Contents

CHAPTER 1

Software Configuration

Programming and running the Onabots Robot requires five components. Each of these components requires the installation of software and configuration. Wherever possible, free open-source software is used.

- ☐ Coding Computer. The Onabots use multiple computers running Debian GNU/Linux. Additional software includes OpenJDK 8, Eclipse with WPI plugins, Git, and ssh client.
- ☐ Driver Station. A Windows computer is required to control the robot as well as configure the OpenMesh Radio and the RoboRIO.
- ☐ OpenMesh Radio. The radio may be configured for wireless control at home but is reconfigured at tournaments to be restricted to the field wireless system.
- ☐ Raspberry PI. This computer runs the Raspian OS, a Debian OS for the Raspberry PI. The only other software this computer uses is Python3 with OpenCV for vision processing.
- ☐ RoboRIO. This requires two pieces of software to be installed: firmware and Java 8. If the firmware is updated, Java 8 will need to be reinstalled.
- ☐ Server. Computer running Debian GNU/Linux. Provides a web server and a version control system using git.

Coding Computers

- OpenJDK 8
- Eclipse
- Git

Also recommend using a combination of Git server and GitHub for public code sharing.

I recommend keeping all documentation, vision code, and robot code in the same repository.

OpenMesh Radio

- Label radio with team number and year with piece of masking tape. This is needed at tournaments when the field judges configure the radio. Keep a radio configured for home use so that it does not have to be reconfigured. This also allows us to use the robot wirelessly during practice. Indicate whether this is the tournament or home radio.
- Configure the radio. This may need to be check later to see if there is a more up-to-date version. Note that 'IL' represents a version used in Israel.
- Radio is powered by the VRM 12V/2A port. Strip 5/16 inn from wires.
- For power, the center of the barrel is positive. Red should be connected to the center. Black is for the outside.

Firmware

Must be performed using a Windows computer.

1. Disable WiFi on Control Panel
2. Open FRC Radio Configuration Utility from desktop
3. Select Ethernet as network interface
4. Enter team number (e.g., 5534)
5. Make sure the radio is on OpenMesh
6. Press 'Load Firmware'
7. Follow on-screen directions pertaining to power cycling
8. After the firmware is loaded, wait for about 2 minutes until the power light has stopped flashing for about a minute.
9. Press 'Configure'
10. Done

CHAPTER 2

Calendar

The schedule for robotics season is all-year. However, certain task have expected dates due to knowledge of the competition, tasks, etc.

Include references to other documents. Preferably hyper-links.

