Robotics Code Explanation

By Konrad Kraemer

**Table Of Contents:**

Table Of Contents 1

Explanation of main.c 2

Feature 1: Hardware abstraction 3

Feature 2: Relative Omnidirectional Drive 4

Feature 3: Autonomous Recording 5

Recording 5

Playback 7

Other Code Files (For Reference) 9

**NOTE:** This is code is split up to highlight three major features. The Rest of the code is included at the bottom

**“main.c”**  (excluding the pragma which sets up the motors and sensors)

1. #include "Vars.h" // file for global variables
2. #include "Writer.h" // file which has code for recording inputs and outputting autonomous code
3. #include "calculateMotors.h"
4. #include "Robot.h" // comment out when testing with virtual worlds
5. #include "RC.h" // comment out when testing with virtual worlds
6. //#include "RobotVirtual.h" // when testing with virtual worlds
7. //#include "RCVirtual.h" // when testing with virtual worlds
8. #include "autoControl.h" // contains autonomous controlling function
9. #include "autoCode.h" // contains generated autonomous code
10. // rotation sensor = 627.2 ticks per rotation.
11. **task main()** **{**
12. resetVars**();** // reset all variables
13. resetSensors**();** // reset all sensors
14. AutoL**();**
15. **while** **(true)** **{**
16. RC**();** // writes joystick values into variables
17. calcMotorValues**();** // writes motor power based on RC’s input
18. writeStream**();** // comment out for actual run
19. RunRobot**();** // writes motor powers and reads sensor values
20. **}**
21. **}**

main() is very simple. It resets all variables and sensors to 0, runs an autonomous, and then loops the remote control code infinitely. There are 2 possible starting locations, and we have an autonomous for each: Left (AutoL), and Right (AutoR).

**Feature 1: Hardware abstraction**

**“Vars.h”**

1. //drivetrain vars
2. int leftBackDrivePower**;**
3. int rightBackDrivePower**;**
4. int leftFrontDrivePower**;**
5. int rightFrontDrivePower**;**
6. //actuator vars
7. int armPower**;**
8. //Remote Vars
9. int rcDriveForward**;**
10. int rcDriveStrafe**;**
11. int rcDriveTurn**;**
12. //Encoder Vars
13. int leftFrontEncoder**;**
14. int rightFrontEncoder**;**
15. int leftBackEncoder**;**
16. int rightBackEncoder**;**
17. int leftArmDegrees**;**
18. int rightArmDegrees**;**
19. //Other Sensor Vars
20. float gyroValue**;**
21. int lineReadMid**;**
22. int lineReadSide**;**
23. //==================
24. //constants
25. //==================
26. const float EFL **=** 1**;**
27. const float EFR **=** 1**;**
28. const float EBL **=** 1**;**
29. const float EBR **=** 1**;**
30. const int POTENTIOMETER\_DIFF **=** 284**;**
31. const float CORRECTION **=** .5**;**
32. const int AUTO\_TIME\_INTERVAL **=** 5**;** // time between execution of autonomous reads (in 10 millisecond increments (5 = 50milliseconds))
33. const int MAINTAIN\_HEIGHT\_ARM\_POWER **=** 10**;**
34. const int DEADZONE **=** 10**;**
35. const int MAX\_WAIT\_TIME **=** 500**;**

We rely on heavy testing for my code. To facilitate testing outside of school, we use a software package called *Virtual Worlds*, which allows you to program and drive a virtual robot. This, however has two problems. You must drive a default robot, and you must use another type of joysick. To solve this problem, we have relied on hardware abstraction. All four driver control functions (lines 43-46) read from or write to variables contained in “Vars.h”. Only RunRobot() interfaces with the actual sensors and motors. Likewise, only RC() reads the joystick values. This has multiple advantages. Firstly, when we are testing with virtual worlds, we only need to use another version of RC() and RunRobot(). The rest of the code can be used as is. Secondly, this allows us to quickly support new robot designs by only changing a few lines in a single location. Lastly, this lets us assume that the encoder variables will be positive when the robot is moving in a certain direction. If we notice otherwise, then we only have to change the code in a single spot, instead of scavenging through and adding negative signs everywhere.

