Module Testing

Improve FTC Hardware/Software Cooperation

Presented by: Team ToborTech

Team 8610 Tobortech

Veteran team, this is our 5th season

Went to worlds last year

So we'll be excited to share some of our knowledge on module testing

What is Module Testing?

At its core, module testing is one thing: the testing of an individual subsystem of the robot to ensure its accuracy before putting the whole robot together

Why do module testing?

Module testing allows a team to bypass common delays during the build process, such as:

- The robot being occupied because the hardware needs to be fixed
- The robot being occupied because another subsystem is having its software debugged

It also has its own advantages:

- Faster development of the software for a new subsystem
- Faster fixing of software bugs associated to one subsystem
- Easier unit testing (testing the motion)

Additional Benefits

Simplifies Integration (module testing verifies accuracy code before integration)

Module Testing Allows Better Documentation (understanding of module is better in the smaller scope)

Allows Better Designs (hardware better coordinates with the software)

Eliminates "Coder's Block" (know exactly what you are trying to accomplish with the sample of code)

Helps Manage Harder Tests (testing complicated modules on the entire robot is difficult)

Prevents Software Bottleneck (code is tested and works before integration)

Improves Efficiency Of Development Cycle

The Structure of the Program

To allow for easier modular testing, we had a specific structure to our code:

- Start with a “hardware” class
  - Holds both hardware declarations and utility functions
  - Two examples exist in FTC’s external samples: hardwareK9Bot and hardwarePushbot
- Other programs reference it by creating a “robot” object to quickly reference the class

Hardware Class

The hardware file has two main functions:

- Define all the constants and variables to run a device
  - The hardware needs to be recognized as an object in the code
  - We recommend declaring constants for servo positions
  - Any variables to improve functionality that are used across multiple functions should also be defined in hardware
- Define all functions you could need to run a device
  - The hardware needs to be initialized
  - Simple motions that are used repeatedly (turning, going a set distance/time) should have a function
  - Motors and servos that act in response to a sensor will need a function

Having an OpMode reference the hardware file

Referencing the hardware file only takes one/two steps:

Step 1: Define a “robot” object to reference the hardware file and initialize it

```
robotHardware robot = new robotHardware();
robot.init();
```

Step 2: reference the object when you want to call a function/constant

```
robot.turnLeft(power, degree);
robot.SERVO_KICK_POSITION;
```

Unit Testing

Testing out basic movement operations for each module is essential for a good autonomous, in particular the chassis. It is best to test these operations with its own dedicated program

- Chassis operations we will demonstrate:
  - Move Straight for Distance
  - Right Turn Degree
  - Left Turn Degree

Motion operations should be tested at various powers to see how accurate they are at higher speeds

```
if (robot.use_minibot) { // Unit Test for minibot
    if (gamepad1.dpad_up) { // forward 24 inches
        robot.StraightIn(speedscale, in: 24);
    } else if (gamepad1.dpad_left) { // left turn 90 degree
        robot.TurnLeftD(speedscale, degree: 90);
    } else if (gamepad1.dpad_right) { // right turn 90 degree
        robot.TurnRightD(speedscale, degree: 90);
    } else if (gamepad1.dpad_down) { // backward 24 inches
        robot.StraightIn( power: -1 * speedscale, in: 24);
    }
}
```

Tune-Up Program

What is it?

It's a tool that allows you to manually adjust servo positions and test autonomous functions