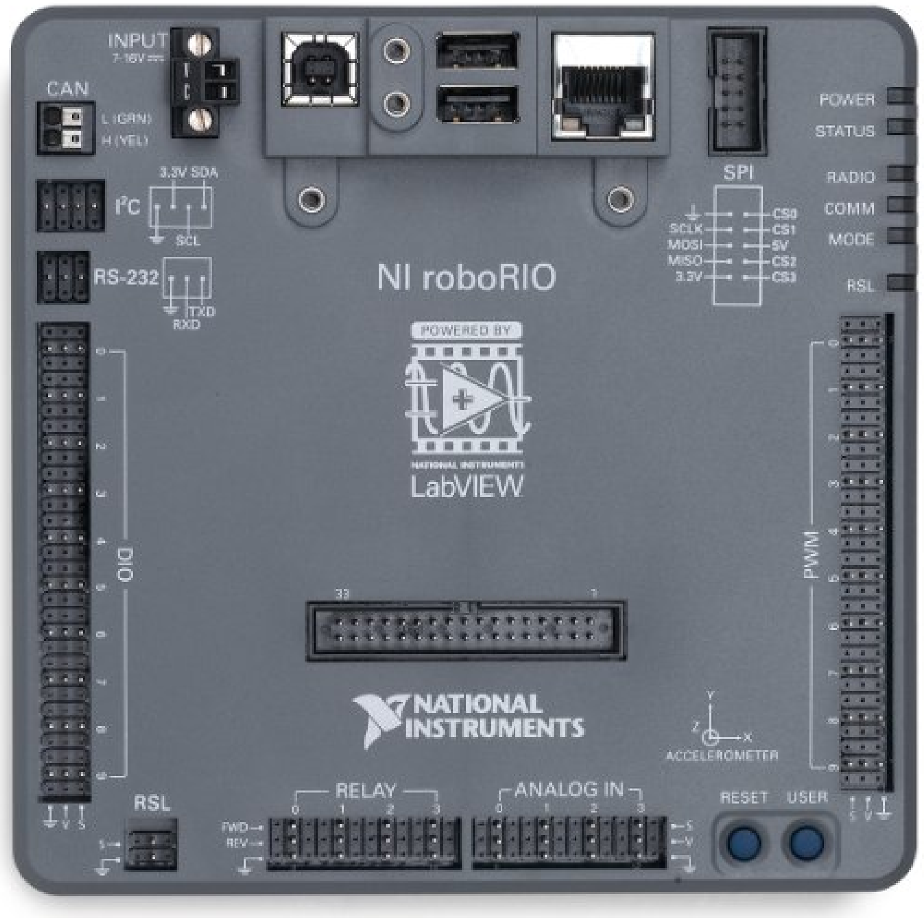
CHAPTER

3 Components

- Motor Controller
- Encoder
- Range Finder
- Camera
- Power Distribution Panel
- RoboRIO
- Pneumatics Control Module
- Voltage Regulator Module

RoboRIO

Some information about the RoboRIO.



Include information about the various ports, their naming.

INPUT Power for RoboRIO. Do not use screws facing top, these hold the black plastic to the RoboRIO. Screws to tighten wires are found on the left side of the black plastic.

Another picture showing the labelled parts of the roborio would be good.

Motor Controllers

Different types of motor controllers. List the different kinds as well as advantages and disadvantages of each. Include links to AndyMark with prices and descriptions.

A motor controller can control more than one motor but need a PWM split cable.

Talk about PWM, as Pulse Width Modulated. Varies the amount of time that a voltage is applied to the motor.

Range of motor controller values: -1 to 1 with 0 indicating no motion.

Actually good to set motor speed from an encoder instead of motor power. Speed changes with friction, internal resistance, and battery voltage.

Wiring requirements, what size gauge is needed?

Braking vs Coast mode

Radio

Encoders

Encoders should be used anywhere you need to control motor speed.

At the beginning of each of these files include the purpose and constraints for code that must exist there.

Include a program flow for each mode.

CHAPTER 4

Eclipse

4.1 Installation

1. Install Java 8

4.2 Configuration

4.3 Creating a Project

CHAPTER

5 Eclipse

Steps to follow. Include pictures and possibly links to video.

- Download and Install Java 8 from Oracle web site
- Download and Install Ecilpse for following site
- Install WPI plugin
- Configure Eclipse
- Firmware in RoboRIO
- Configure access points
- Update firmware (see yellow paper, Not IS - that is for Israeli teams)
- Vision tracking set-up with RaspberryPI

