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.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         // Sonar       .Initialize();
17         // Vision      .Initialize();
18     }
19
20     public static void Periodic () {
21         Driver      .Display();
22         Elevator    .Display();
23         Navigation  .Display();
24         Swerve      .Display();
25
26         // Sonar       .Display();
27         // Vision      .Display();
28     }
29
30 }
```

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 SendableChooser<String> chooser = new SendableChooser<>();
18
19     public static void Initialize () {
20         chooser.setDefaultOption("Nothing", kDefault );
21         chooser.setDefaultOption("Path 01", kPath01  );
22         chooser.setDefaultOption("Path 02", kPath02  );
23         chooser.setDefaultOption("Path 03", kPath03  );
24         SmartDashboard.putData  ("PATH",    chooser  );
25
26         Stage.Initialize();
27     }
28
29     public static void Periodic () {
30         Stage.Begin();
31
32         switch ( chooser.getSelected() ) {
33             case "Default" : Track.Track_00(); break;
34             case "Path-01" : Track.Track_01(); break;
35             case "Path-02" : Track.Track_02(); break;
36             case "Path-03" : Track.Track_03(); break;
37         }
38
39         Stage.Display();
40
41         Elevator.Periodic();
42         Swerve.UpdateFieldRelative( Autopilot.vx, Autopilot.vy, Autopilot.vt );
43
44         // Example chassis speeds: 1 meter per second forward, 3 meters
45         // per second to the left, and rotation at 1.5 radians per second
46         // counterclockwise.                          F    L   CCW
47         // ChassisSpeeds speeds = new ChassisSpeeds(1.0, 3.0, 1.5);
48
49         // THESE NEED TO BE SET BY THE AUTONOMOUS MODE
50
51         // double curPitch  = Navigation.GetPitch();
52         // SmartDashboard.putNumber("Robot-Pitch", curPitch);
53         // SmartDashboard.putNumber("Robot-Stage", stage );
54
55         // double
56         //     vx = 0.00,
57         //     vy = 0.00,
58         //     vt = 0.00;
59
60         // if ( stage == 0 ) {
61         //     vx = 0.08;
62         //     if ( curPitch > 0 ) { stage++; };
63         // }
64
65         // if ( stage == 1 ) {
```

```
66          //      vx = 0;
67          // }
68
69          // Navigation.Periodic();
70          // double curAng = Navigation.GetDirection();
71          // double target = 0;
72
73          // double diff = curAng - target;
74
75          // SmartDashboard.putNumber("Robot-DIFF", diff);
76
77          // // SMALLEST ANGLE TO SWIVEL: -180 to 180
78          // double minTurn = ( diff + 180 ) % 360 - 180;
79          //      double turnMag = Math.abs    ( minTurn );
80          //      double turnDir = Math.signum( minTurn );
81
82          // // MINIMIZE WHEEL SWIVEL: +120 becomes -60
83          // if ( turnMag > 0 ) {
84          //      // turnMag  = 180 - minTurn; // Turn smaller angle
85          //      turnDir *= -1;            // and reverse swivel
86          // }
87
88          // // DETERMINE POWER USING PSEUDO PID CONTROLLER
89          // if       ( turnMag > 20 ) { vt = 0.15; }
90          // else if  ( turnMag > 10 ) { vt = 0.10; }
91          // else if  ( turnMag >  3 ) { vt = 0.06; }
92          // else if  ( turnMag >  2 ) { vt = 0.00; }
93          // else if  ( turnMag >  1 ) { vt = 0.00; }
94          // else                      { vt = 0.00; }
95
96          // vt *= turnDir;
97
98          // SET MOTOR CONTROLLERS
99          // SteerMotor.set( TalonFXControlMode.PercentOutput, PID.calculate( SP, PV ) );
100
101         // double diff = ( cur  ) % 360 - 180;
102
103     }
104
105 }
```

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     public static void DriveStraight ( double Vx, double Vy ) {
36         vx = Vx; vy = Vy; vt = 0;
37     }
38
39 //
40 //
41 //
42     public static void DriveNorth ( double Speed ) {
43         vx = +Speed; vy = 0; vt = 0;
44     }
45
46     public static void DriveSouth ( double Speed ) {
47         vx = -Speed; vy = 0; vt = 0;
48     }
49
50     public static void DriveWest ( double Speed ) {
51         vx = 0; vy = +Speed; vt = 0;
52     }
53
54     public static void DriveEast ( double Speed ) {
55         vx = Speed; vy = -Speed; vt = 0;
56     }
57
58 //
59 //
60 //
61     public static void DriveNorthWest ( double Speed ) {
62         double radians = Math.toRadians( 45 );
63         double speed   = Speed * Math.cos( radians );
```

