber("Robot-Auton Time",      GetAutonDuration() );
41
42       for ( int i = 0; i < 5; i++ ) {
43          SmartDashboard.putNumber("Stage Time " + i, StageTime[i] );
44          SmartDashboard.putNumber("Stage Dist " + i, StageDistance[i]      );
45       }
46    }
47
48 //
49 // The Next method advances to the next stage after storing Stage
50 // information. The Last method stops
51 //
52    public static void Begin () {
53       ReadyToAdvance = true;
54    }
55
56    public static void Next () {
57       StageDistance[ StageNumber ] = GetDistance();
58       StageTime     [ StageNumber ] = GetStageTime();
59
60       if ( (ReadyToAdvance == true) && (DoneWithAuton == false) ) {
61
62          ResetOdometer();
63
64          StageStartTime = System.currentTimeMillis();
65          StageNumber++;
```

```
66          }
67      }
68
69      public static void Last () {
70          AutonFinalTime = System.currentTimeMillis();
71          DoneWithAuton  = true;
72          // ReadyToAdvance = false;
73          // StageNumber = Settings.MAX_NUMBER_OF_STAGES - 1;
74
75          Autopilot.Stop();
76          Elevator.Reset();
77      }
78
79      public static void Fail () {
80          AutonFinalTime = System.currentTimeMillis();
81          ReadyToAdvance = false;
82          StageNumber    = Settings.MAX_NUMBER_OF_STAGES;
83      }
84
85  //
86  // Get...Time methods are useful in auton mode to determine the amount
87  // of time that the current stage or the entire auton process has been
88  // executing.
89  //
90      public static double GetAutonDuration () {
91          return ( System.currentTimeMillis() - AutonStartTime ) / 1000.0;
92      }
93
94      public static double GetStageTime () {
95          return ( System.currentTimeMillis() - StageStartTime ) / 1000.0;
96      }
97
98      public static void WaitForDuration ( double Duration ) {
99          if ( GetStageTime() < Duration ) {
100             ReadyToAdvance = false;
101         }
102     }
103
104 //
105 //
106 //
107     public static double GetDistance () {
108         double FL = Swerve.FL_module.DriveMotor.getSelectedSensorPosition();
109         // double FR = Swerve.FL_module.DriveMotor.getSelectedSensorPosition();
110         // double RL = Swerve.FL_module.DriveMotor.getSelectedSensorPosition();
111         // double RR = Swerve.FL_module.DriveMotor.getSelectedSensorPosition();
112
113         // ABS SINCE SOME WHEELS GOING BACKWARD
114         FL = Math.abs( FL );
115         // FR = Math.abs( FR );
116         // RL = Math.abs( RL );
117         // RR = Math.abs( RR );
118
119         // return Math.abs( FL );
120         // TAKE AN AVERAGE FOR SIMPLICITY
121         // return ( FL + FR + RL + RR ) * Settings.IN_PER_CLICK / 4;
122         return ( FL / 1000 * 3/4 );
123     }
124
125     public static void ResetOdometer () {
126         Swerve.FL_module.DriveMotor.setSelectedSensorPosition( 0 );
127         Swerve.FR_module.DriveMotor.setSelectedSensorPosition( 0 );
128         Swerve.RL_module.DriveMotor.setSelectedSensorPosition( 0 );
129         Swerve.RR_module.DriveMotor.setSelectedSensorPosition( 0 );
130     }
131
132     public static void WaitForDistance ( double Distance ) {
133         if ( GetDistance() <= Distance ) {
```