Chapter

6

Robot Code

Autonomous.java

```
package org.usfirst.frc.team5534.robot;

public class Autonomous {

    public static double  AutonStartTime;
    public static double  StageStartTime;
    public static double  AutonFinalTime;

    public static double  AutonStartDistance;
    public static double  StageStartDistance;

    public static int      StageNumber;
    public static boolean StillWorking;

    // ===

    public static void Init() {
        AutonStartTime = System.currentTimeMillis();
        StageStartTime = AutonStartTime;
        Navigation.Reset();
        StageNumber = 0;
    }

    public static void LastStage() {
        Drivetrain.Stop();
        StillWorking = true;
    }

    // ===

    public static void Periodic() {
        StillWorking = false;

            switch (Chooser.autonSelected) {

                case Chooser.autonLeftStation:
                    Station.LeftStation();
                    break;

                case Chooser.autonCenterStation:
                    Station.CenterStation();
                    break;

                case Chooser.autonRightStation:
                    Station.RightStation();
                    break;

                case Chooser.autonCrossBaseline:
                    Station.CrossBaseline();
                    break;

                case Chooser.autonTesting:
                 Station.Testing();
                 break;

                default:
                    break;

        }

        if ( StillWorking == false ) { NextStage(); }

    }

    // ===

    public static void NextStage() {

        // DRIVE POWER
        Drivetrain.PowerD = 0;
        Drivetrain.PowerT = 0;

        // RESET STAGE DISTANCE AND TIME
        StageStartDistance = Navigation.GetDistance();
        StageStartTime     = System.currentTimeMillis();

        // NEXT STAGE
        StageNumber++;
    }


    public static double GetAutonDuration() {
        return ( System.currentTimeMillis() - AutonStartTime )/1000.0;
    }


    public static double GetStageDuration() {
        return ( System.currentTimeMillis() - StageStartTime )/1000.0;
    }


    public static double GetStageDistance() {
        return Navigation.GetDistance() - StageStartDistance;
    }

    // ===

    public static void WaitForDistance( double distance ) {
        if ( GetStageDistance() < distance ) { StillWorking = true; }
    }


    public static void WaitForDuration( double duration ) {
        if ( GetStageDuration() < duration ) { StillWorking = true; }
    }


//  public static boolean WaitForHeading( double tolerance ) {
//      if ( Drivetrain.TargetMin<-tolerance || Drivetrain.TargetMax>tolerance ) {
//          StillWorking = true;
//          return true;
//      }
//      else {
//          return false;
//      }
//  }
//
//
//  public static boolean WaitForTarget( double tolerance ) {
//      if ( Drivetrain.TargetMin<-tolerance || Drivetrain.TargetMax>tolerance ) {
//          StillWorking = true;
//          return true;
//      }
//      else {
//          return false;
//      }
//
//  }

}
```

Autopilot.java

```
package org.usfirst.frc.team5534.robot;

public class Autonomous {

    public static double  AutonStartTime;
    public static double  StageStartTime;
    public static double  AutonFinalTime;

    public static double  AutonStartDistance;
    public static double  StageStartDistance;

    public static int      StageNumber;
    public static boolean StillWorking;

    // ===

    public static void Init() {
        AutonStartTime = System.currentTimeMillis();
        StageStartTime = AutonStartTime;
        Navigation.Reset();
        StageNumber = 0;
    }

    public static void LastStage() {
        Drivetrain.Stop();
        StillWorking = true;
    }

    // ===

    public static void Periodic() {
        StillWorking = false;

            switch (Chooser.autonSelected) {

                case Chooser.autonLeftStation:
                    Station.LeftStation();
                    break;

                case Chooser.autonCenterStation:
                    Station.CenterStation();
                    break;

                case Chooser.autonRightStation:
                    Station.RightStation();
                    break;

                case Chooser.autonCrossBaseline:
                    Station.CrossBaseline();
                    break;

                case Chooser.autonTesting:
                 Station.Testing();
                 break;

                default:
                    break;

        }

        if ( StillWorking == false ) { NextStage(); }

    }

    // ===

   public static void NextStage() {

      // DRIVE POWER
      Drivetrain.PowerD = 0;
      Drivetrain.PowerT = 0;

      // RESET STAGE DISTANCE AND TIME
      StageStartDistance = Navigation.GetDistance();
      StageStartTime     = System.currentTimeMillis();

      // NEXT STAGE
      StageNumber++;
   }


   public static double GetAutonDuration() {
      return ( System.currentTimeMillis() - AutonStartTime )/1000.0;
   }


   public static double GetStageDuration() {
      return ( System.currentTimeMillis() - StageStartTime )/1000.0;
   }


   public static double GetStageDistance() {
      return Navigation.GetDistance() - StageStartDistance;
   }

   // ===

   public static void WaitForDistance( double distance ) {
      if ( GetStageDistance() < distance ) { StillWorking = true; }
   }


   public static void WaitForDuration( double duration ) {
      if ( GetStageDuration() < duration ) { StillWorking = true; }
   }


// public static boolean WaitForHeading( double tolerance ) {
//    if ( Drivetrain.TargetMin<-tolerance || Drivetrain.TargetMax>tolerance ) {
//       StillWorking = true;
//       return true;
//    }
//    else {
//       return false;
//    }
// }
//
//
// public static boolean WaitForTarget( double tolerance ) {
//    if ( Drivetrain.TargetMin<-tolerance || Drivetrain.TargetMax>tolerance ) {
//       StillWorking = true;
//       return true;
//    }
//    else {
//       return false;
//    }
//
// }

}
```