```
64          vx = +speed; vy = +speed; vt = 0;
65      }
66
67      public static void DriveNorthEast ( double Speed ) {
68          double radians = Math.toRadians( 45 );
69          double speed   = Speed * Math.cos( radians );
70          vx = +speed; vy = -speed; vt = 0;
71      }
72
73      public static void DriveSouthWest ( double Speed ) {
74          double radians = Math.toRadians( 45 );
75          double speed   = Speed * Math.cos( radians );
76          vx = -speed; vy = +speed; vt = 0;
77      }
78
79      public static void DriveSouthEast ( double Speed ) {
80          double radians = Math.toRadians( 45 );
81          double speed   = Speed * Math.cos( radians );
82          vx = -speed; vy = -speed; vt = 0;
83      }
84
85  //
86  //
87  //
88      public static void SetWheelsToHeading ( double Angle) {
89          Swerve.SetWheelsToHeading( Angle );
90      }
91
92  //
93  // TurnToHeading sets the turn power variable in Autonomous mode to reach
94  // the desired heading using the shortest wheek swivel.
95  //
96      public static void TurnToHeading ( double NewHeading ) {
97          double minTurn = HeadingDiff( NewHeading );
98          double turnMag = Math.abs    ( minTurn );
99          double turnDir = Math.signum( minTurn );
100
101             // MINIMIZE WHEEL SWIVEL: +120 becomes -60
102             if ( turnMag > 0 ) {
103                 turnMag  = 180 - minTurn; // Turn smaller angle
104                 turnDir *= -1;            // and reverse swivel
105             }
106
107             // DETERMINE POWER USING PSEUDO PID CONTROLLER
108             if      ( turnMag > 20 ) { vt = 0.15; }
109             else if ( turnMag > 10 ) { vt = 0.10; }
110             else if ( turnMag >  3 ) { vt = 0.06; }
111             else if ( turnMag >  2 ) { vt = 0.00; }
112             else if ( turnMag >  1 ) { vt = 0.00; }
113             else                     { vt = 0.00; }
114
115         LastHeading = NewHeading;
116         vx = 0; vy = 0; vt *= turnDir;
117     }
118
119 //
120 // Stop sets the robot speed vector to zero. This is useful only in Autonomous
121 // mode. It should not be used elsewhere.
122 //
123     public static void Stop () {
124         vx = 0; vy = 0; vt = 0;
125     }
126
127 //
128 // These methods rotate the robot at a constant counter-clockwise speed and
129 // clockwise speed respectively. This is only useful in Autonomous mode.
130 //
131     public static void TurnLeftAtSpeed ( double Speed ) {
```

```
132         vx = 0; vy = 0; vt = +Speed;
133     }
134
135     public static void TurnRightAtSpeed ( double Speed ) {
136         vx = 0; vy = 0; vt = -Speed;
137     }
138
139     // public static void AdjustTurnSpeed( double Speed ) {
140     //     // double error = Speed - Navigation.GetTurnSpeed();
141     //     // LastPowerT  += error * 0.0001;
142     // }
143
144     // public static void DriveStraight ( double Speed, double Heading ) {
145
146     // }
147
148 }
```

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     }
37
38 }
```

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 //
27 //
28 //
29     public static void Reset () {
30         ElevArm     .Reset();
31         ElevClaw    .Reset();
32         ElevLift    .Reset();
33         ElevWrist   .Reset();
34     }
35
36     public static void Set ( double A, double C, double L, double W ) {
37         ElevArm     .SetPosition( A );
38         ElevClaw    .SetPosition( C );
39         ElevLift    .SetPosition( L );
40         ElevWrist   .SetPosition( W );
41     }
42
43 }
```

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;
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

```java
package frc.robot.Hardware;

import com.ctre.phoenix.sensors.CANCoder;

public class EncTalonFX {

    public CANCoder FalconEncoder;
    public final static int kUnitsPerRevolution = 2048;

    public EncTalonFX ( int CanBusID ) {
        FalconEncoder = new CANCoder( CanBusID );
    }

