

## Chapter 1

# Robot Code

## Robot.java

```
1 package frc.robot;
2
3 import edu.wpi.first.wpilibj.TimedRobot;
4 import frc.robot.Mode.Autonomous;
5 import frc.robot.Mode.Disabled;
6 import frc.robot.Mode.Simulation;
7 import frc.robot.Mode.Teleop;
8 import frc.robot.Mode.Test;
9 import frc.robot.Mode.Onabot;
10
11 public class Robot extends TimedRobot {
12     @Override public void robotInit      () { Onabot      .Initialize(); }
13     @Override public void robotPeriodic  () { Onabot      .Periodic();   }
14
15     @Override public void autonomousInit  () { Autonomous .Initialize(); }
16     @Override public void autonomousPeriodic () { Autonomous .Periodic();   }
17
18     @Override public void disabledInit    () { Disabled   .Initialize(); }
19     @Override public void disabledPeriodic () { Disabled   .Periodic();   }
20
21     @Override public void teleopInit      () { Teleop     .Initialize(); }
22     @Override public void teleopPeriodic  () { Teleop     .Periodic();   }
23
24     @Override public void testInit        () { Test       .Initialize(); }
25     @Override public void testPeriodic    () { Test       .Periodic();   }
26
27     @Override public void simulationInit   () { Simulation .Initialize(); }
28     @Override public void simulationPeriodic () { Simulation .Periodic();   }
29 }
```

## Onabot.java

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

## Autonomous.java

```
1 package frc.robot.Mode;
2
3 import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
4 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
5 import frc.robot.Hardware.Autopilot;
6 import frc.robot.Hardware.Elevator;
7 import frc.robot.Hardware.Stage;
8 import frc.robot.Hardware.Swerve;
9 import frc.robot.Hardware.Track;
10
11 public class Autonomous {
12
13     public static final String kDefault = "Nothing";
14     public static final String kPath01 = "Path-01";
15     public static final String kPath02 = "Path-02";
16     public static final String kPath03 = "Path-03";
17     public static final String kPath04 = "Path-04";
18     public static final SendableChooser<String> chooser = new SendableChooser<>();
19
20     public static void Initialize () {
21         chooser.setDefaultOption("Nothing", kDefault );
22         chooser.setDefaultOption("Path_01", kPath01 );
23         chooser.setDefaultOption("Path_02", kPath02 );
24         chooser.setDefaultOption("Path_03", kPath03 );
25         chooser.setDefaultOption("Path_04", kPath04 );
26         chooser.setDefaultOption("Path_05", kPath04 );
27         SmartDashboard.putData ( "PATH",      chooser );
28
29         Stage.Initialize();
30     }
31
32     public static void Periodic () {
33
34         Stage.Begin();
35
36         switch ( chooser.getSelected() ) {
37             case "Default" : Track.Track_00(); break;
38             case "Path-01" : Track.Track_01(); break;
39             case "Path-02" : Track.Track_02(); break;
40             case "Path-03" : Track.Track_03(); break;
41             case "Path-04" : Track.Track_04(); break;
42             case "Path-05" : Track.Track_05(); break;
43         }
44
45         Stage.Next();
46
47         if ( Stage.Number <= 100 ) { Stage.Display(); }
48
49
50         SmartDashboard.putString("CURRENT_PATH", chooser.getSelected() );
51
52         // EXECUTE COMMANDS
53         Elevator.Periodic();
54         Swerve.UpdateRobotRelative( Autopilot.vx, Autopilot.vy, Autopilot.vt );
55
56         // Example chassis speeds: 1 meter per second forward, 3 meters
57         // per second to the left, and rotation at 1.5 radians per second
58         // counterclockwise.                      F      L      CCW
59         // ChassisSpeeds speeds = new ChassisSpeeds(1.0, 3.0, 1.5);
60
61         // THESE NEED TO BE SET BY THE AUTONOMOUS MODE
62
63         // double curPitch = Navigation.GetPitch();
64         // SmartDashboard.putNumber("Robot-Pitch", curPitch);
65         // SmartDashboard.putNumber("Robot-Stage", stage );
```