Button.java

```java
package org.usfirst.frc.team5534.robot;

public class Button {

    public static final int GearGrab  = 1;
    public static final int GearDrop  = 2;

    public static final int GearRaise = 3;
    public static final int GearLower = 4;

    public static final int Climber   = 2;

}
```

Chooser.java

```
package org.usfirst.frc.team5534.robot;

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

import edu.wpi.first.wpilibj.I2C;
import edu.wpi.first.wpilibj.I2C.Port;

public class Chooser {

   public static final String autonDoNothing     = "Do Nothing";
   public static final String autonLeftStation   = "Left Station";
   public static final String autonCenterStation = "Center Station";
   public static final String autonRightStation  = "Right Station";
   public static final String autonCrossBaseline = "Cross Baseline";
   public static final String autonTesting       = "Testing";

   public static String autonSelected = autonTesting;


   public static AnalogInput autonChooser = new AnalogInput(
      Ports.AIO_AutonPotent );

    public static void GetSelected() {

      double voltage = autonChooser.getVoltage() / 5.00 * 100;

      if ( voltage < 10 ) { autonSelected = autonDoNothing;     }

      else if ( voltage < 30 ) { autonSelected = autonLeftStation;   }

      else if ( voltage < 50 ) { autonSelected = autonCenterStation; }

      else if ( voltage < 70 ) { autonSelected = autonRightStation;  }

      else if ( voltage < 90 ) { autonSelected = autonCrossBaseline; }

      else { autonSelected = autonTesting; }

      SmartDashboard.putNumber("VOLTAGE",  autonChooser.getVoltage() );

    }

}
```

Climber.java

```java
package org.usfirst.frc.team5534.robot;

import edu.wpi.first.wpilibj.Spark;

public class Climber {

    public static Spark ClimbMotor = new Spark( Ports.PWM_ClimbMotor );


    public static void Ascend() {
        ClimbMotor.set( Preferences.ClimbSpeed );
    }


    public static void Stop() {
        ClimbMotor.set( 0.00 );
    }

}
```

Dashboard.java

```
package org.usfirst.frc.team5534.robot;

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

public class Dashboard {

    // The dashboard package communicates with the SmartDashboard on the Driver Station.
    // An alternative would by to run a Python script on the DriverStation that communicates
    // with the robot using NetworkTables.

     public static NetworkTable DashboardNetworkTable = NetworkTable.getTable("Dashboard");


    public static void Double( String key, double value ) {
        SmartDashboard.putNumber( key, value );
        DashboardNetworkTable.putNumber( key, value );
    }


    public static void Update() {

        // MOTOR POWER
        Double("L␣Power",   Drivetrain.DrivePowerL      );
        Double("R␣Power",   Drivetrain.DrivePowerR      );

        // MOTOR SPEED
        Double("Thurst",    Navigation.GetDriveSpeed()  );
        Double("Speed",     Navigation.GetDriveSpeed()  );
        Double("Distance",  Navigation.GetDistance()    );

        double testing = DashboardNetworkTable.getNumber("Testing");
        Double("Testing", testing );

        // GYROSCOPE
        Double("Heading",   Navigation.GetHeading()     );

        // SONAR
        Double("Ranger",    Navigation.GetRange()       );

        // VISION
        Double("Score",     Vision.GetTargetScore()     );
        Double("X",         Vision.GetTargetX()         );
        Double("Y",         Vision.GetTargetY()         );

    }

}
```