}
```

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  import edu.wpi.first.math.kinematics.SwerveModuleState;
6
7  public class Module {
8
9      EncTalonFX  SteerEncoder;
10     int         ModuleNumber;
11     String      ModuleName;
12     WPI_TalonFX DriveMotor;
13     WPI_TalonFX SteerMotor;
14
15     // PIDController PID;
16
17     public Module ( String ModuleName, int ModuleNumber ) {
18
19         // REMEMBER VALUES
20         this.ModuleName   = ModuleName;
21         this.ModuleNumber = ModuleNumber;
22
23         // ID'S FOLLOW A PATTERN BASED ON MODULE NUMBERS
24         int DriveMotorID = ModuleNumber *2 -1;
25         int SteerMotorID = ModuleNumber *2 -0;
26
27         // CONFIGURE MOTOR THROUGH SOFTWARE
28         // TalonFXConfiguration config = new TalonFXConfiguration();
29         // config.supplyCurrLimit.enable                    = false;
30         // config.supplyCurrLimit.currentLimit              = 30.0;
31         // config.supplyCurrLimit.triggerThresholdCurrent = 30.0;
32         // config.supplyCurrLimit.triggerThresholdTime     = 1.5;
33
34         // DEFINE AND CONFIGURE DRIVE MOTOR
35         DriveMotor = new WPI_TalonFX ( DriveMotorID );
36         DriveMotor.setNeutralMode( NeutralMode.Brake );
37         // driveMotor.configAllSettings ( config );
38
39         // DEFINE AND CONFIGURE STEER MOTOR
40         SteerMotor = new WPI_TalonFX ( SteerMotorID );
41         SteerMotor.setNeutralMode( NeutralMode.Brake );
42         // steerMotor.configAllSettings ( config );
43
44         // DEFINE STEER ENCODER
45         SteerEncoder = new EncTalonFX ( ModuleNumber );
46         // SteerEncoder.FalconEncoder.setPosition( 0 );
47
48         // PID CONTROLLER
49         // PID = new PIDController( 0.1, 0, 0 );
50         // PID.enableContinuousInput( 0, 360 );
51         // PID.setIntegratorRange( -0.5, 0 );
52         // PID.setTolerance( 5, 10 );
53     }
54
55     public void Display () {
56
57         // DRIVE MOTOR
58         // double DrivePercent = DriveMotor.getMotorOutputPercent();
59         // SmartDashboard.putNumber( ModuleName + "_Drive Percent", DrivePercent );
60
61         // STEER MOTOR
62         // double SteerPercent = SteerMotor.getMotorOutputPercent();
63         // SmartDashboard.putNumber( ModuleName + "_Steer Percent", SteerPercent );
64
65         // STEER ENCODER
```

```
66          // SmartDashboard.putNumber(ModuleName + " DIFF", diff );
67          // SmartDashboard.putNumber(ModuleName + " SMAL", s );
68      }
69
70      public void ResetDriveEncoder () {
71          // DriveMotor
72      }
73
74      public double GetDirection () {
75          return SteerEncoder.FalconEncoder.getAbsolutePosition();
76      }
77
78      public void Update( SwerveModuleState state ) {
79
80          // CALCULATE DRIVE VALUES
81          double DriveRatio = state.speedMetersPerSecond;
82          double reverse    = 1;
83
84          // CALCULATE TURN VALUES
85          double SP = state.angle.getDegrees(); // Desired state (Final)
86          double PV = GetDirection();           // Current state (Initial)
87              SP = ( SP + 360 ) % 360;          // Ensure SP is between 0 and 360
88              double diff = -( SP - PV );       // Why is this negated? Should setInverted have been used?
89
90          // SMALLEST ANGLE TO SWIVEL: -180 to 180
91          double minTurn = ( diff + 180 ) % 360 - 180;
92              double turnMag = Math.abs   ( minTurn );
93              double turnDir = Math.signum( minTurn );
94
95          // MINIMIZE WHEEL SWIVEL: +120 becomes -60
96          if ( turnMag > 90 ) {
97              turnMag  = 180 - minTurn; // Turn smaller angle
98              turnDir *= -1;            // and reverse swivel
99              reverse *= -1;            // and reverse drive.
100         }
101
102         // DETERMINE POWER USING PSEUDO PID CONTROLLER
103         double SteerRatio = 0;
104         if      ( turnMag > 20 ) { SteerRatio = 0.20; }
105         else if ( turnMag > 10 ) { SteerRatio = 0.10; }
106         else if ( turnMag >  5 ) { SteerRatio = 0.06; }
107         else if ( turnMag >  3 ) { SteerRatio = 0.04; }
108         else if ( turnMag >  2 ) { SteerRatio = 0.00; }
109         else if ( turnMag >  1 ) { SteerRatio = 0.00; }
110         else                     { SteerRatio = 0.00; }
111
112         // SET MOTOR CONTROLLERS
113         DriveMotor.setVoltage( DriveRatio * 10 * reverse );
114         SteerMotor.setVoltage( SteerRatio * 10 * turnDir );
115         // SteerMotor.set( TalonFXControlMode.PercentOutput, PID.calculate( SP, PV ) );
116
117         // SmartDashboard.putNumber(ModuleName + " DIFF", diff );
118         // SmartDashboard.putNumber(ModuleName + " SMAL", s );
119     }
120
121     public void SetToHeading ( double Angle ) {
122
123         // CALCULATE TURN VALUES
124         double SP = Angle;                // Desired state (Final)
125         double PV = GetDirection();       // Current state (Initial)
126             SP = ( SP + 360 ) % 360;      // Ensure SP is between 0 and 360
127             double diff = -( SP - PV ); // Why is this negated? Should setInverted have been used?
128
129         // DETERMINE POWER USING PSEUDO PID CONTROLLER
130         double SteerRatio = 0;
131         if      ( diff > 40 ) { SteerRatio = 0.50; }
132         else if ( diff > 20 ) { SteerRatio = 0.20; }
133         else if ( diff > 10 ) { SteerRatio = 0.10; }
```

```
134         else if ( diff >  5 ) { SteerRatio = 0.06; }
135         else if ( diff >  3 ) { SteerRatio = 0.04; }
136         else if ( diff >  2 ) { SteerRatio = 0.00; }
137         else if ( diff >  1 ) { SteerRatio = 0.00; }
138         else                  { SteerRatio = 0.00; }
139
140         SteerMotor.setVoltage( SteerRatio * 10 );
141     }
142
143 }
```

