BUZZ XX Software

An explanation of the code behind our 2015 FRC robot

Language: Labview

by: James Pillot IV

FRC Team 33 The Killer Bees

Software Mentors: Eric Yahrmatter, Jim Zondag

Special thanks to all the mentors who gave me great insight, assistance, and intuition in creating this project.

The Architecture

In 2013, the Killer Bees customized the standard Labview architecture provided by the WPI library to improve efficiency. Based on this work, we continue to develop our software using these customizations. **Everything exists inside of a timed execution structure that executes once per cycle at 100 HZ (a delta in time per cycle of 10 milliseconds) except for the values from the joysticks and the reporting of the CPU loads that run in parallel loops that execute at different rates.**

Init:

The init.vi is the opening VI of the program. This is treated like the Begin.VI in the standard template where motors, joysticks, sensors, and Bee-Script are initialized.

Input Proc:

The input proc VI processes all the raw data we need and turns it into data we can use. This could be something like taking the voltage output of potentiometers and turning it into standard measure units (inches, degrees, etc.). We also map all of our joystick inputs into their respective controls (where we say button B on the Xbox Controller makes the arm go up).

//

HMI/Auton:

The HMI and Auton exist inside of a case structure so that HMI.vi executes when the driver station reports that we are in tele-op. The Auton.vi executes when the driver station reports that we are in auton. If anything else is going on, the robot remains idle and the disabled signal is passed through to the other VI's to inform the robot to do nothing. *Regardless of the competition state, all other VI's in the main loop still execute once per cycle until the robot is cut off (init, input proc, control, output proc, dashboard).*

//

HMI:

This is the human machine interface. In this VI we read the processed inputs from the joysticks and decide what each subsystem needs to do based off of the input of the driver/operator. So if the driver says go forward and the operator commands the arm to go down, then those commands will be registered and fed to the Control VI for further action. We also have code to choose our autons in this VI.

Auton:

Our autonomous VI's go in here. The commands from bee-script are processed and fed through to the VI's underneath that act like the HMI VI and pass them through to the Control VI. If the script commands the arm to go up, then the Auton VI will treat that command like it is the operator telling the arm to go up and push it through to the Control VI to achieve the desired result.

Control:

The Control VI handles all of the advanced control work that is ultimately the logic of the system. This is mostly state machines and feedback loops.

Output Proc:

The Ouput Processing VI takes all the motor and solenoid outputs from the Control.VI and sends them to the motors and solenoids.

Dashboard:

The Dashboard VI writes all of the important data from the current cycle and pushes it to the dashboard so the drive team is aware of everything that they need to know about the robot (i.e. confirmation from a digital sensor that the robot is possessing a game piece).

The RECYCLE RUSH Buzz XX Code in Detail

A breakdown of the code from this year explained in the same pattern as the architecture.

A Physical Explanation of the Robot:

Buzz XX played RECYCLE RUSH, the 2015 FRC Game. Starting from the base and moving up, the robot had: a 4 wheel, all omni, west coast drive; collector wheels that could move in and out like clapping hands via pistons; a single stage elevator to pick up totes; a 4 bar and a claw at the end to pick up cans; a stack clamp that could lock its position to allow us to score, and can burglars that were deployed through the use of the stack clamp piston and reeled in through the use of a wench.

Sensor Breakdown by subsystem:

Drive: 2 Encoders

Intake: 2 Sic Sensors (2 sonar sensors that acted like a photogate and closed the transverse wheels when tripped)

Elevator: 1 potentiometer, 2 vex bump (contact sensors - let elevator know when a tote was collected)

4 Bar (Arm): 1 potentiometer

Stack Clamp: 1 potentiometer (reported how many totes there were)

Claw: 1 potentiometer, 1 sic sensor (let operator know if the can was firmly in claw or not)

Intro: Upon opening the project and looking at the Main.vi block diagram, one should notice the clusters titled HMI to Control, SensorInputData, Gamepad Data, Actuator Inputs. The HMI to Control cluster contains all of the variables (data types - boolean, float, int, string, etc.) I need to store data from the HMI.vi and feed it to the Control.vi. This included the states the arm could be, the states the stacker could be, the drive values transferred over from the arcade drive vi in HMI, and the boolean output of special requests (unlock the stack clamp, etc.). The SensorInputData holds all the processed data from Input Proc such as the stacker height, the arm angle, number of totes, etc.). The GamePad data holds all of the processed gamepad data from Input Procs (is the open claw button pressed? if so then report true so that the HMI can tell Control that the claw state is open and that the claw needs to open). The Acuator Inputs cluster holds all the variables that will be used to tell the motors what value to drive at and the pistons if they should be firing or not.

