FRC PROGRAMMING

PROGRAMMING FOR FIRST ROBOTICS 2018

LABVIEW, JAVA, C++

LABVIEW

- Labview is a graphical programming language
- Developed by National Instruments, who makes the RoboRio, and built to run on it as native code
- Program segments are accessed by VI files
  - VI panel shows variables and other information back to the program
  - VI code is separate, which uses a flow-chart style development
- Command and Control programming framework
- Global variables can be accessed by any vi
- Dashboard can be programmed using the same language
- Order of operation can be problematic

JAVA

- Object Oriented, text based programming framework
- Deploys code to run on the Java VM on the Robo Rio
- Time based robot
- Uses Subsystem / Command framework
- Can easily chain commands together
- Debugging is graphical
- Easier to work with code in teams (merge, diff)

C++

- Very similar to Java in code styles and functionality
- Requires header files defining the code
- More complex syntax than Java (references)
- Compiles to run as native code on the Robo Rio

JAVA BASICS – VARIABLES

- declare a variable by entering the datatype and name. Can be a simple or complex type.
  - Ex: int FRCTeam;
- Now we have declared it, so we can set our team to 3951 by entering
  - FRCTeam = 3951;
- Notice, the end has a semi-colon. This tells us the end of the statement, so it can span multiple lines.
- You can also declare the variable and set the value at the same time
  - Int FRCTeam = 3951;

JAVA BASICS – VARIABLES

- int FRCTeamNumber = 3951;
- String FRCTeam = “Suits”;
- double ControllerSpeed = 0.5;
- boolean CubeInArms = true;

JAVA BASICS – MATH

- double Speed = 0.5;
- Speed = Speed * 2; (Speed = 1)
- Speed *= 2; (Speed = 2)
- Speed = Speed / 4 (Speed = 0.5)
- Speed = Speed – 0.25; (Speed = 0.25)
- Speed = Speed + 0.25; (Speed = 0.5)
- Speed++; (Speed = 1.5)

JAVA BASICS - LOGIC

- double Speed = 0.5
- boolean Stopped = false;
- if(Speed > 0) {
  Stopped = false;
  } else {
  Stopped = true;
  }
- Stopped = (Speed == 0);
- boolean GoingSlow = false;
- if(Speed > 0 && Speed < 0.1) {
  GoingSlow = true;
  }
- boolean GoingSlow = (Speed > 0 && Speed < 0.1);
- boolean NeedToMoveFaster = false;
- If (GoingSlow || Stopped) {
  NeedToMoveFaster = true;
  }

JAVA BASICS – FOR LOOPS

```
int Max = 10;
int Total = 0;
for (int Current = 0; Current < Max; Current++) {
        Total = Total + Current;
}
```

- First value of Current is 0, and the last is 9, incrementing by 1 each time
- Total = 0, 1, 3, 6, 10……

JAVA BASICS – WHILE LOOPS

- ```
  int Max = 10;
  int Total = 0;
  int Current = 0;
  while(Current < Max) {
          Total = Total + Current;
          Current++;
  }
  ```
- First value of Current is 0, and the last is 9, incrementing by 1 each time
- Total = 0, 1, 3, 6, 10……
  ```

JAVA BASICS – WHILE LOOPS

- **int Max = 10;**
  int Total = 0;
  int Current = 0;
  while(Current < Max) {
  Current++;
  Total = Total + Current;
  }
- First value of Current is 1, and the last is 10, incrementing by 1 each time
- Total = 1, 3, 6, 10……

JAVA BASICS - FUNCTIONS

- Functions contain a block of code to execute
- Includes scope, return type, and parameters
- **public boolean GoingSlow(int Speed) {**

  return Speed <= 0.25;

  }
- boolean AreWeGoingSlow = GoingSlow(Speed);

JAVA ADVANCED – CLASSES

- Classes are complex data types for variables. They can include functions and variables.
- Can store data inside and provide accessible scope
  - public variables or functions – anything can access
  - private variables or functions – only code inside the class can access
- Can inherit and extend other data types
  - The Robot class that contains the main loop of the robot inherits a TimedRobot
  - The Climber subsystem extends the Subsystem data type
- Has a constructor which determines what happens when the class is created using the new syntax
  - DriveTrain drivetrain = new Drivetrain();
  - WPI_TalonSRX frontLeftMotor = new WPI_TalonSRX(RobotMap.DRIVE_LEFT_FRONT_MOTOR_CANID);
- Access properties by using the period after assigned
  - drivetrain.stopMotor();
- Can override inherited functions

FRC JAVA LAYOUT – ROBOT.JAVA

- Main loop for the robot
- Initialize subsystems
- Periodic loop for teleop
  - Calls scheduler
- Autonomous initialization function
- Autonomous Periodic function

FRC JAVA LAYOUT – ROBOTMAP.JAVA

- Declares static variables that are set once and not changed
  - Global Variables!
  - Robot values that do not change, like controller IDs.

FRC JAVA LAYOUT – OI.JAVA

- Operator interface
- Store IDs of buttons and joysticks as variables
  - Reference by name, not number when initializing!
- Declares joysticks for use in other functions (drive)
- Binds commands to joystick buttons
  - whenPressed / whenReleased
  - whileHeld
  - cancelWhenActive

FRC JAVA LAYOUT - SUBSYSTEMS

- Declares and initializes motors / sensors
- Create functions to perform tasks
  - Check encoder / Gyro
  - Run motor
- Can have a default command that always runs if none other
  - Like reserved if not set
  - Used to bind joysticks
    - DriveTrain waits for the joystick to be pressed and will drive
    - Climber waits for up joystick to be pressed to raise.

SUBSYSTEM