Drivetrain.java

```java
package org.usfirst.frc.team5534.robot;

import edu.wpi.first.wpilibj.Spark;

public class Drivetrain {

    public static Spark DriveMotorLF = new Spark( Ports.PWM_DriveMotor_LF );
    public static Spark DriveMotorLR = new Spark( Ports.PWM_DriveMotor_LR );
    public static Spark DriveMotorRF = new Spark( Ports.PWM_DriveMotor_RF );
    public static Spark DriveMotorRR = new Spark( Ports.PWM_DriveMotor_RR );

    public static double DrivePowerL;
    public static double DrivePowerR;

    public static double PowerD = 0;
    public static double PowerT = 0;


    public static void DriveByPower( double drivePower, double turnPower ) {
        DrivePowerL = +( drivePower + turnPower );
        DrivePowerR = -( drivePower - turnPower );

        DriveMotorLF.set( DrivePowerL ); DriveMotorLR.set( DrivePowerL );
        DriveMotorRF.set( DrivePowerR ); DriveMotorRR.set( DrivePowerR );
    }


    public static void Stop() {
        PowerD = 0;
        PowerT = 0;
    }


    public static void DriveToTarget( double targetSpeed ) {

        double DriveSpeedError = targetSpeed - Navigation.GetDriveSpeed();

        if ( targetSpeed > 0 ) { PowerD = Math.max( PowerD,  0.18 ); }
        if ( targetSpeed < 0 ) { PowerD = Math.min( PowerD, -0.18 ); }

        PowerD = Math.max( PowerD, 0.18 );
        PowerD += DriveSpeedError * 0.0001;

        double NewHeading = Navigation.GetHeading() + Vision.GetTargetX()/14;
        double NextSpeed = Navigation.GetDelta( NewHeading );

        NextSpeed = Math.min( NextSpeed,  20 );
        NextSpeed = Math.max( NextSpeed, -20 );

        double SpeedError = NextSpeed - Navigation.GetTurnSpeed();

        PowerT += SpeedError * 0.0002;

    }


    public static void TurnToHeading( double TargetHeading ) {

        double NextSpeed = Navigation.GetDelta( TargetHeading );

            NextSpeed = Math.min( NextSpeed+5,  36 );
            NextSpeed = Math.max( NextSpeed-5, -36 );

        double SpeedError = NextSpeed - Navigation.GetTurnSpeed();

        if ( NextSpeed < 0 ) { PowerT = Math.min( PowerT, −0.18 ); }
        if ( NextSpeed > 0 ) { PowerT = Math.max( PowerT, +0.18 ); }

        PowerD = 0; PowerT += SpeedError*0.0002;

    }


    public static void TurnToTarget() {

        PowerD = 0;

        double CurrentHeading = Navigation.GetHeading();

        double NewHeading = CurrentHeading + Vision.GetTargetX() / 14;

        TurnToHeading( NewHeading );

    }


    public static void DriveToHeading( double TargetSpeed, double TargetBearing) {

        double DeltaDriveSpeed = TargetSpeed − Navigation.GetDriveSpeed();

        if ( TargetSpeed > 0 ) { PowerD = Math.max( PowerD,  0.18 ); }
        if ( TargetSpeed < 0 ) { PowerD = Math.min( PowerD, −0.18 ); }

        if ( DeltaDriveSpeed > 0 ) { PowerD += 0.0005; }
        if ( DeltaDriveSpeed < 0 ) { PowerD −= 0.0005; }

        double DeltaTurnAngle = Navigation.GetDelta( TargetBearing );

        PowerT = DeltaTurnAngle * 0.002;

    }


    public static void TurnAtSpeed( double targetSpeed ) {

        double delta = targetSpeed − Navigation.GetTurnSpeed();

        if ( targetSpeed > 0 ) { PowerT = Math.max( DriveMotorLF.get(),  0.18 ); };
        if ( targetSpeed < 0 ) { PowerT = Math.min( DriveMotorLF.get(), −0.18 ); };

        PowerD = 0; PowerT += delta * 0.0005;

    }

}
```