```
134             ReadyToAdvance = false;
135          }
136       }
137
138    //
139    //
140    //
141       public static void WaitForHeading ( double Heading, double Tolerance ) {
142          double diff = Autopilot.HeadingDiff( Heading );
143          if ( Math.abs( diff ) < Tolerance ) {
144             ReadyToAdvance = false;
145          }
146       }
147
148    //
149    // Second draft of code to be used in auton. Drive forward until we notice an incline.
150    // At that point we advance stages and continue to drive forward until we notice a
151    // balanced condition. It would be good to also have a maximum distance travelled for
152    // each stage and fail if the condition is not met.
153    //
154       public static void WaitForBalance ( double Tolerance ) {
155          double pitch = Navigation.GetPitch();
156          if ( Math.abs( pitch ) > Tolerance ) {
157             ReadyToAdvance = false;
158          }
159       }
160
161       public static void WaitForIncline ( double Angle ) {
162          double pitch = Navigation.GetPitch();
163          if ( Math.abs( pitch ) < Angle ) {
164             ReadyToAdvance = false;
165          }
166       }
167
168    //
169    //
170    //
171       public static void WaitForArm ( double Tolerance ) {
172          if ( Math.abs( Arm.PosER ) > 1.5 ) {
173             ReadyToAdvance = false;
174          }
175       }
176
177       public static void WaitForArmLift ( double ArmTolerance, double LiftTolerance ) {
178          WaitForArm ( ArmTolerance  );
179          WaitForLift( LiftTolerance );
180       }
181
182       public static void WaitForLift ( double Tolerance ) {
183          if ( Math.abs( Lift.displacement ) > 1.5 ) {
184             ReadyToAdvance = false;
185          }
186       }
187
188    // public static void WaitForWheelAlignment ( double Angle ) {
189
190       // }
191       // public static void WaitForHeading ( double Heading, double Tolerance ) {
192
193       // }
194
195    // public static boolean WaitForHeading( double targetHeading, double tolerance ) {
196    //    if ( Math.abs(Navigation.GetDelta(targetHeading)) < tolerance ) { return true; }
197    // }
198    //
199    //
200    // public static boolean WaitForTarget( double tolerance ) {
201    //    if ( Drivetrain.TargetMin<-tolerance || Drivetrain.TargetMax>tolerance ) {
```

```
202 //         StillWorking = true;
203 //         return true;
204 //     }
205 //     else {
206 //         return false;
207 //     }
208 //
209 // }
210
211
212 }
```

Swerve.java

```java
package frc.robot.Hardware;

import edu.wpi.first.math.geometry.Rotation2d;
import edu.wpi.first.math.geometry.Translation2d;
import edu.wpi.first.math.kinematics.ChassisSpeeds;
import edu.wpi.first.math.kinematics.SwerveDriveKinematics;
import edu.wpi.first.math.kinematics.SwerveModuleState;

public class Swerve {

    // CHASSIS SPEEDS
    public static ChassisSpeeds
        RobotSpeed;

    // MODULE DEFINITIONS
    public static Module
        FL_module,
        FR_module,
        RL_module,
        RR_module;

    // TRANSLATION OBJECTS
    static Translation2d
        FL_Trans2d,
        FR_Trans2d,
        RL_Trans2d,
        RR_Trans2d;

    // KINEMATICS OBJECT
    public static SwerveDriveKinematics
        Kinematics;

    // INITIALIZE
    public static void Initialize () {

        // CHASSIS SPEEDS
        RobotSpeed = new ChassisSpeeds( 0, 0, 0 );

        // MODULE DEFINITIONS
        FL_module = new Module( "FL", Settings.FL_moduleNumber );
        FR_module = new Module( "FR", Settings.FR_moduleNumber );
        RL_module = new Module( "RL", Settings.RL_moduleNumber );
        RR_module = new Module( "RR", Settings.RR_moduleNumber );

        // TRANSLATION OBJECT
        FL_Trans2d = new Translation2d( Settings.FLx, Settings.FLy );
        FR_Trans2d = new Translation2d( Settings.FRx, Settings.FRy );
        RL_Trans2d = new Translation2d( Settings.RLx, Settings.RLy );
        RR_Trans2d = new Translation2d( Settings.RRx, Settings.RRy );

        // KINEMATICS OBJECT
        Kinematics = new SwerveDriveKinematics( FL_Trans2d, FR_Trans2d, RL_Trans2d, RR_Trans2d );
    }

    public static void Display () {
        // SmartDashboard.putNumber("Robot-vx", RobotSpeed.vxMetersPerSecond      );
        // SmartDashboard.putNumber("Robot-vy", RobotSpeed.vyMetersPerSecond      );
        // SmartDashboard.putNumber("Robot-vt", RobotSpeed.omegaRadiansPerSecond );

        FL_module.Display();
        FR_module.Display();
        RL_module.Display();
        RR_module.Display();
    }

```