**Feature 2: Relative Omnidirectional Drive**

**“calculateMotors.h”**

1. void calcMotorValues**()** **{**
2. int actualFWD **=** rcDriveForward **\*** cosDegrees**(**gyroValue**)**
3. **+** rcDriveStrafe **\*** cosDegrees**(**90 **+** gyroValue**);**
4. int actualLFT **=** rcDriveForward **\*** sinDegrees**(**gyroValue**)**
5. **+** rcDriveStrafe **\*** sinDegrees**(**90 **+** gyroValue**);**
6. **if** **(**abs**(**actualLFT**)** **<=** DEADZONE**)** //absolute value <= 10 (gives preference to Forward)
7. actualLFT **=** 0**;**
8. /\*
9. Fwd = Positive
10. Left = Positive
11. \*/
12. leftFrontDrivePower **=** actualFWD **+** actualLFT **+** rcDriveTurn**;**
13. leftBackDrivePower **=** actualFWD **-** actualLFT **+** rcDriveTurn**;**
14. rightFrontDrivePower **=** actualFWD **-** actualLFT **-** rcDriveTurn**;**
15. rightBackDrivePower **=** actualFWD **+** actualLFT **-** rcDriveTurn**;**
16. **}**

This function calculates the power which will be applied to each motor. This came about because of a simple problem: when the robot is facing towards the driver, the controls are reversed. Forward is towards the driver, backwards is away, left is to the driver’s right, and right is to the driver’s left. Our team’s drivers found this very annoying, so we came up with this solution. What if “forward” always was away from the driver, and what if “left” was always to his left? This is what this function seeks to implement

**Feature 3: Autonomous Recording**

This section focuses on the unique way how the autonomous is made. In the past, we wrote it by hand using trial and error. This was a very tedious process, so we was thinking about ways how I could speed the process up. A team member saw a documentary on how cars were made, and half way through the video, it showed one of the technicians programming one of the robots. The technician was moving the robot to certain locations, and the robot would then go to these points autonomously. Our new version of autonomous implements this idea: it writes itself while the robot is driven.