Init:

In this VI all the motors, sensors, and joysticks are initialized and the strings are fed into arrays instead of the “set” and “get” vi’s so that we can look up the devices manually and take stress off of the CPU. Bee-script is also initialized here and the file names are parsed for each auton that is run. We had a 20 pt auto (we used 3totewithramp at worlds and after because we thought we would have a ramp, but we did not actually use a ramp, we added can grabbers instead), a can stealing auto, a do nothing auto, and a auto that would pick up a can from the staging zone. The scripts can be found in Auton and then Routines if you open your file manager. They are .BEE which can be opened as .txt files.

Input Processing:

In this VI, we process all of our gamepad data and sensor inputs to units that we can actually use. The gamepad data is processed and all of that data writes the variables inside the GamepadData cluster. The Sensor inputs write the variables inside the SensorInputData cluster. The Units Please VI takes the inputs of a max and minimum unit measurement and its corresponding sensor voltage to return an output of a desired unit for every input of the incoming sensor voltage. This is used to know the 4 bar position in degrees, the stacker height in inches, and the stack clamp position in degrees. The Angle to Tote Number VI takes the stack clamp angle and compares it to a series of ranges to report a number of how many totes the machine has. Each tote would make the bar move higher and increase the angle of the stack clamp so that the range for 3 totes could be 162-180 degrees but the range for 4 totes could be 180-200 degrees. The claw was the only subsystem left in raw units instead of an actual standard unit because we never got around to it, so yeah. The compressor is also turned on and off as needed through this VI.

HMI:

In our teleop code section we take the inputs from the Gamepad Data cluster that came from input procs (essentially which button has been pressed and how much input from the joysticks we have received) and use it to write the variables in our HMI to Control cluster. We controlled our 4 bar and claw with pre-set positions, so instead of using manual control on the subsystems (even though it was made available) we had a button for an open, closed, and slightly open claw position, and different buttons for different 4 bar heights. These pre-set positions are initially defined as states (those are the enums) and sent to the Control VI where more is done with them

later. We also ran the joystick inputs through the math necessary for an arcade drive style of control (left joystick (y-axis) controls forward-backward input, right joystick (x-axis) controls left-right input) and mapped the button presses on the d-pad to cycle through autons. We also have extra amenities such as the ability to shut the stacker motor off and on and manually deploy the can burglars in telop if needed. However, if you look closely you may notice that there is no button mapping to any heights for the elevator other than the manual override, the command to score, the command to exit the scoring the position, and the command to put the stack as low to the ground as possible for driving around with a lower center of gravity if necessary (never used). That is because our elevator was completely automated this year. To build a stack of 6 the operator would simply hold the collect button and let the bump sensor register that the tote was there and the stacker would do the rest. The Ramp PWM VI is complementary to the Drive VI and controls the rate of input from the joysticks per cycle. This makes it so that even if the driver slams the joysticks forward, the robot won't respond so violently but instead, gradually progress to the desired velocity at a previously specified rate.

Control:

Intake Wheels VI : In this VI we take input from the operator and the sensors on the intake to determine if the intake should be open or closed. It is a simple state machine that first considers if the operator has requested manual control or automation. If in manual control, the intake opens and closes based off of user input. If in automation, the intake opens and closes based off of sensor input (is a tote there or not).

Stack Clamp/Can Grab VI: Takes the requests from HMI and turns them into a state for the state machine to work with.

Stacker State Machine VI: The VI responsible for the automation of the elevator, it looks at data from nearly every subsystem and commands all of them on what to do (except the drivetrain and 4 bar) based on how many totes we have, what the operator wants to do, and what we just did. For example if we had 3 totes and we were about to collect our fourth tote, then the claw would be commanded to slightly open and then close once the elevator reached its setpoint (this was necessary to push the can up because our elevator could not power through the tight grip our claw put on the can). The main goal of the VI - determine the desired elevator position (output as a state), intake state (open or closed), claw action, and stack clamp state without any user input to stack totes as fast as possible.

Drive to Outputs: Take right and left drive outputs from arcade drive and send them to the motors.

Arm:

Arm Setpoint Index: Look at the arm state and pick a position to send to PID loop.

Arm Reset I? : Tell PID if the integral term needs to be reset.

Arm Gain Index: Pick which gains to send to PID.

Arm PID: PID loop for the 4 bar.

Arm Manual Override: Overrides PID output with output from the joystick and makes output 0 if the position is exceeding the safety limits.

Stacker:

Follows same pattern as Arm VI

Claw:

Follows same pattern as Arm VI except for state machine. The state machine in this VI makes the claw compress the can slightly and then open up a little bit once the can is compressed to flip over sideways cans.

Output to Intake:

Takes the desired states of the subsystems with pistons and makes them true or false to feed to the solenoid.

Dashboard:

Writes relevant data to dashboard.

Output Processing:

Writes the motors and solenoids to make the robot perform desired actions.