```java
public class Arms extends Subsystem {
    private WPI_VictorSPX armWheelLeftMotor;
    private WPI_VictorSPX armWheelRightMotor;
    private SpeedControllerGroup armWheelMotors;
    private WPI_VictorSPX armPositionMotor;
    private Encoder armPositionEncoder;
    private DigitalInput armPositionLimitSwitch;
    private AnalogInput armPositionSensor;


    public Arms() {
        super("Arms");
        armWheelLeftMotor = new WPI_VictorSPX(RobotMap.ARM_WHEEL_LEFT_MOTOR_CANID);
        armWheelRightMotor = new WPI_VictorSPX(RobotMap.ARM_WHEEL_RIGHT_MOTOR_CANID);
        armWheelMotors = new SpeedControllerGroup(armWheelLeftMotor, armWheelRightMotor);
        armPositionMotor = new WPI_VictorSPX(RobotMap.ARM_POSITION_MOTOR_CANID);
        armPositionEncoder = new Encoder(RobotMap.ARM_POSITION_ENCODER_A_CHANNEL_DIO_INPUT,
                RobotMap.ARM_POSITION_ENCODER_B_CHANNEL_DIO_INPUT);
        armPositionEncoder.reset();
        armPositionLimitSwitch = new DigitalInput(RobotMap.ARM_POSITION_LIMIT_SWITCH_DIO_INPUT);
        armPositionSensor = new AnalogInput(RobotMap.ARM_POSITION_SENSOR_ANALOG_INPUT);
    }

    public void RunArmWheels(double speed)
    {
        armWheelMotors.set(speed);
    }

    public boolean CubeInArms()
    {
        //trigger if voltage > 5?
        return armPositionSensor.getAverageVoltage() > 5;
    }

    public void StopArmWheels() {
        armWheelMotors.stopMotor();
    }

    public void MoveArmsTowardsPosition(int position) {
        //lower at speed of -0.2
        double speed = -0.2;
        //is the encoder value higher than where we need to go?
        //higher = arm is below. 0 = up position, 130 = down.
        if(GetArmPosition() > position)
            speed = 1;
        MoveArms(speed);
    }
```

```java
    public int GetArmPosition() {
        return armPositionEncoder.get();
    }

    public void ResetArmEncoder() {
        armPositionEncoder.reset();
    }
    public void MoveArms(double speed) {
        armPositionMotor.set(ControlMode.PercentOutput, speed);
    }

    public void ArmsToSpit() {
        armPositionMotor.set(ControlMode.Position, 25);
    }

    public void ArmsToBottom() {
        armPositionMotor.set(ControlMode.Position, 130);
    }
    public void StopArms() {
        armPositionMotor.stopMotor();
    }

    public int ArmPosition() {
        return armPositionMotor.getSelectedSensorPosition(0);
    }

    public boolean ArmsAtTop() {
        // top = 0 with deadband of 5
        return armPositionEncoder.get() < 5;
    }
    public boolean ArmsAtSpit() {
        //target 25 with deadband of 5
        return armPositionEncoder.get() < 30 && armPositionEncoder.get() < 20;
    }
    public boolean ArmsAtBottom() {
        //target 130 with deadband of 5
        return armPositionEncoder.get() < 135 && armPositionEncoder.get() < 125;
    }
    public boolean ArmsAtLimitSwitch()
    {
        return armPositionLimitSwitch.get();
    }

