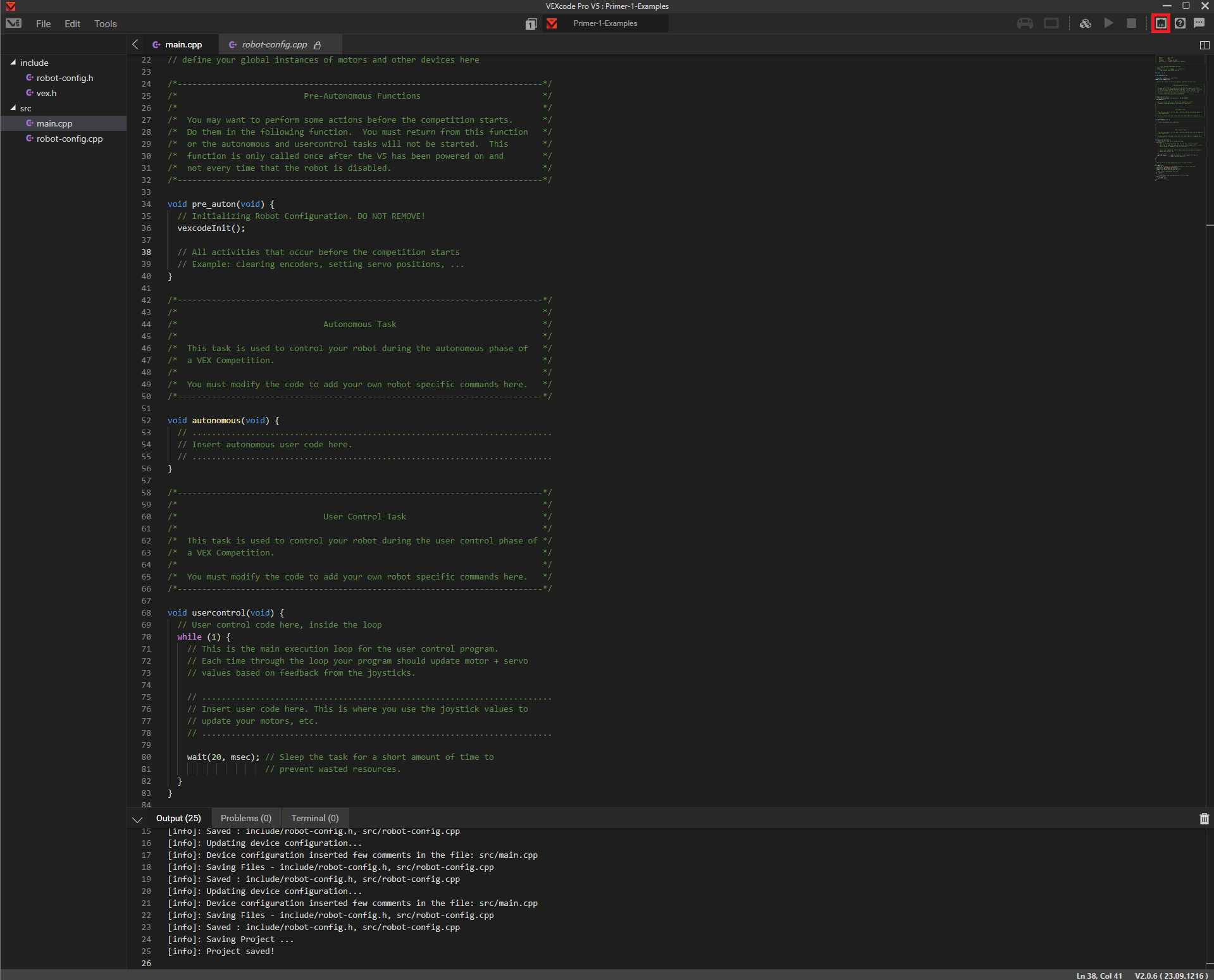
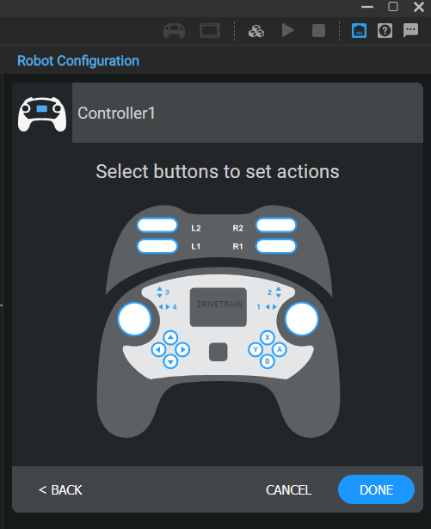
Basic Primer of VEXCODE: C++

Setting Up General Configuration

To first set up the motors, drivetrain and other objects, you must click the button highlighted in red. It will permit you to assign ports to the various motors and sensors connected to your brain. After clicking on it should show a tab with a plus sign and “Add a device”.