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
57     public static void Fail () {
58         AutonFinalTime = System.currentTimeMillis();
59         ReadyToAdvance = false;
60         Number         = 100;
61     }
62
63 //
64 // Get...Time methods are is useful in auton mode to determine the amount
65 // of time that the current stage or the entire auton process has been
```

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

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

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;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;

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;

    // ODOMETRY OBJECT
    // public static SwerveDriveOdometry
    //     Odometry;

    // 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();
```

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

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.SetWheelsToHeading( 90 );
9                  Stage.WaitForDuration( 1.00 );
10                 break;
11
12             case 1:
13                 Autopilot.DriveWest( 0.08 );
14                 Stage.WaitForDuration( 2.00 );
15                 break;
16
17             default:
18                 Stage.Last();
19                 break;
20         }
21     }
22
23     public static void Track_01 () {
24         switch ( Stage.Number ) {
25             case 0:
26                 break;
27
28             default:
29                 Stage.Last();
30                 break;
31         }
32     }
33
34     public static void Track_02 () {
35         switch ( Stage.Number ) {
36             case 0:
37                 break;
38
39             default:
40                 Stage.Last();
41                 break;
42         }
43     }
44
45     public static void Track_03 () {
46         switch ( Stage.Number ) {
47             case 0:
48                 break;
49
50             default:
51                 Stage.Last();
52                 break;
53         }
54     }
55
56 }
```

Default.java (Driver)

```java
package frc.robot.Driver;

import edu.wpi.first.wpilibj.Joystick;
import frc.robot.Hardware.Navigation;
import frc.robot.Hardware.Swerve;
import frc.robot.Mode.Teleop;

public class Default {

    public static void Periodic () {

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

        // SIMPLE JOYSTICK DEADBAND
        if ( Math.abs( Xratio ) < 0.15 ) { Xratio = 0; }
        if ( Math.abs( Yratio ) < 0.15 ) { Yratio = 0; }
        if ( Math.abs( Tratio ) < 0.20 ) { Tratio = 0; }

        if ( DriveStick.getRawButton( 7 ) ) {
            Navigation.Reset();
        }

        // SEND SPEEDS TO SWERVE CLASS
        Swerve.UpdateRobotRelative( Xratio, Yratio, Tratio );
    }

}
```