---

```

66
67 // Navigation.Periodic();
68 // double curAng = Navigation.GetDirection();
69 // double target = 0;
70
71 // double diff = curAng - target;
72
73 // // SMALLEST ANGLE TO SWIVEL: -180 to 180
74 // double minTurn = ( diff + 180 ) % 360 - 180;
75 // double turnMag = Math.abs ( minTurn );
76 // double turnDir = Math.signum( minTurn );
77
78 // // MINIMIZE WHEEL SWIVEL: +120 becomes -60
79 // if ( turnMag > 0 ) {
80 // // turnMag = 180 - minTurn; // Turn smaller angle
81 // turnDir *= -1; // and reverse swivel
82 // }
83
84 // // DETERMINE POWER USING PSEUDO PID CONTROLLER
85 // if ( turnMag > 20 ) { vt = 0.15; }
86 // else if ( turnMag > 10 ) { vt = 0.10; }
87 // else if ( turnMag > 3 ) { vt = 0.06; }
88 // else if ( turnMag > 2 ) { vt = 0.00; }
89 // else if ( turnMag > 1 ) { vt = 0.00; }
90 // else { vt = 0.00; }
91
92 // vt *= turnDir;
93
94 // SET MOTOR CONTROLLERS
95 // SteerMotor.set( TalonFXControlMode.PercentOutput, PID.calculate( SP, PV ) );
96
97 // double diff = ( cur ) % 360 - 180;
98
99 }
100
101 }

```

## Teleop.java

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

## 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 straight 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 wheel 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 > 0 ) {
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 = 0.20; }
104     else if ( turnMag > 10 ) { vt = 0.10; }
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

```
1 package frc.robot.Hardware;
2
3 import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
4 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
5 import frc.robot.Driver.Aubrey;
6 import frc.robot.Driver.Default;
7 import frc.robot.Driver.Nate;
8 import frc.robot.Driver.Steensma;
9
10 public class Driver {
11
12     public static final String kDefault = "Default";
13     public static final String kAubrey = "Aubrey";
14     public static final String kNate = "Nate";
15     public static final String kSteensma = "Steensma";
16     public static final SendableChooser<String> chooser = new SendableChooser<>();
17
18     public static void Initialize () {
19         chooser.setDefaultOption("Default", kDefault );
20         chooser.addOption      ("Aubrey",   kAubrey  );
21         chooser.addOption      ("Nate",     kNate    );
22         chooser.addOption      ("Steensma", kSteensma );
23         SmartDashboard.putData ("DRIVER",  chooser  );
24     }
25
26     public static void Periodic () {
27         switch ( chooser.getSelected() ) {
28             case "Default" : Default .Periodic();
29             case "Aubrey"  : Aubrey  .Periodic();
30             case "Nate"    : Nate    .Periodic();
31             case "Steensma": Steensma .Periodic();
32         }
33     }
34
35     public static void Display () {
36         SmartDashboard.putString("Driver", chooser.getSelected() );
37     }
38
39 }
```

## Elevator.java

```
1 package frc.robot.Hardware;
2
3 public class Elevator {
4
5     public static void Initialize () {
6         ElevArm    .Initialize ();
7         ElevClaw   .Initialize ();
8         ElevLift   .Initialize ();
9         ElevWrist  .Initialize ();
10    }
11
12    public static void Periodic () {
13        ElevArm    .Periodic ();
14        ElevClaw   .Periodic ();
15        ElevLift   .Periodic ();
16        ElevWrist  .Periodic ();
17    }
18
19    public static void Display () {
20        ElevArm    .Display ();
21        ElevClaw   .Display ();
22        ElevLift   .Display ();
23        ElevWrist  .Display ();
24    }
25
26    public static void Reset () {
27        ElevArm    .Reset ();
28        ElevClaw   .Reset ();
29        ElevLift   .Reset ();
30        ElevWrist  .Reset ();
31    }
32
33    public static void Set ( double A, double C, double L, double W ) {
34        ElevArm    .SetPosition( A );
35        ElevClaw   .SetPosition( C );
36        ElevLift   .SetPosition( L );
37        ElevWrist  .SetPosition( W );
38    }
39
40 }
```

