XRP Workshop Volunteer Details

Thank you for helping us support growing the interest and knowledge of our younger generation in STEM. For the XRP Workshop, your main duties will be helping the kids/teams with their programming (Java based). Below is some information that should help guide you through that process. If you are ever in doubt, please reach out to one of the head mentors (Zhiquan Yeo or Joe Pokorny) and they can provide next steps.

**Get to know the Hardware:**  
We are using the [XRP (Experimental Robotics Platform)](https://www.sparkfun.com/products/22230). This comes with a chassis, wheels, roller balls, drivetrain motors (and encoders for them), one servo motor with arm, color sensor, range finder, and a raspberry pi pico microcontroller. We will also supply 4 AA Batteries and a usb cord to connect to the laptop (for robot program uploading).

**Steps for setting up and connecting the XRP:**Physically build the robot using the steps here (note the video is really useful): <https://xrpusersguide.readthedocs.io/en/latest/course/building.html>

In order to upload your code, you will need to follow the instructions here:  
<https://docs.wpilib.org/en/latest/docs/xrp-robot/index.html>

Most important is to hold “BOOTSEL” button, then press and release “RESET” button. This should enter the XRP into upload mode. Drag and drop the uf2 file (<https://github.com/wpilibsuite/xrp-wpilib-firmware/releases/download/v1.0.0/xrp-wpilib-firmware-1.0.0.uf2>) onto the robot. Once this is done, remove the usb cable, and cycle the robot’s power. A new access point should show in your list of networks starting with XRP (if in doubt, enter upload mode with USB cable connected to your laptop and open xrp-status.txt on the PICODISK drive. The text file will have the exact SSID).

**Programming**  
We will be using Java and a style called command-based programming, see here:  
<https://docs.wpilib.org/en/2022/docs/software/commandbased/index.html>

The XRPReference example is in this style already and we plan to build upon it.

There are several classes in wpilib that you can leverage. You can find the documentation on them here:  
<https://first.wpi.edu/wpilib/allwpilib/docs/release/java/edu/wpi/first/wpilibj/package-summary.html>

And the complete list here:  
<https://first.wpi.edu/wpilib/allwpilib/docs/release/java/index.html>

Note, because the XRP has limited hardware capabilities, we have special classes for our motors (XRPMotor) and servos (XRPServo).

Our suggestion is to leverage the example (XRPReference documented steps to obtain it here <https://docs.wpilib.org/en/latest/docs/xrp-robot/programming-xrp.html>) as a starting point.

Note we leverage Command-Based Programming (more on that here  
<https://docs.wpilib.org/en/stable/docs/software/commandbased/index.html>). Command-Based Programming is based on the premise that you have a set of commands that you want to take on some subsystems (hardware). For example, you may want a robot to drive forward for 5 seconds. In command-based programming you would construct a “drive forward” command that could take time as an input. This command would act on the subsystem drivetrain which is the hardware that will be used to move the robot. This can then be extended to have a sequence of commands (drive forward, turn right, drive forward, etc) or commands in parallel which act on various parts of your hardware (drive forward and lift arm concurrently for example). Note, a subsystem can only act on one command at a time (or in other words, only one command can have control of the subsystem at a time). Additionally, many more examples are given in the above link if you’d like to learn more.

**File Breakdown:  
  
RobotContainer.java** - Where you will instantiate the subsystem(s) and specify the default command(s) for each.

**Robot.java** - skeleton of the different robot states you can be in based on the Mode that is active (Disabled, Teleoperated, Autonomous, or Test). There is an init and periodic for each. The init is run only once when you enter that Mode where the periodic is repeatedly ran in a loop while remaining in that state. You can specify the command(s) you want to run for each mode (note, cancelling a command restores the default to that subsystem).

**subsystem folder** - where all your “hardware” components will be. For example, Drivetrain.java represents the interfacing with the hardware for your wheels. It is recommended that the students create a new subsystem here if they decided to add an actuator (Arm.java, Lifter.java, Intake.java, etc)

**commands folder** - where all the commands you want to provide to the robot are. These commands can be independent or done as a grouping. For example, AutonomousDistance.java shows how you can create a set of commands to be run in sequence.

**Troubleshooting:**

1. **When in doubt, try changing the batteries** **-** (4 AAs on bottom of XRP). New batteries solve a lot of issues that just don’t make sense (including spotty or dropped wifi connection).
2. **Simulation build error** **-** “WPIHaljni” dependency path not found  
   We are still working on this. If you see it, please call a head mentor over to capture the details. In the meantime, the workaround is to not simulate from the option in the drop-down, but rather after the failure, in the terminal in VS Code, type:  
   ./gradlew simulateJavaRelease
3. **Simulation build error** **-** All others  
   Something is wrong with the Java code. Please investigate the error outputs.
4. **Robot not moving -** 
   1. First check to see if the robot is connected using <http://192.168.42.1:5000>
      1. If you see a webpage, you can reach your robot
   2. Ensure this is set in build.gradle:  
        
      wpi.sim.envVar("HALSIMXRP\_HOST", "192.168.42.1")  
      wpi.sim.addXRPClient().defaultEnabled = true
   3. Ensure you are using “Simulate Robot Code”
   4. In Simulator UI, check that you are in “**Teleoperated**” mode
   5. In Simulator UI, check that your Joystick is mapped to **Joystick[0]** . If it says “Unassigned”, drag-and-drop the Logitech/Xbox from the “System Joysticks” on the left.
   6. In Simulator UI, verify that pressing buttons and joystick knobs do in fact light up in your Joystick[0] area. If not, try unplugging, and switching the game interface mode switch on the back of the controller from “X” to “D” or “D” to “X”. This switch changes the button/axis mappings for the controller.
   7. Verify the code. Did you map the buttons correctly in code (most likely RobotContainer or a command file)? Did you make sure that the command is executed (either directly or via a defaultCommand)? If it is to run in a loop continuously, did you ensure your command has an override isFinished returning false (like ArcadeDrive)?
   8. Make sure you are using XRPMotor for your motors and not SparkMax or any standard motor controllers for the RoboRio.
   9. See point #1 again
5. **Servo not moving -** 
   1. Make sure you are using XRPServo class and not Servo
   2. Make sure phyical servo is connected to board
   3. Make sure the exact servo physical channel is specified correctly on object construction.
   4. Make sure to set the default command of the servo/arm subsystem is set in RobotContainer initialization.
   5. Make sure the servo/arm command isFinished method returns false so that it's always running.
   6. If using the setAngle() method from the example, make sure to use values between 0 and 180 (in degrees).
   7. See point #1 again