GearCollector.java

```
package org.usfirst.frc.team5534.robot;

import edu.wpi.first.wpilibj.DoubleSolenoid;

public class GearCollector {

    public static DoubleSolenoid GearGrabber = new DoubleSolenoid(
            Ports.PCM_GearGrabber[0],
            Ports.PCM_GearGrabber[1]);


    public static DoubleSolenoid GearLifter = new DoubleSolenoid(
            Ports.PCM_GearLifter[0],
            Ports.PCM_GearLifter[1]);


    public static void Grab() { GearGrabber.set( DoubleSolenoid.Value.kForward ); }


    public static void Drop() { GearGrabber.set( DoubleSolenoid.Value.kReverse ); }


    public static void Raise() { GearLifter.set( DoubleSolenoid.Value.kForward ); }


    public static void Lower() { GearLifter.set( DoubleSolenoid.Value.kForward ); }

}
```

Monitor.java

```java
package org.usfirst.frc.team5534.robot;

import java.io.File;
//import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;

public class Monitor {


    public static PrintWriter pw          = null;
    public static File        file        = new File("/home/lvuser/data.txt");
    public static long        initialTime = 0;


    public static void Open() {
        file.delete();
        initialTime = System.currentTimeMillis();
        try { pw = new PrintWriter( file ); } catch (IOException ioe) {}
    }


    public static void Close() {
        if ( pw != null ) {
            pw.close();
            pw = null;
        }
    }


    public static void Write() {
        if ( pw != null ) {
            // pw.print( (System.currentTimeMillis()-initialTime)/1000.0+" " );

            pw.print( Navigation.GetRange()+"␣" );
            pw.print( Vision.GetTargetX()+"␣" );
            pw.print( Navigation.GetHeading()+"␣" );
            pw.print( "\n" );
        }
    }

}
```

Navigation.java

```java
package org.usfirst.frc.team5534.robot;

import edu.wpi.first.wpilibj.ADXRS450_Gyro;
import edu.wpi.first.wpilibj.Encoder;
import edu.wpi.first.wpilibj.Ultrasonic;

public class Navigation {

    public static ADXRS450_Gyro Gyroscope = new ADXRS450_Gyro();

    public static Encoder DriveEncoderR = new Encoder(
            Ports.DIO_DriveEncoderR[0],
            Ports.DIO_DriveEncoderR[1],
            true); // Reverse direction on right side

    public static Ultrasonic Sonar = new Ultrasonic(
            Ports.DIO_RangeFinder[0],
            Ports.DIO_RangeFinder[1]);

    // ===

    public static void Initialize() {
        DriveEncoderR.setDistancePerPulse(0.009523);
        DriveEncoderR.reset();
        Sonar.setAutomaticMode(true);
        Gyroscope.calibrate();
        Gyroscope.reset();
    }


    public static void Reset() {
        DriveEncoderR.setDistancePerPulse(0.009523);
        DriveEncoderR.reset();
        Sonar.setAutomaticMode(true);
        Gyroscope.reset();
    }

    // ===

    public static double GetDistance() {
        return DriveEncoderR.getDistance();
    }


    public static double GetDelta( double targetHeading ) {
        return ( targetHeading - Gyroscope.getAngle() + 180 ) % 360 - 180;
    }


    public static double GetDriveSpeed() {
        return DriveEncoderR.getRate();
    }


    public static double GetHeading() {
        return Gyroscope.getAngle();
    }


    public static double GetRange() {
        return Sonar.getRangeInches();
    }


    public static double GetTurnSpeed() {
        return Gyroscope.getRate();
    }

}
```