## ElevArm.java

```
1 package frc.robot.Hardware;
2
3 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
4
5 public class ElevArm {
6
7     public static double target_position = 0;
8
9     //
10    //
11    //
12    public static void Initialize () {
13
14    }
15
16    public static void Periodic () {
17        // Some sort of controller to find the position
18        // difference and set the motor ratio. Might need
19        // a PID controller to hold position.
20    }
21
22    public static void Display () {
23        SmartDashboard.putNumber("Elevator-Arm_Pos", GetPosition() );
24        SmartDashboard.putNumber("Elevator-Arm_Tar", target_position );
25    }
26
27    //
28    //
29    //
30    public static double GetPosition () {
31        return 0;
32    }
33
34    public static void SetPosition ( double pos ) {
35        target_position = pos;
36    }
37
38    //
39    //
40    //
41    public static void Reset () {
42        Retract();
43    }
44
45    public static void Extend () {
46
47    }
48
49    public static void Retract () {
50
51    }
52
53 }
```

## ElevClaw.java

```

1 package frc.robot.Hardware;
2
3 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
4
5 public class ElevClaw {
6
7     public static double target_position;
8
9     //
10    //
11    //
12    public static void Initialize () {
13
14    }
15
16    public static void Periodic () {
17        // Some sort of controller to find the position
18        // difference and set the motor ratio. Might need
19        // a PID controller to hold position.
20    }
21
22    public static void Display () {
23        SmartDashboard.putNumber("Elevator-Claw_Pos", GetPosition() );
24        SmartDashboard.putNumber("Elevator-Claw_Tar", target_position );
25    }
26
27    //
28    //
29    //
30    public static void Reset () {
31        Open();
32    }
33
34    public static void Grab () {
35
36    }
37
38    public static void Open () {
39
40    }
41
42    //
43    //
44    //
45    public static double GetPosition () {
46        return 0;
47    }
48
49    public static void SetPosition ( double pos ) {
50        target_position = pos;
51    }
52
53 }

```

## ElevLift.java

```
1 package frc.robot.Hardware;
2
3 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
4
5 public class ElevWrist {
6
7     public static double target_position;
8
9     //
10    //
11    //
12    public static void Initialize () {
13
14    }
15
16    public static void Periodic () {
17        // Some sort of controller to find the position
18        // difference and set the motor ratio. Might need
19        // a PID controller to hold position.
20    }
21
22    public static void Display () {
23        SmartDashboard.putNumber("Elevator-Wrist_Pos", GetPosition() );
24        SmartDashboard.putNumber("Elevator-Wrist_Tar", target_position );
25    }
26
27    //
28    //
29    //
30    public static double GetPosition () {
31        return 0;
32    }
33
34    public static void SetPosition ( double pos ) {
35        target_position = pos;
36    }
37
38    //
39    //
40    //
41    public static void Reset () {
42        Bend();
43    }
44
45    public static void Bend () {
46
47    }
48
49    public static void Straighten () {
50
51    }
52
53 }
```