**“Writer.h”**

1. void writeStream**()** **{**
2. //----------------------
3. **if** **(**leftBackDrivePower **<** **-**127 **+** DEADZONE**)**
4. leftBackDrivePower **=** **-**127 **+** DEADZONE**;**
5. **else** **if** **(**leftBackDrivePower **>** 127 **-** DEADZONE**)**
6. leftBackDrivePower **=** 127 **-** DEADZONE**;**
7. **if** **(**rightBackDrivePower **<** **-**127 **+** DEADZONE**)**
8. rightBackDrivePower **=** **-**127 **+** DEADZONE**;**
9. **else** **if** **(**rightBackDrivePower **>** 127 **-** DEADZONE**)**
10. rightBackDrivePower **=** 127 **-** DEADZONE**;**
11. **if** **(**leftFrontDrivePower **<** **-**127 **+** DEADZONE**)**
12. leftFrontDrivePower **=** **-**127 **+** DEADZONE**;**
13. **else** **if** **(**leftFrontDrivePower **>** 127 **-** DEADZONE**)**
14. leftFrontDrivePower **=** 127 **-** DEADZONE**;**
15. **if** **(**rightFrontDrivePower **<** **-**127 **+** DEADZONE**)**
16. rightFrontDrivePower **=** **-**127 **+** DEADZONE**;**
17. **else** **if** **(**rightFrontDrivePower **>** 127 **-** DEADZONE**)**
18. rightFrontDrivePower **=** 127 **-** DEADZONE**;**
19. **if** **(**armPower **<** **-**127 **+** DEADZONE**)**
20. armPower **=** **-**127 **+** DEADZONE**;**
21. **else** **if** **(**armPower **>** 127 **-** DEADZONE**)**
22. armPower **=** 127 **-** DEADZONE**;**
24. // if there is something happening, write autonomous code
25. **if** **(**abs**(**leftBackDrivePower**)** **>** 10 **||** abs**(**rightBackDrivePower**)** **>** 10
26. **||** abs**(**leftFrontDrivePower**)** **>** 10 **||** abs**(**rightFrontDrivePower**)** **>** 10
27. **||** abs**(**armPower**)** **>** 10 **||** abs**(**beltPower**)** **>** 10
28. **||** abs**(**leftFrontEncoder**)** **>** 10 **||** abs**(**rightFrontEncoder**)** **>** 10
29. **||** abs**(**leftBackEncoder**)** **>** 10 **||** abs**(**rightBackEncoder**)** **>** 10
30. **||** abs**(**leftArmDegrees**)** **>** 10 **||** abs**(**rightArmDegrees**)** **>** 10**)** **{**
31. writeDebugStream**(**"Auton(%d, "**,** **(**leftFrontDrivePower**));**
32. writeDebugStream**(**"%d, "**,** **(**leftBackDrivePower**));**
33. writeDebugStream**(**"%d, "**,** **(**rightFrontDrivePower**));**
34. writeDebugStream**(**"%d, "**,** **(**rightBackDrivePower**));**
35. writeDebugStream**(**"%d, "**,** **(**armPower**));**
36. writeDebugStream**(**"%d, "**,** **(**beltPower**));**
37. writeDebugStream**(**"%d, "**,** **(**leftFrontEncoder**));**
38. writeDebugStream**(**"%d, "**,** **(**leftBackEncoder**));**
39. writeDebugStream**(**"%d, "**,** **(**rightFrontEncoder**));**
40. writeDebugStream**(**"%d, "**,** **(**rightBackEncoder**));**
41. writeDebugStream**(**"%d, "**,** **(**leftArmDegrees**));**
42. writeDebugStreamLine**(**"%d);"**,** **(**rightArmDegrees**));**
43. writeDebugStreamLine**(**"RunRobot();"**);**
44. writeDebugStreamLine**(**"wait10Msec(%d);"**,** **(**AUTO\_TIME\_INTERVAL**));**
45. **}**
46. wait10Msec**(**AUTO\_TIME\_INTERVAL**);**
47. **}**

Lines 5 to 28 check each motor value to make sure it is between -117 and positive 117. This is because the maximum power is 127, and since there is always a margin of error, the deadzone of 10 provides enough room to correct the motor speed to make sure the robot is driving as close as possible to what was recorded.

Next, the if statement on lines 31-36 makes sure that something has happened. This is so that the robot won’t start spitting out code until it starts moving.

Lastly, writeDebugStream() and writeDebugStreamLine() outputs text into the debug stream, which is similar to Java’s System.out.print() and System.out.printLn().

1. Auton**(-**98**,** 117**,** 117**,** **-**98**,** 1**,** 0**,** 749**,** 762**,** 723**,** 688**,** 0**,** 0**);**
2. RunRobot**();**
3. wait10Msec**(**5**);**

This is an example output.

The wait10Msec(5) is highly important, because the remote control has a very low bandwidth, so if it were to send messages any more frequently, some lines would be missing or only partially present.