Pilot.java

```java
package org.usfirst.frc.team5534.robot;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.XboxController;

public class Pilot {

    public static Joystick DriveStick = new Joystick( Ports.USB_ArcadeStick );


    public static double GetThrottle() {
        return DriveStick.getThrottle();
    }


    public static double GetThrust() {
        return -DriveStick.getY();
    }


    public static double GetTurn() {
        return DriveStick.getX() / 3;
    }


    public static double GetTwist() {
        return DriveStick.getTwist() / 3;
    }


    public static boolean GetButton( int button ) {
        return DriveStick.getRawButton( button );
    }

    public static XboxController Xbox = new XboxController( Ports.USB_Xbox );

}
```

Ports.java

```
package org.usfirst.frc.team5534.robot;

public class Ports {

    public static final int     AIO_AutonPotent    = 3;

    // ===

    public static final int[]  DIO_DriveEncoderR  = {0,1};
    public static final int[]  DIO_RangeFinder    = {7,6};

    // ===

    public static final int[]  PCM_GearGrabber    = {0,1};
    public static final int[]  PCM_GearLifter     = {2,3};

    // ===

    public static final int     PWM_DriveMotor_LF  = 7;
    public static final int     PWM_DriveMotor_LR  = 8;
    public static final int     PWM_DriveMotor_RF  = 2;
    public static final int     PWM_DriveMotor_RR  = 3;

    public static final int     PWM_ClimbMotor     = 4;

    // ===

    public static final int     USB_ArcadeStick    = 1;
    public static final int     USB_Xbox           = 2;

}
```

Preferences.java

```java
package org.usfirst.frc.team5534.robot;

public class Preferences {

    public static double ClimbSpeed = 1.00;

    // ===

    public static double CameraCenterX = 320+25;
    public static double CameraCenterY = 200;

}
```

Robot.java

```java
package org.usfirst.frc.team5534.robot;

import edu.wpi.first.wpilibj.IterativeRobot;

public class Robot extends IterativeRobot {

    @Override
    public void robotInit() {
        Vision.Initialize();
    }

    @Override
    public void robotPeriodic()         {
        Drivetrain.DriveByPower( Drivetrain.PowerD, Drivetrain.PowerT );
        Dashboard.Update();
        Monitor.Write();
    }


    @Override
    public void disabledInit()          {
        Monitor.Close();
    }

    @Override
    public void disabledPeriodic()      {}


    @Override
    public void autonomousInit()        {
        Autonomous.Init();
    }

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


    @Override
    public void teleopInit()            {
        Teleop.Init();
        Monitor.Open();
    }

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


    @Override
    public void testInit()              {}

    @Override
    public void testPeriodic()          {}

}
```

Station.java

```
package org.usfirst.frc.team5534.robot;

public class Station {


    public static void LeftStation() {

    }


    public static void CenterStation() {

    }


    public static void RightStation() {

    }


    public static void CrossBaseline() {

    }


    public static void DoNothing() {

    }


    public static void Testing() {
        switch (Autonomous.StageNumber) {

            case 0:
                Navigation.Reset();
                break;

            case 1:
                Autopilot.DriveForwardToDistance(20, 18*12);
                break;

            case 2:
                Drivetrain.TurnToHeading( 90 );
                break;

            case 3:
//              Autopilot.DriveToSurface( 24, 12 );
                break;

            default:
                Autonomous.LastStage();
                break;
        }

    }

}
```