## ElevWrist.java

```
1 package frc.robot.Hardware;
2
3 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
4
5 public class ElevWrist {
6
7     public static double target_position;
8
9     //
10    //
11    //
12    public static void Initialize () {
13
14    }
15
16    public static void Periodic () {
17        // Some sort of controller to find the position
18        // difference and set the motor ratio. Might need
19        // a PID controller to hold position.
20    }
21
22    public static void Display () {
23        SmartDashboard.putNumber("Elevator-Wrist_Pos", GetPosition() );
24        SmartDashboard.putNumber("Elevator-Wrist_Tar", target_position );
25    }
26
27    //
28    //
29    //
30    public static double GetPosition () {
31        return 0;
32    }
33
34    public static void SetPosition ( double pos ) {
35        target_position = pos;
36    }
37
38    //
39    //
40    //
41    public static void Reset () {
42        Bend();
43    }
44
45    public static void Bend () {
46
47    }
48
49    public static void Straighten () {
50
51    }
52
53 }
```

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

```

---

```

66     double PV = GetDirection();           // Current state (Initial)
67     SP = ( SP + 360 ) % 360;              // Ensure SP is between 0 and 360
68
69     // SMALLEST ANGLE TO SWIVEL: -180 to 180
70     double minTurn = ( PV - SP + 180 ) % 360 - 180;
71     double turnMag = Math.abs ( minTurn );
72     double turnDir = Math.signum( minTurn );
73
74     // MINIMIZE WHEEL SWIVEL: +120 becomes -60
75     if ( turnMag > 90 ) {
76         turnMag = 180 - turnMag; // Turn smaller angle
77         turnDir *= -1;           // and reverse swivel
78         reverse *= -1;           // and reverse drive
79     }
80
81     // DETERMINE POWER USING PSEUDO PID CONTROLLER
82     double SteerRatio = 0;
83     if ( turnMag > 20 ) { SteerRatio = 0.20; }
84     else if ( turnMag > 10 ) { SteerRatio = 0.08; }
85     else if ( turnMag > 1 ) { SteerRatio = 0.07; }
86     else { SteerRatio = 0.00; }
87
88     // If any the heading difference of any wheel is more than one degree and the
89     // module has not turned in the last 20 ms then increase the turning speed.
90     double CurrentPosition = SteerEncoder.FalconEncoder.getAbsolutePosition();
91     boolean is_moving = CurrentPosition == LastPosition ? false : true;
92     if ( turnMag < 1 ) { SpeedPlus = 0.000; }
93     else if ( ! is_moving ) { SpeedPlus += 0.001; }
94     else { }
95
96     // SET TURNING FLAG, CHECKED BY DRIVETRAIN
97     still_turning_flag = turnMag >= 5 ? true : false;
98
99     // RESET LAST POSITION TO SEE IF MOVEMENT OCCURED
100    LastPosition = CurrentPosition;
101    SteerRatio += SpeedPlus;
102
103    // SET MOTOR CONTROLLERS
104    DriveMotor.setVoltage( DriveRatio * 10 * reverse );
105    SteerMotor.setVoltage( SteerRatio * 10 * turnDir );
106 }
107
108 }
```

## 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.smartdashboard.SmartDashboard;
6
7 public class Navigation {
8
9     public static AHRS NavX;
10
11     public static void Initialize () {
12         NavX = new AHRS( SPI.Port.kMXP );
13         NavX.calibrate();
14         Reset();
15     }
16
17     public static void Periodic () {}
18
19     public static void Display() {
20         SmartDashboard.putNumber( "Nav-Yaw",    GetYaw() );
21         SmartDashboard.putNumber( "Nav-Pitch",   GetPitch() );
22         SmartDashboard.putNumber( "Nav-Roll",    GetRoll() );
23     }
24
25     //
26     // SUPPORT METHODS
27     //
28     public static void Reset () { NavX.reset(); }
29
30     public static double GetPitch () { return NavX.getPitch(); } // Forward tilt : - is up
31     public static double GetRoll () { return NavX.getRoll(); } // Side-to-side : + is ?
32     public static double GetYaw () { return -NavX.getYaw(); } // Twist : + is CCW
33
34     public static double GetDirection () { return GetYaw(); }
35 }