```
66      public static void UpdateFieldRelative ( double vx, double vy, double vt ) {
67          Rotation2d     Rot2d  = Rotation2d.fromDegrees( Navigation.GetYaw() );
68          ChassisSpeeds Speeds = ChassisSpeeds.fromFieldRelativeSpeeds( vx, vy, vt, Rot2d );
69          Update( Speeds );
70      }
71
72      public static void UpdateRobotRelative ( double vx, double vy, double vt ) {
73          ChassisSpeeds Speeds = new ChassisSpeeds( vx, vy, vt );
74          Update( Speeds );
75      }
76
77      private static void Update ( ChassisSpeeds Speeds ) {
78
79          // WAIT FOR WHEEL TO ADJUST TO HEADING
80          // boolean ok_to_drive = true;
81          // if ( FL_module.still_turning_flag ) { ok_to_drive = false; }
82          // if ( FR_module.still_turning_flag ) { ok_to_drive = false; }
83          // if ( RL_module.still_turning_flag ) { ok_to_drive = false; }
84          // if ( RR_module.still_turning_flag ) { ok_to_drive = false; }
85
86          // ALIGN WHEELS BEFORE TRANSLATION
87          // if ( ! ok_to_drive ) {
88          //     Speeds.vxMetersPerSecond = 0;
89          //     Speeds.vyMetersPerSecond = 0;
90          // }
91
92          // CALCULATE INDIVIDUAL MODULE STATES
93          SwerveModuleState[] ModuleStates = Kinematics.toSwerveModuleStates( Speeds );
94
95          // NORMALIZE WHEEL RATIOS IF ANY SPEED IS ABOVE SPECIFIED MAXIMUM
96          // SwerveDriveKinematics.desaturateWheelSpeeds( ModuleStates, Settings.MAX_DRIVE_RATIO );
97
98          // UPDATE ROBOT SPEEDS
99          // RobotSpeed = Kinematics.toChassisSpeeds( ModuleStates );
100
101         // UPDATE EACH MODULE
102         FL_module.Update( ModuleStates[0] );
103         FR_module.Update( ModuleStates[1] );
104         RL_module.Update( ModuleStates[2] );
105         RR_module.Update( ModuleStates[3] );
106     }
107
108 }
```

RobotRelative.java (Driver)

```java
package frc.robot.Driver;

import edu.wpi.first.wpilibj.XboxController;
import frc.robot.Elevator.Arm;
import frc.robot.Elevator.Claw;
import frc.robot.Elevator.Roller;
import frc.robot.Hardware.Elevator;
import frc.robot.Hardware.Swerve;
import frc.robot.Mode.Teleop;

public class RobotRelative {

    public static void Periodic () {

        // SHORT CUT FOR MANIP STICK
        XboxController Manip = Teleop.ManipStick;

        // GET VALUES
        double Xratio = Teleop.Xratio;
        double Yratio = Teleop.Yratio;
        double Tratio = Teleop.Tratio;

        // JOYSTICK COMPONENTS
        double Xmag = Math.abs( Xratio ); double Xsig = Math.signum( Xratio );
        double Ymag = Math.abs( Yratio ); double Ysig = Math.signum( Yratio );
        double Tmag = Math.abs( Tratio ); double Tsig = Math.signum( Tratio );

        // APPLY DEADZONE AND SCALE SPEEDS: e.g., 0.20 is a 20% dead zone
        if ( Xmag < 0.10 ) { Xmag = 0; } else { Xmag = Math.pow( Xmag-0.10, 2 ) / 2; }
        if ( Ymag < 0.10 ) { Ymag = 0; } else { Ymag = Math.pow( Ymag-0.10, 2 ) / 2; }
        if ( Tmag < 0.20 ) { Tmag = 0; } else { Tmag = Math.pow( Tmag-0.20, 2 ) / 2; }

        if ( Manip.getStartButton() ) {
            Arm.SetLO();
        }

        if ( Manip.getBackButton() ) {
            Arm.SetHI();
        }


        // TESTING COMMANDS
        if ( Manip.getAButtonPressed() ) {
            Elevator.Preset1();
        }

        if ( Manip.getBButtonPressed() ) {
            Elevator.Preset2();
        }

        if ( Manip.getYButtonPressed() ) {
            Elevator.Preset3();
        }

        Roller.Stop();

        if ( Manip.getRightBumper() ) {
            Roller.Suck();
        }

        if ( Manip.getLeftBumper() ) {
            Roller.Spit();
        }

        if ( Manip.getXButtonPressed() ) {
```

```
66                  Claw.Toggle();
67              }
68
69              if ( Manip.getLeftStickButton() ) {
70                  Arm.Arm.setSelectedSensorPosition( 90 );
71              }
72
73              // EXECUTE SWERVE
74              Swerve.UpdateRobotRelative( Xmag*Xsig, Ymag*Ysig, Tmag*Tsig );
75          }
76      }
```