**“autoControl.h”**

1. void Auton**(**int leftFrontWheel**,** int leftBackWheel**,**
2. int rightFrontWheel**,** int rightBackWheel**,**
3. int armSpeed**,** int beltSpeed**,**
4. int leftFrontEncoderGoal**,** int leftBackEncoderGoal**,**
5. int rightFrontEncoderGoal**,** int rightBackEncoderGoal**,**
6. int leftPotentiometerGoal**,** int rightPotentiometerGoal**)** **{**
7. // guarantee that the loop doesn't run if no motors with encoders are running.
8. int wait **=** 0**;**
9. **while** **(((**leftFrontEncoderGoal **-** leftFrontEncoder **>=** 0 **&&** leftFrontDrivePower **>** 0**)**
10. **||** **(**leftFrontEncoderGoal **-** leftFrontEncoder **<=** 0 **&&** leftFrontDrivePower **<** 0**)**
11. **||** **(**leftFrontDrivePower **==** 0**))**
12. **&&** **((**leftBackEncoderGoal **-** leftBackEncoder **>=** 0 **&&** leftBackDrivePower **>** 0 **)**
13. **||** **(**leftBackEncoderGoal **-** leftBackEncoder **<=** 0 **&&** leftBackDrivePower **<** 0 **)**
14. **||** **(**leftBackDrivePower **==** 0**))**
15. **&&** **((**rightFrontEncoderGoal **-** rightFrontEncoder **>=** 0 **&&** rightFrontDrivePower **>** 0 **)**
16. **||** **(**rightFrontEncoderGoal **-** rightFrontEncoder **<=** 0 **&&** rightFrontDrivePower **<** 0 **)**
17. **||** **(**rightFrontDrivePower **==** 0**))**
18. **&&** **((**rightBackEncoderGoal **-** rightBackEncoder **>=** 0 **&&** rightBackDrivePower **>** 0 **)**
19. **||** **(**rightBackEncoderGoal **-** rightBackEncoder **<=** 0 **&&** rightBackDrivePower **<** 0 **)**
20. **||** **(**rightBackDrivePower **==** 0**))**
21. **&&** **((**leftPotentiometerGoal**-**leftArmDegrees **>** 0 **&&** armPower **>** 0**)**
22. **||** **(**leftPotentiometerGoal**-**leftArmDegrees **>** 0 **&&** armPower **>** 0**)**
23. **||** **(**armPower **==** 0**))**
24. **&&** **((**rightPotentiometerGoal**-**rightArmDegrees **>** 0 **&&** armPower **>** 0**)**
25. **||** **(**rightPotentiometerGoal**-**rightArmDegrees **>** 0 **&&** armPower **>** 0**)**
26. **||** **(**armPower **==** 0**))**
27. **&&** **(**abs**(**leftFrontDrivePower**)** **>** 10 **||** abs**(**leftBackDrivePower**)** **>** 10
28. **||** abs**(**rightFrontDrivePower**)** **>** 10 **||** abs**(**rightBackDrivePower**)** **>** 10
29. **||** armPower **!=** 0 **||** beltPower **!=** 0**)**
30. //or while you haven't waited longer than the max wait time
31. **&&** **!**wait **>** MAX\_WAIT\_TIME**)** **{**
32. //END WHILE CONDITION
33. RunRobot**();** // update variables
34. wait**++;** // increase the wait tick
35. **}**
36. **if** **(**MAX\_WAIT\_TIME **<=** wait**)** // if loop was ended by waiting
37. writeDebugStreamLine**(**"I gave up"**);** // notify via debug stream
38. //applies values to motors
39. leftFrontDrivePower **=** leftFrontWheel**;**
40. leftBackDrivePower **=** leftBackWheel**;**
41. rightFrontDrivePower **=** rightFrontWheel**;**
42. rightBackDrivePower **=** rightBackWheel**;**
43. armPower **=** armSpeed**;**
44. beltPower **=** beltSpeed**;**
45. //This section makes minor modifications to motor speeds if they overshot or undershot their goal.
46. int diff **=** leftFrontEncoderGoal **-** leftFrontEncoder**;**
47. leftFrontDrivePower **+=** diff **\*** CORRECTION**;**
48. diff **=** leftBackEncoderGoal **-** leftBackEncoder**;**
49. leftBackDrivePower **+=** diff **\*** CORRECTION**;**
50. diff **=** rightFrontEncoderGoal **-** rightFrontEncoder**;**
51. rightFrontDrivePower **+=** diff **\*** CORRECTION**;**
52. diff **=** rightBackEncoderGoal **-** rightBackEncoder**;**
53. rightBackDrivePower **+=** diff **\*** CORRECTION**;**
54. diff **=** leftPotentiometerGoal **-** leftArmDegrees**;**
55. armPower **+=** diff **\*** CORRECTION**;**
56. //NOTE: right arm not necessary because that correction is done in Robot.h
57. **}**

This is the function which replicates the conditions that were present upon recording.