```

## Settings.java

```
1 package frc.robot.Hardware;
2
3 public class Settings {
4
5     // CONTROLLER PORTS
6     public static int
7         DriveStickID = 0,
8         ManipStickID = 1;
9
10    // MAXIMUM MODULE SPEEDS
11    public static double
12        MAX_DRIVE_RATIO = 0.20;
13
14    // MODULE ASSIGNMENTS
15    public static int
16        FL_moduleNumber = 1,
17        FR_moduleNumber = 4,
18        RL_moduleNumber = 5,
19        RR_moduleNumber = 3;
20
21    // MODULE LOCATIONS
22    public static double
23        FLx = 1, FLy = 1,
24        FRx = 1, FRy = -1,
25        RLx = -1, RLy = 1,
26        RRx = -1, RRy = -1;
27
28    // CLICKS PER FOOT
29    public static double
30        IN_PER_CLICK = ( Math.PI * 4 ) / EncTalonFX.kUnitsPerRevolution;
31
32 }
```

## Stage.java

```

1 package frc.robot.Hardware;
2
3 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
4
5 public class Stage {
6
7     public static double   AutonStartTime;
8     public static double   AutonFinalTime;
9     public static double   StageStartTime;
10
11     public static int       Number;
12     public static boolean   ReadyToAdvance;
13     public static double[]  StageDistance = new double[10];
14     public static double[]  StageTime     = new double[10];
15
16     public static double    NegTilt = 0;
17     public static double    PosTilt = 0;
18
19
20     public static void Initialize () {
21         AutonStartTime = System.currentTimeMillis();
22         StageStartTime = AutonStartTime;
23         Number         = 0;
24     }
25
26     public static void Display () {
27         SmartDashboard.putNumber("Robot-Stage-Number",    Number           );
28         SmartDashboard.putNumber("Robot-Stage-Distance",  GetDistance()       );
29         SmartDashboard.putNumber("Robot-Stage-Time",      GetStageTime()      );
30         SmartDashboard.putNumber("Robot-Auton-Time",      GetAutonDuration() );
31     }
32
33     //
34     // The Next method advances to the next stage after storing Stage
35     // information. The Last method stops
36     //
37     public static void Begin () {
38         Autopilot.Stop();
39         ReadyToAdvance = true;
40     }
41
42     public static void Next () {
43         if ( ReadyToAdvance == true ) {
44             StageDistance[Number] = GetDistance();
45             StageTime    [Number] = GetStageTime();
46             ResetOdometer();
47             StageStartTime = System.currentTimeMillis();
48             Number++;
49         }
50     }
51
52     public static void Last () {
53         AutonFinalTime = System.currentTimeMillis();
54         ReadyToAdvance = false;
55
56         Autopilot.Stop();
57     }
58
59     public static void Fail () {
60         AutonFinalTime = System.currentTimeMillis();
61         ReadyToAdvance = false;
62         Number         = 100;
63     }
64
65     //