A screenshot of a computer

AI-generated content may be incorrect.Click on the said button and then click controller leading to this screen: Now click done and select add a device again but select drivetrain this time with the correct quantity of motors, if it is greater than 4, reference Appendix A: Exception 1. From here select the left motor ports starting with 1, then going in order of free odd ports (This is a convention, not a requirement). Once you have selected the left motors, proceed to selecting the right motor ports, just with even numbers this time. In the case that you have an inertial sensor, click inertial sensor and select that port. This then takes you to the following configuration page which is shown below. If you are using blue motor cartridges, then select the 6:1 ratio. If you are using green motor cartridges then do not change your choice, and if you are using red motor cartridges select 36:1. Then you select wheel size, and input gear ratio via inputting the starting gear, for example a 36 tooth, then the secondary gear, for example an 84-tooth gear. Additionally, if the drivetrain is completely reversed, click the down arrow centered on the drivetrain. Finally, click done at the bottom left of the configuration window. You may now reopen the Controller tab and click on the joysticks until the correct configuration appears. The diagram does a rather good job of explaining the possibilities, so I will leave that to it.

Next, to add motors you will select add a device yet again and then select, motor, a port, then a cartridge as previously referenced. Repeat this motor creation process unless you have a set of motors that you wish to have work as one. In that case select the option: motor group. For all sensors, simply follow these steps; Click add a device, select the addon to the bot, click the port, and look at the final config page for additional instructions such as swapping direction. Finally for three-wire devices complete the first step for sensors but select three-wire; then select the device you are planning to use and assign a letter port (the slot ports found on one of the shorter sides of the brain. For pneumatics controllers, use a digital out.

User control

A screen shot of a computer program

AI-generated content may be incorrect.To begin user control programming, you must first navigate to the competition template ~ line 70, the line that states “void usercontrol(void) {“. From here go to the next line and here you can write the lines of code that you only want to operate once at the start of usercontrol such as velocity settings for motors or the drivetrain. Below this, inside of the while loop, you will put pieces of code you wish to run constantly, such as buttons to control parts of your robot. You would do this via if statements as shown below. Don’t be concerned about the specifics of what is inside of the if clauses, just focus on what it entails. This screenshot details an if clause that checks for various states of controller buttons to check what to do with a motor called Arm. An if clause starts with the word if, then a state to check such as a controller button being pressed or not. The brackets afterwards are what will be executed if that state is true. If you wish a state check to only happen if something else has not happened, add an else if below the brackets of the if clause. This will only be checked if the above if clause is not true. A reason that you may wish to control a motor through a single if grouping is that you likely don’t want to have a motor being sent both forward and reverse commands at the same time (the last command given is given priority). The else if clause will act similar to an if clause. Just with the additional stipulation shown about. Finally, the else clause states that if neither the if nor possibly multiple else ifs are false, then this will happen. This is best for stopping a motor if neither command buttons are being pressed as shown above.

Command Functions

There are a variety of command functions using the inbuilt classes of the vex library. These classes are used to create motors, so far, the program has created them for you, but you still need to know how to use them. Below are all the useful functions with descriptions and an example line.

Drivetrain:

These commands can also be used with a smartdrive. A drivetrain is what the drivetrain that you created earlier is you didn’t add an inertial sensor.

Drivetrain.drive(enum);

Causes the drivetrain to drive in a given direction for as long as it isn’t interrupted. The acceptable inputs (things to put where enum currently is) are forward, fwd, and reverse.

Drivetrain.driveFor(direction, distance, distanceUnit);

Causes the drivetrain to drive in a given direction for a certain distance. It works by first inputting direction, such as forward or reverse; then number such as 10 where distance currently is, and a distance unit where distanceUnit is such as inches or millimeters(mm). It also has additional functions detailed in Appendix: Details 1 & 2.

Drivetrain.turn(direction);

Causes the drivetrain to turn in a certain direction until stopped. That direction can be right or left. It also has additional functions detailed in Appendix: Details 1.

Drivetrain.turnFor(direction, amount, rotationUnit);

Causes the drivetrain to turn in a direction, by an amount in the rotational unit of your choosing. Direction can be Forward, Reverse, or fwd. Amount can be a number with decimals or no decimals, and rotationUnit can be degrees or rev(revolutions). It also has additional functions detailed in Appendix: Details 1.

Drivetrain.stop(brakeType);

This command causes the drivetrain to stop with a certain type. In the brakeType slot, you can put coast which just stops power to the drivetrain, brake which stops the motors, then has it coast, and finally hold which just stops the motors and keeps them stopped.