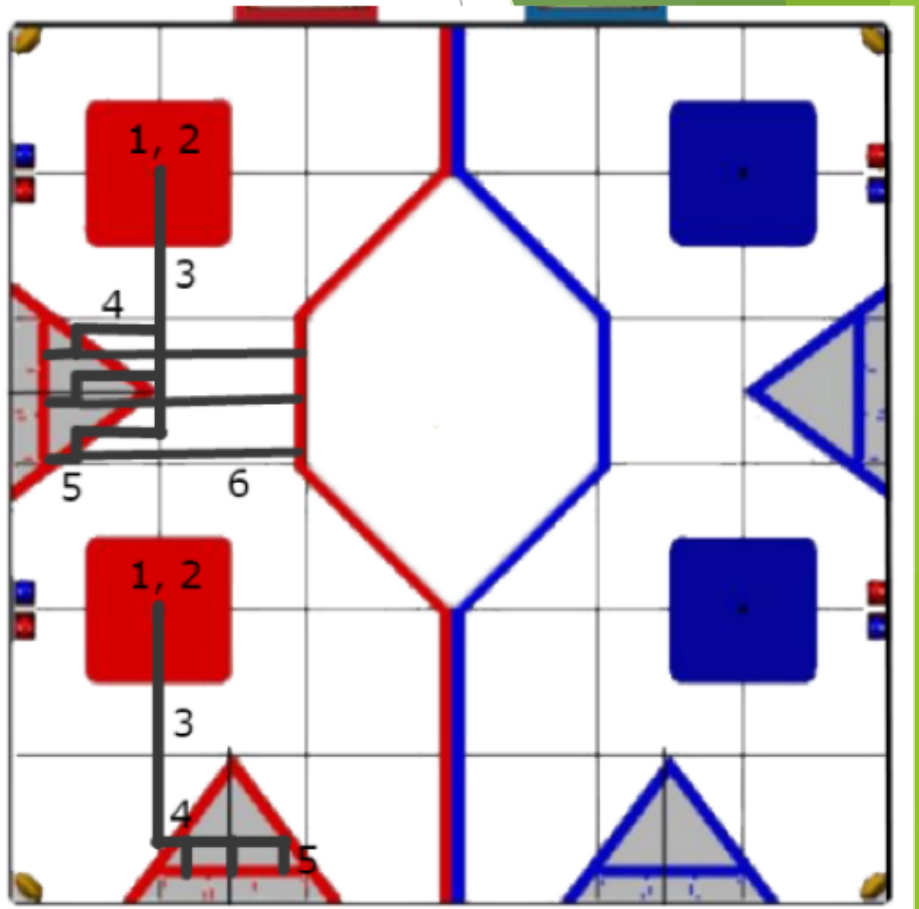
This is useful because:

- Quickly get the servo position values you need
  - Ex. Values for a stick hitting a ball
  - You can even do this before hardware for a module is build (such as initial, up, down, etc)
- Efficiently test autonomous functions
  - Test separate functions before integration
    - Ex. autonomous from last season