```

```
66 // Get...Time methods are is useful in auton mode to determine the amount
67 // of time that the current stage or the entire auton process has been
68 // executing.
69 //
70 public static double GetAutonDuration () {
71     return ( System.currentTimeMillis() - AutonStartTime ) / 1000.0;
72 }
73
74 public static double GetStageTime () {
75     return ( System.currentTimeMillis() - StageStartTime ) / 1000.0;
76 }
77
78 public static void WaitForDuration ( double Duration ) {
79     if ( GetStageTime() < Duration ) {
80         ReadyToAdvance = false;
81     }
82 }
83
84 //
85 //
86 //
87 public static double GetDistance () {
88     double FL = Swerve.FL_module.DriveMotor.getSelectedSensorPosition();
89     double FR = Swerve.FR_module.DriveMotor.getSelectedSensorPosition();
90     double RL = Swerve.RL_module.DriveMotor.getSelectedSensorPosition();
91     double RR = Swerve.RR_module.DriveMotor.getSelectedSensorPosition();
92
93     // ABS SINCE SOME WHEELS GOING BACKWARD
94     FL = Math.abs( FL );
95     FR = Math.abs( FR );
96     RL = Math.abs( RL );
97     RR = Math.abs( RR );
98
99     // TAKE AN AVERAGE FOR SIMPLICITY
100     return ( FL + FR + RL + RR ) * Settings.IN_PER_CLICK / 4;
101 }
102
103 public static void ResetOdometer () {
104     Swerve.FL_module.DriveMotor.setSelectedSensorPosition( 0 );
105     Swerve.FR_module.DriveMotor.setSelectedSensorPosition( 0 );
106     Swerve.RL_module.DriveMotor.setSelectedSensorPosition( 0 );
107     Swerve.RR_module.DriveMotor.setSelectedSensorPosition( 0 );
108 }
109
110 public static void WaitForDistance ( double Distance ) {
111     if ( GetDistance() < Distance ) {
112         ReadyToAdvance = false;
113     }
114 }
115
116 //
117 //
118 //
119 public static void WaitForHeading ( double Heading, double Tolerance ) {
120     double diff = Autopilot.HeadingDiff( Heading );
121     if ( Math.abs( diff ) < Tolerance ) {
122         ReadyToAdvance = false;
123     }
124 }
125
126 //
127 // Second draft of code to be used in auton. Drive forward until we notice an incline.
128 // At that point we advance stages and continue to drive forward until we notice a
129 // balanced condition. It would be good to also have a maximum distance travelled for
130 // each stage and fail if the condition is not met.
131 //
132 public static void WaitForBalance ( double Tolerance ) {
133     double pitch = Navigation.GetPitch();
```

```
134         if ( Math.abs( pitch ) > Tolerance ) {
135             ReadyToAdvance = false;
136         }
137     }
138
139     public static void WaitForIncline ( double Angle ) {
140         double pitch = Navigation.GetPitch();
141         if ( Math.abs( pitch ) < Angle ) {
142             ReadyToAdvance = false;
143         }
144     }
145
146     //
147     //
148     //
149     // public static void WaitForWheelAlignment ( double Angle ) {
150
151     // }
152     // public static void WaitForHeading ( double Heading, double Tolerance ) {
153
154     // }
155
156     // public static boolean WaitForHeading( double targetHeading, double tolerance ) {
157     //     if ( Math.abs(Navigation.GetDelta(targetHeading)) < tolerance ) { return true; }
158     // }
159     //
160     //
161     // public static boolean WaitForTarget( double tolerance ) {
162     //     if ( Drivetrain.TargetMin<-tolerance || Drivetrain.TargetMax>tolerance ) {
163     //         StillWorking = true;
164     //         return true;
165     //     }
166     //     else {
167     //         return false;
168     //     }
169     // }
170     // }
171
172
173 }
```