Drivetrain.isSpinning();

Drivetrain.isDone();

These commands will output either true or false based on if a Drivetrain motor is spinning or the Drivetrain is still attempting to do something.

Drivetrain.setDriveVelocity(amount, velocityUnits);

Drivetrain.setTurnVelocity(amount, velocityUnits);

These two commands modify drive and turn speeds respectively. You may want to do this due to autonomous inconsistency at higher speeds. The amount can be any number, but in effect, you cannot tell a motor to go faster than it standardly can. The velocity units can either by percent (0-100 for amount) and rpm (0-600 depending on the motors used).

Drivetrain.setStopping(brakeType);

Sets the default brake type for the brake command so you don’t need to specify what type. In the brakeType slot, you can put coast which just stops power to the drivetrain, brake which stops the motor, then has it coast, and finally hold which just stops the motor and keeps it stopped.

There are more functions for the Drivetrain class, but they are unnecessary at the current moment.

Smartdrive:

A smartdrive is a drivetrain with an attached inertial sensor which gives it the following additional commands otherwise you can refer to the commands above.

Drivetrain.turnToHeading(angle, degrees\*);

This causes the drivetrain to turn to a definite heading, so even if there is an obstacle it will continue turning until the inertial sensor thinks it has turned enough. Angle is formatted in the same way as nautical bearing if north was the direction the robot started in.

\*Can use radians, but I doubt that those reading this will need to use them.

Motor:

Pretty Self explatory

Motor.spin(direction);

This command causes the motor to spin the direction listed. Direction can either be forward(fwd) or reverse.

Motor.setVelocity(integer, units);

Sets the speed of which the motor will spin at. Units can be rpm or percentage of max rpm.

Motor.spinFor(direction, amount, units);

Causes the motor to go in the direction give for amount units. Direction can either be forward(fwd) or reverse. Amount can be any number, and units can be degrees or rotations.

Motor.stop(brakeType)l

Causes the motor to stop with the braking type given. The brakeTypes are coast which just stops power to the drivetrain, brake which stops the motor, then has it coast, and finally hold which just stops the motor and keeps it stopped.

Motor.setVelocity(amount units);

Sets motor velocity. WARNING ALL VELOCITIES ARE AUTOMATICALLY SET TO 50% AT THE STARTOF AUTONOMOUS. The amount can be any number, but in effect, you cannot tell a motor to go faster than it standardly can. The velocity units can either by percent (0-100 for amount) and rpm (0-600 depending on the motors used).

Motor.setMaxTorque(amount, units);

Same logic as above, but just set it to 100%. I debated not including it because it’s the preset and why would you weaken your motor?

Motor.setStopping(brakeType);

Same function as the drivetrain variant. The brakeTypes are coast which just stops power to the drivetrain, brake which stops the motor, then has it coast, and finally hold which just stops the motor and keeps it stopped.

Motor\_group:

Same exact commands as above, just applying to all motors at the same time in a coordinated manner.

Digital\_out/Pneumatics:

For use with pneumatics and 3 wire controllers(Ports A,B,etc)

Digital\_out.set(Boolean);

Sets whether the air is being let through or not. Forget if true or false lets it through. Probably true.

Boolean can either be true or false, letting air through or not.

Autonomous

Apologies, had to use vscode for this section, so it will be slightly less comparative to your experience.

A computer code with colorful text

AI-generated content may be incorrect.This is an example of autonomous code. You put it inside of the brackets that are after the “void autonomous(void)” Claw is a motor and drivetrain has an inertial sensor(is a smartdrive). Honestly, it is the simplest to write, but most frustrating in practice. Simply use the deterministic(driveFor, turnFor, spinFor, etc) function/commands to tell the robot what to do to accomplish the objective. Attempt Autonomous Win Point if possible, as it gives quite the advantage, and best of luck. Also, ALL VELOCITIES ARE AUTOMATICALLY SET TO 50% AT THE STARTOF AUTONOMOUS.

Final Remarks:

There might be another Primer for a template that I may create for public use in the future. If you have an issue, you can contact me at [s024033@students.lmsd.org](mailto:s024033@students.lmsd.org). The profile picture will either be of a snow leopard or of some amalgamation of technology.

Appendix:

To those with a 6 motor drive train, insert the following code into your file outside of any function, preferably before pre\_auton.

A screen shot of a computer

AI-generated content may be incorrect. You can change the names of the motorgroups and the drivetrain, but these are what I use. Preferably don’t configure the drivetrain the normal way and this, and you will likely have to create an inertial sensor, but it’s relatively intuitive, so you should be fine. Apologies for the iffy grammar, and best of luck.