Teleop.java

```java
package org.usfirst.frc.team5534.robot;

public class Teleop {


    public static void Init() {
        Navigation.Reset();
    }


    public static void Periodic() {



//      Drivetrain.PowerD = Pilot.GetThrust();
//      Drivetrain.PowerT = Pilot.GetTurn();
//
//      if ( Pilot.GetButton( Button.Climber )) {
//          Climber.Ascend();
//      }
//      else {
//          Climber.Stop();
//      }
//
//
//      // ===
//
//
//      if ( Pilot.Xbox.getTriggerAxis( Hand.kLeft ) != 0 ) {
//          GearCollector.Lower();
//      }
//      else {
//          GearCollector.Raise();
//      }
//
//
//      if ( Pilot.Xbox.getTriggerAxis( Hand.kRight ) != 0 ) {
//          GearCollector.Drop();
//      }
//      else {
//          GearCollector.Grab();
//      }


        if ( Pilot.GetButton(1) ) {
            Navigation.Initialize();
            Drivetrain.PowerD = 0;
            Drivetrain.PowerT = 0;
        }

        else if ( Pilot.GetButton(2) ) {
            Drivetrain.TurnToHeading( 180 );
        }

        else if ( Pilot.GetButton(3) ) {
            Drivetrain.DriveToHeading( 24, 0 );
        }

        else if ( Pilot.GetButton(4) ) {
            Drivetrain.TurnToTarget();
        }

        else if ( Pilot.GetButton(5) ) {

        }

        else if ( Pilot.GetButton(6) ) {
            if ( Navigation.GetRange() > 16 ) {
                Drivetrain.DriveToTarget( 20 );
            }
            else {
                Drivetrain.Stop();
            }
        }

        else {
            Climber.Stop();
            Drivetrain.PowerD = Pilot.GetThrust();
            Drivetrain.PowerT = Pilot.GetTwist();
        }

    }


}
```

Vision.java

```java
package org.usfirst.frc.team5534.robot;

import edu.wpi.first.wpilibj.CameraServer;
import edu.wpi.first.wpilibj.networktables.NetworkTable;
import edu.wpi.cscore.UsbCamera;

public class Vision {

    public static UsbCamera PilotCamera;

    public static void Initialize() {
       PilotCamera = CameraServer.getInstance().startAutomaticCapture();
       PilotCamera.setFPS(30);
    }


    // ===


    public static double TargetScore;
    public static double TargetX;
    public static double TargetY;


    public static NetworkTable VisionNetworkTable = NetworkTable.getTable("Vision");


    public static double GetTargetScore() {
      return VisionNetworkTable.getNumber("Score", 0 );
    }


    public static double GetTargetX() {
      return VisionNetworkTable.getNumber("X", 0 ) - Preferences.CameraCenterX;
    }


    public static double GetTargetY() {
      return VisionNetworkTable.getNumber("Y", 0 ) - Preferences.CameraCenterY;
    }

}
```

CHAPTER

7

Java Tutorial

Notes on Structure

Core Files

- Robot.java
- Button.java
- Port.java
- Chooser
- Display
- ModeRobot . . . Init() and Periodic()
- ModeAutonomous . . . stub that determines which autonomous code to execute. Contains Init() and Periodic(). Calls correct autonmous mode
- ModeTeleop . . . Init() and Periodic()
- ModeDisabled . . . Optional. Contains Init() and Periodic()
- ModeTest . . . Optional. Contains Init() and Periodic()
- AutonCenterStation
- AutonRightStation
- AutonLeftStation
- AutonCrossBaseline
- AutonDoNothing
- State
- Drivetrain . . . moves the robot
- Navigation . . . code for gyroscope, visiontracking, camera, range finders
- Odometer
- Visiontracking . . .
- Autopilot . . . read input from gyroscope, encoders, visiontracking, sonic rangers, etc. Includes code to make decisions based upon Navigation

  Some examples include. These are the things the robot must do on its own for movement DriveTowardSpring()

  TurnToBearing()

  TurnLeft()

  TurnRight()

  TurnTowardSpring()

  DriveForward()

  DriveReverse()
- Pilot . . . reads input from joystick and buttons