## Swerve.java

```
1 package frc.robot.Hardware;
2
3 import edu.wpi.first.math.geometry.Rotation2d;
4 import edu.wpi.first.math.geometry.Translation2d;
5 import edu.wpi.first.math.kinematics.ChassisSpeeds;
6 import edu.wpi.first.math.kinematics.SwerveDriveKinematics;
7 import edu.wpi.first.math.kinematics.SwerveModuleState;
8 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
9
10 public class Swerve {
11
12     // CHASSIS SPEEDS
13     public static ChassisSpeeds
14         RobotSpeed;
15
16     // MODULE DEFINITIONS
17     public static Module
18         FL_module,
19         FR_module,
20         RL_module,
21         RR_module;
22
23     // TRANSLATION OBJECTS
24     static Translation2d
25         FL_Trans2d,
26         FR_Trans2d,
27         RL_Trans2d,
28         RR_Trans2d;
29
30     // KINEMATICS OBJECT
31     public static SwerveDriveKinematics
32         Kinematics;
33
34     // INITIALIZE
35     public static void Initialize () {
36
37         // CHASSIS SPEEDS
38         RobotSpeed = new ChassisSpeeds( 0, 0, 0 );
39
40         // MODULE DEFINITIONS
41         FL_module = new Module( "FL", Settings.FL_moduleNumber );
42         FR_module = new Module( "FR", Settings.FR_moduleNumber );
43         RL_module = new Module( "RL", Settings.RL_moduleNumber );
44         RR_module = new Module( "RR", Settings.RR_moduleNumber );
45
46         // TRANSLATION OBJECT
47         FL_Trans2d = new Translation2d( Settings.FLx, Settings.FLy );
48         FR_Trans2d = new Translation2d( Settings.FRx, Settings.FRy );
49         RL_Trans2d = new Translation2d( Settings.RLx, Settings.RLy );
50         RR_Trans2d = new Translation2d( Settings.RRx, Settings.RRy );
51
52         // KINEMATICS OBJECT
53         Kinematics = new SwerveDriveKinematics( FL_Trans2d, FR_Trans2d, RL_Trans2d, RR_Trans2d );
54     }
55
56     public static void Display () {
57         SmartDashboard.putNumber( "Robot-vx", RobotSpeed.vxMetersPerSecond );
58         SmartDashboard.putNumber( "Robot-vy", RobotSpeed.vyMetersPerSecond );
59         SmartDashboard.putNumber( "Robot-vt", RobotSpeed.omegaRadiansPerSecond );
60
61         FL_module.Display();
62         FR_module.Display();
63         RL_module.Display();
64         RR_module.Display();
65     }
```



```

66
67     public static void UpdateFieldRelative ( double vx, double vy, double vt ) {
68         Rotation2d Rot2d = Rotation2d.fromDegrees( Navigation.GetYaw() );
69         ChassisSpeeds Speeds = ChassisSpeeds.fromFieldRelativeSpeeds( vx, vy, vt, Rot2d );
70         Update( Speeds );
71     }
72
73     public static void UpdateRobotRelative ( double vx, double vy, double vt ) {
74         ChassisSpeeds Speeds = new ChassisSpeeds( vx, vy, vt );
75         Update( Speeds );
76     }
77
78     private static void Update ( ChassisSpeeds Speeds ) {
79
80         // WAIT FOR WHEEL TO ADJUST TO HEADING
81         boolean ok_to_drive = true;
82         if ( FL_module.still_turning_flag ) { ok_to_drive = false; }
83         if ( FR_module.still_turning_flag ) { ok_to_drive = false; }
84         if ( RL_module.still_turning_flag ) { ok_to_drive = false; }
85         if ( RR_module.still_turning_flag ) { ok_to_drive = false; }
86
87         // ALIGN WHEELS BEFORE TRANSLATION
88         if ( ! ok_to_drive ) {
89             Speeds.vxMetersPerSecond = 0;
90             Speeds.vyMetersPerSecond = 0;
91         }
92
93         // CALCULATE INDIVIDUAL MODULE STATES
94         SwerveModuleState[] ModuleStates = Kinematics.toSwerveModuleStates( Speeds );
95
96         // NORMALIZE WHEEL RATIOS IF ANY SPEED IS ABOVE SPECIFIED MAXIMUM
97         SwerveDriveKinematics.desaturateWheelSpeeds( ModuleStates, Settings.MAX_DRIVE_RATIO );
98
99         // UPDATE ROBOT SPEEDS
100        RobotSpeed = Kinematics.toChassisSpeeds( ModuleStates );
101
102        // UPDATE EACH MODULE
103        FL_module.Update( ModuleStates[0] );
104        FR_module.Update( ModuleStates[1] );
105        RL_module.Update( ModuleStates[2] );
106        RR_module.Update( ModuleStates[3] );
107    }
108
109 }