    @Override
    protected void initDefaultCommand() {
        // TODO Auto-generated method stub

    }
```

FRC JAVA LAYOUT - COMMANDS

- Commands can initialize with parameters (constructors)
- Commands can be a command group (chained commands!)
- Can require one or more subsystems (interrupt other running commands)
- Overrides functions when the command Executes, Ending, or Interrupted
- Overrides a function to determine whether or not the command should be done

COMMAND - SPIT

- This command is meant to be bound to a joystick button with whilePressed (stops when released)
- Has 2 different constructors
- Requires the arms subsystem
- It can be set with a timeout, or set to 1 if none set
  - Great for autonomous mode! Spit for 5 seconds.
- It is interruptible

```
package org.usfirst.frc.team3951.robot.commands;

import org.usfirst.frc.team3951.robot.OI;

public class Spit extends  Command {

    public Spit() {
        this(1);
    }
    public Spit(double timeout) {
        super("Spit");
        requires(Robot.arms);
        setInterruptible(true);
        //if no timeout set, set to 1.
        if(timeout <= 0)
            timeout = 1;
        setTimeout(timeout);
    }

    //run in full reverse
    @Override
    protected void execute() {
        Robot.arms.RunArmWheels(-1);
    }

    //run until the arm spit button is release
    @Override
    protected boolean isFinished() {
        //chew until they release the button.
        return isTimedOut();
    }

    //at the end, stop the motor.
    @Override
    protected void end() {
        Robot.arms.StopArmWheels();
    }

    //if the command is interrupted(cancelled), stop the motor.
    @Override
    protected void interrupted() {
        Robot.arms.StopArmWheels();
    }
}
```

COMMAND GROUP - AUTONOMOUS CENTER

```
public class AutonomousCenter extends CommandGroup {
    public AutonomousCenter() {
        //run the arms up for 15 seconds, or until we interrupt the command with the next one.
        addParallel(new ArmsToSpitHold(15));
        //drive forward 45 inches, timeout at 5 seconds
        addSequential(new DriveForDistance(RobotMap.DRIVE_DISTANCE_SPEED, 45, 5));

        String gameData;
        gameData = DriverStation.getInstance().getGameSpecificMessage();
        boolean TeamSwitchLeft = false;
        if(gameData.length() > 0)
        {
            TeamSwitchLeft = gameData.charAt(0) == 'L';
        }
        if(TeamSwitchLeft == true) {
            //rotate left 90 degrees, for 5 sec max, deadband 5 degrees
            addSequential(new RotateDegrees(90, Direction.LEFT, 5, RobotMap.ROTATE_DEGREES_DEADBAND));
            //drive forward 75 inches, timeout at 5 seconds
            addSequential(new DriveForDistance(RobotMap.DRIVE_DISTANCE_SPEED, 75, 5));
            //rotate back to 0
            addSequential(new RotateToDegrees(0, Direction.RIGHT, 5, RobotMap.ROTATE_DEGREES_DEADBAND));

        } else {
            ////rotate left 90 degrees, for 5 sec max, deadband 5 degrees
            addSequential(new RotateDegrees(90, Direction.RIGHT, 5, RobotMap.ROTATE_DEGREES_DEADBAND));
            //drive forward 49 inches, timeout at 5 seconds
            addSequential(new DriveForDistance(RobotMap.DRIVE_DISTANCE_SPEED, 49, 5));
            //rotate back to 0
            addSequential(new RotateToDegrees(0, Direction.LEFT, 5, RobotMap.ROTATE_DEGREES_DEADBAND));

        }

        //drive forward 66 inches, timeout at 5 seconds
        addSequential(new DriveForDistance(RobotMap.DRIVE_DISTANCE_SPEED, 66, 5));
        addSequential(new Spit());
    }
}
```