The first while loop updates the sensor variables (calls RunRobot() ) until any encoder has reached its “goal”. The goal is the value the encoder had at that moment during the recording run. Alternatively, the loop is aborted after a certain amount of iterations. This was added because the robot has to score cubes on posts which are next to a wall. If the robot reaches the wall earlier than expected, it can’t reach its goal value, and the loop wouldn’t terminate otherwise.

Next comes the section where the motor power variables are set to the same value they had at that instant during the recording run. Due to a variety of reasons, the motors will not, however go equally far every time, and so each value gets minutely adjusted to correct for whether it overshot or undershot.

**Other Code Files (For Reference)**

“RC.h”

1. void RC**(){**
2. //Drivetrain
3. rcDriveForward **=** vexRT**[**Ch3**];** //left thumbstick up down
4. rcDriveStrafe **=** vexRT**[**Ch4**];** //left thumbstick left right
5. rcDriveTurn **=** vexRT**[**Ch1**];** //right thumbstick left right
6. //Scissor Lift
7. armPower **=** vexRT**[**Ch2/\*Xmtr2\*/**];** //right thumbstick up down on second controller
8. **if** **(**armPower **>=** 0 **&&** armPower **<** MAINTAIN\_HEIGHT\_ARM\_POWER**)**
9. armPower **=** MAINTAIN\_HEIGHT\_ARM\_POWER**;**
10. **}**

“Robot.h”

1. void RunRobot**(){**
2. //read sensor values and put them into the variables
3. leftFrontEncoder **=** nMotorEncoder**[**leftFront**];**
4. rightFrontEncoder **=** nMotorEncoder**[**rightFront**];**
5. leftBackEncoder **=** nMotorEncoder**[**leftBack**];**
6. rightBackEncoder **=** nMotorEncoder**[**rightBack**];**
7. leftArmDegrees **=** SensorValue**[**leftScissorRot**]** **-** POTENTIOMETER\_DIFF**;**
8. rightArmDegrees **=** SensorValue**[**rightScissorRot**];**
9. beltSensorVal **=** nMotorEncoder**[**frontClaw**];**
10. gyroValue **=** SensorValue**[**gyro**]/**10**;** // because 27 degrees is "270"
11. motor**[**leftFront**]** **=** leftFrontDrivePower **\*** EFL**;**
12. motor**[**leftBack**]** **=** leftBackDrivePower **\*** EBL**;**
13. motor**[**rightFront**]** **=** rightFrontDrivePower **\*** EFR**;**
14. motor**[**rightBack**]** **=** rightBackDrivePower **\*** EBR**;**
15. motor**[**leftScissor1**]** **=** armPower**;**
16. motor**[**leftScissor2**]** **=** armPower**;**
17. motor**[**rightScissor1**]** **=** armPower**;**
18. motor**[**rightScissor2**]** **=** armPower**;**
19. int diff **=** rightArmDegrees **-** leftArmDegrees**;**
20. **if** **(**armPower **>** 0**)** **{**
21. **if** **(**leftArmDegrees **>** rightArmDegrees**)** **{**
22. motor**[**leftScissor1**]** **+=** diff**;**
23. motor**[**leftScissor2**]** **+=** diff**;**
24. **}**
25. **else** /\*if (leftArmDegrees < rightArmDegrees)\*/ **{**
26. motor**[**rightScissor1**]** **-=** diff**;**
27. motor**[**rightScissor2**]** **-=** diff**;**
28. **}**
29. **}** **else** /\*if (scissorPower < 0)\*/ **{**
30. **if** **(**leftArmDegrees **>** rightArmDegrees**)** **{**
31. motor**[**rightScissor1**]** **-=** diff**;**
32. motor**[**rightScissor2**]** **-=** diff**;**
33. **}**
34. **else** /\*if (leftArmDegrees < rightArmDegrees)\*/ **{**
35. motor**[**leftScissor1**]** **+=** diff**;**
36. motor**[**leftScissor2**]** **+=** diff**;**
37. **}**
38. **}**
39. motor**[**frontClaw**]** **=** beltPower**;**
40. **}**