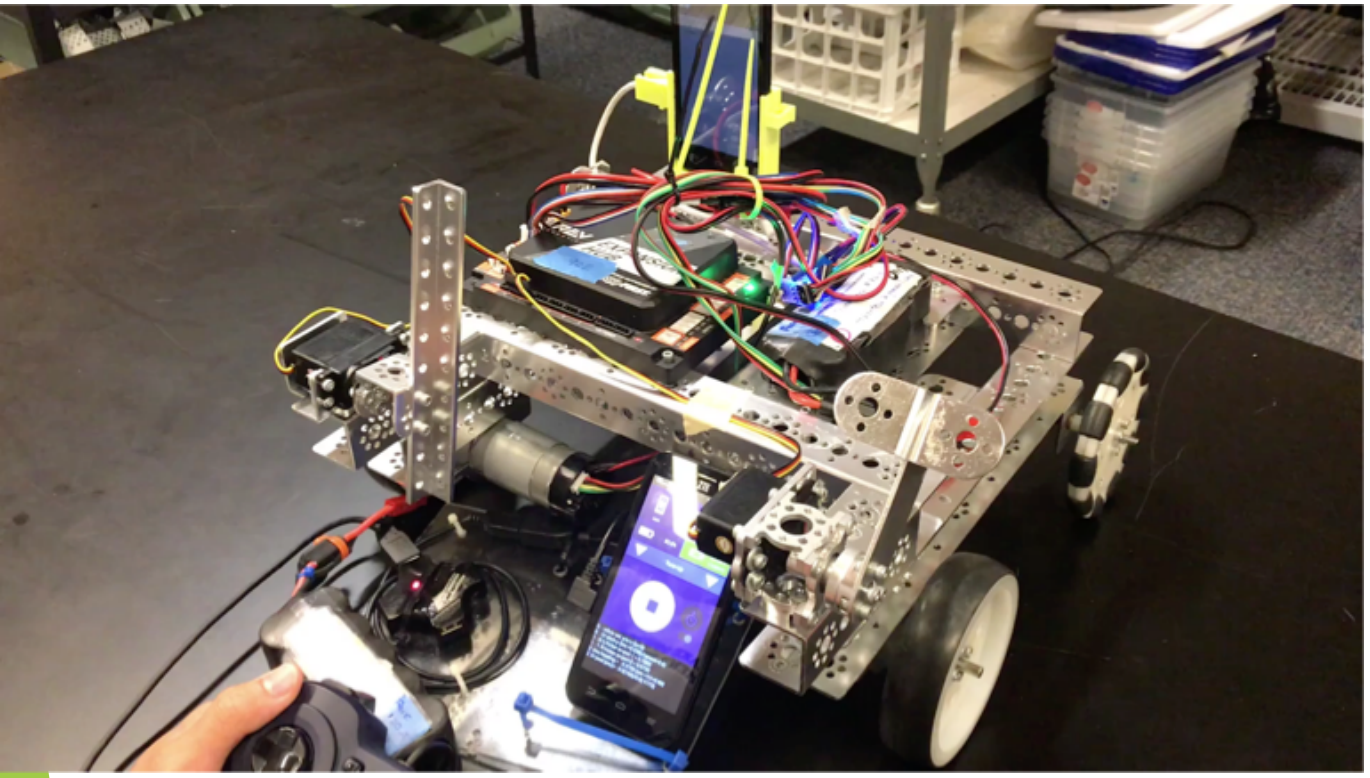
Autonomous function testing example

Partition a sequence into separate functions:

1. **Read the ball colors and knock off the right ball**
2. **Drive off the platform**
3. **Turn 90 degree toward the cryptobox**
4. **Drive close to the cryptobox and crab in parallel to the column.**
5. **Deliver the block and turn around to face the center block pile**
6. **Drive to the block pile, intake two blocks, and return to the cryptobox to deliver the blocks.**



Tune-Up Program - how does it work?



Alter servo values
Toggle b/t servos
Control servo “speed”
Test autonomous functions
Change speed of auto fxns
Next few slides will explain key elements of code found in tune up that allow use to perform the actions above

Servo Array

This is where you define all the servos you need to test

```
Servo[] sv_list = {
        robot.sv_kicker,
        robot.sv_elbow,
        robot.sv_shoulder
       };
```

Allows you to toggle between and calibrate multiple servos in this one program.
Provides flexibility to add/delete any servos efficiently.

Increment Variable and Controlling servo adjustment speed

```
static double INCREMENT = 0.001;


   if (gamepad1.left_stick_y<-0.1) {
        INCREMENT += 0.001;
        if (INCREMENT>0.5) INCREMENT = 0.5;
    } else if (gamepad1.left_stick_y>0.1) {
        INCREMENT -= 0.001;
        if (INCREMENT<0.001) INCREMENT = 0.001;
    }
```

The higher the increment value,

the faster the servo will move when you press y or a.

Speedscale variable and controlling autonomous function speed

```
double speedscale = 0.5;


if (gamepad1.right_stick_y<-0.1) {
   speedscale += 0.01;
   if (speedscale>1.0) speedscale = 1.0;
} else if (gamepad1.right_stick_y>0.1) {
   speedscale -= 0.01;
   if (speedscale<0.1) speedscale = 0.1;
}
```

The higher the speedscale value, the faster the robot will move/perform the autonomous function(s).