```

## Track.java

```
1 package frc.robot.Hardware;
2
3 public class Track {
4
5     public static void Track_00 () {
6         switch ( Stage.Number ) {
7             case 0:
8                 Autopilot.Stop();
9                 Stage.WaitForDuration( 2.00 );
10                break;
11
12            default:
13                Stage.Last();
14                break;
15        }
16    }
17
18    public static void Track_01 () {
19        switch ( Stage.Number ) {
20            case 0:
21                Stage.WaitForDuration( 1.00 );
22                break;
23
24            default:
25                Stage.Last();
26                break;
27        }
28    }
29
30    public static void Track_02 () {
31        switch ( Stage.Number ) {
32            case 0:
33                Stage.WaitForDuration( 2.00 );
34                break;
35
36            default:
37                Stage.Last();
38                break;
39        }
40    }
41
42    public static void Track_03 () {
43        switch ( Stage.Number ) {
44            case 0:
45                Autopilot.DriveNorth( 0.20 );
46                Stage.WaitForDistance( 20 * 100 );
47                break;
48
49            case 1:
50                Autopilot.TurnToHeading( 0 );
51                Stage.WaitForDuration( 3.00 );
52                break;
53
54            // case 2:
55            //     Autopilot.DriveSouth( 0.20 );
56            //     Stage.WaitForDistance( 136 );
57            //     break;
58
59            default:
60                Stage.Last();
61                break;
62        }
63    }
64
65    public static void Track_04 () {
```

```
66         switch ( Stage.Number ) {
67             case 0:
68                 Autopilot.TurnToHeading( 90 );
69                 Stage.WaitForDuration( 2.00 );
70                 break;
71
72             case 1:
73                 Autopilot.TurnToHeading( 180 );
74                 Stage.WaitForDuration( 2.00 );
75                 break;
76
77             default:
78                 Stage.Last();
79                 break;
80         }
81     }
82
83     public static void Track_05 () {
84         switch ( Stage.Number ) {
85             case 0:
86                 Stage.WaitForDuration( 1.00 );
87                 break;
88
89             default:
90                 Stage.Last();
91                 break;
92         }
93     }
94
95     public static void Track_06 () {
96         switch ( Stage.Number ) {
97             case 0:
98                 Stage.WaitForDuration( 1.00 );
99                 break;
100
101             default:
102                 Stage.Last();
103                 break;
104         }
105     }
106
107 }
```

## Default.java (Driver)

```
1 package frc.robot.Driver;
2
3 import frc.robot.Hardware.ElevLift;
4 import frc.robot.Hardware.Swerve;
5 import frc.robot.Mode.Teleop;
6
7 public class Default {
8
9     public static void Periodic () {
10
11         // GET VALUES
12         double Xratio = Teleop.Xratio;
13         double Yratio = Teleop.Yratio;
14         double Tratio = Teleop.Tratio;
15
16         // JOYSTICK COMPONENTS
17         double Xmag = Math.abs( Xratio ); double Xsig = Math.signum( Xratio );
18         double Ymag = Math.abs( Yratio ); double Ysig = Math.signum( Yratio );
19         double Tmag = Math.abs( Tratio ); double Tsig = Math.signum( Tratio );
20
21         // APPLY DEADZONE AND SCALE SPEEDS: e.g., 0.20 is a 20% dead zone
22         if ( Xmag < 0.10 ) { Xmag = 0; } else { Xmag = Math.pow( Xmag-0.10, 2 ) / 2; }
23         if ( Ymag < 0.10 ) { Ymag = 0; } else { Ymag = Math.pow( Ymag-0.10, 2 ) / 2; }
24         if ( Tmag < 0.20 ) { Tmag = 0; } else { Tmag = Math.pow( Tmag-0.20, 2 ) / 2; }
25
26         // TESTING COMMANDS
27         if ( Teleop.DriveStick.getRawButton( 7 ) ) { ElevLift.SetHigh (); }
28         else { ElevLift.SetLow (); }
29
30         // SEND SPEEDS TO SWERVE CLASS
31         Swerve.UpdateRobotRelative( Xmag*Xsig, Ymag*Ysig, Tmag*Tsig );
32     }
33 }
```