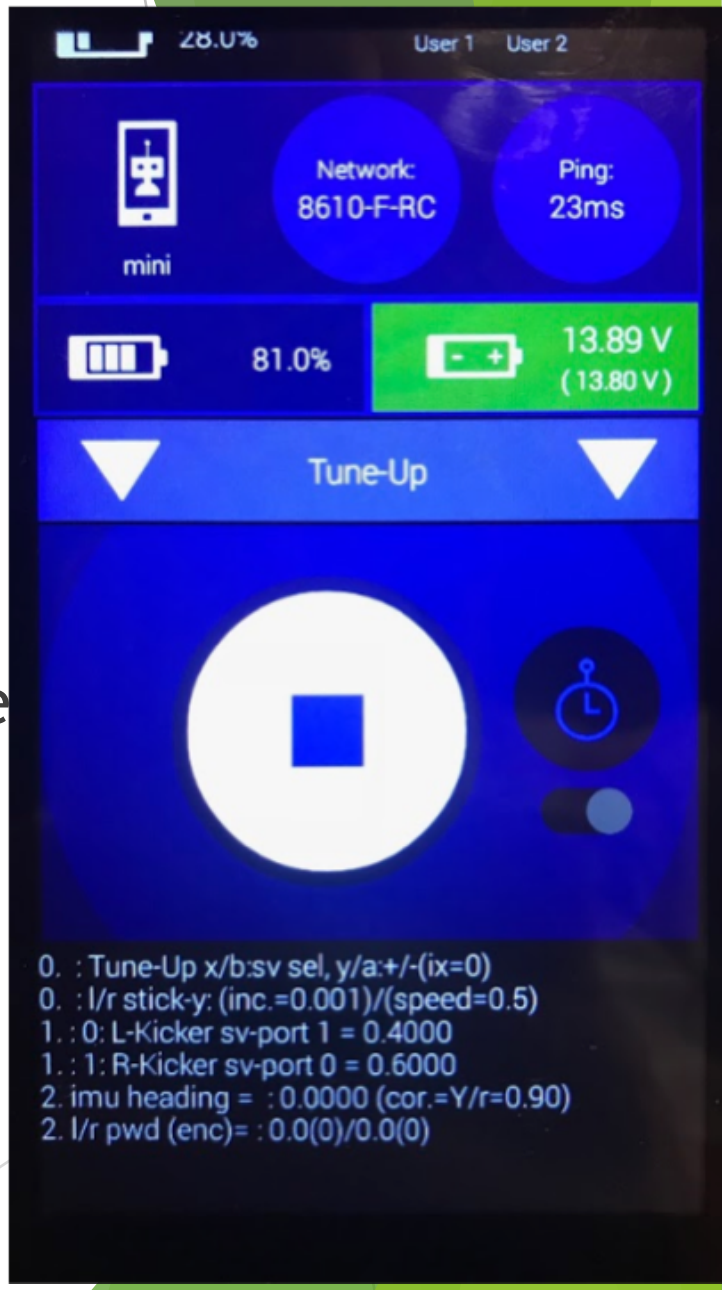
Telemetry

```
telemetry.addData("0. ", "x/b:sv sel, y/a:+/-(ix=%d)", cur_sv_ix);

telemetry.addData("0. ", "l/r stick-y: (inc.=%4.3f)/(speed=%2.1f)",
INCREMENT,speedscale);

telemetry.update();
```

Telemetry is text displayed on the driver station phone
We use telemetry to display info such as the current increment, speedscale, and **servo values**



Live Demo of Tune-Up Program

We will:

- Find servo positions needed to kick a ball
- Toggle between servos
- Move servos
- Speed up/Slow down servo movement

Conclusion

All the functions and elements of code that we have discussed today make up a rough outline of Tune-Up; there is much more to the program that makes it a very powerful calibration tool
You can download our Tune-Up program here:
https://github.com/tobortechftc/min-bot
Feel free to explore the program and tweak it as you like

Any questions?

Any other questions?

Contact us here!

email: tobortech@gmail.com
twitter: @tobortech8610

Follow us here!

Instagram: @tobortech.ftc
Facebook: @ToborTech
Snap: @tobortech.ftc
YouTube: Tobor Tech