February 13, 2015

Jerry Fan

Mareep

K-FORCE

Impulse Programming notebook

Contents

[Version 1 – Time V1 2](#_Toc411619636)

[Version 2 – Time V2 3](#_Toc411619637)

[Autonomous 4](#_Toc411619638)

[User Control 5](#_Toc411619639)

[Autonomous Strategy 6](#_Toc411619640)

[Version 3 – Time V3 7](#_Toc411619641)

[Version 4 – Time V4 8](#_Toc411619642)

[Version 5 – Time V5 10](#_Toc411619643)

[Version 6 – Impulse V1 11](#_Toc411619644)

# Version 1 – Time V1

Date: 07/06/2014

Compiler: RobotC

Sensors Setup:

#pragma config(Motor, port1, LDB, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port2, RDB, tmotorVex393, openLoop)

#pragma config(Motor, port3, LL1, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port4, LL2, tmotorVex393, openLoop)

#pragma config(Motor, port5, RL1, tmotorVex393, openLoop)

#pragma config(Motor, port6, RL2, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port9, RDF, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port10, LDF, tmotorVex393, openLoop)

The first robot we had was quite a simple one without any real sensors, which made autonomous programming hard and imprecise. The build was finished very close to the scrimmage date, so I did not have chance to work on an autonomous routine, given that the user control code was more important to implement first.

User Control:

task usercontrol()

{

int rightLift;

int leftLift;

int leftDF;

int rightDF;

int leftDB;

int rightDB;

while (true)

{

//Lift

//Get Controller Values

leftLift=(vexRT[Btn5U]-vexRT[Btn5D])\*127;

rightLift=(vexRT[Btn5U]-vexRT[Btn5D])\*127;

//Give Lift Values

motor[LL1] = leftLift+10;

motor[LL2] = leftLift+10;

motor[RL1] = rightLift+10;

motor[RL2] = rightLift+10;

//Drive

//Left Stick Control (Strafe and movement)

leftDF=vexRT[Ch3]+vexRT[Ch4];

leftDB=vexRT[Ch3]-vexRT[Ch4];

rightDF=vexRT[Ch3]-vexRT[Ch4];

rightDB=vexRT[Ch3]+vexRT[Ch4];

//Right Stick Control (Turning)

leftDF=leftDF+vexRT[Ch1];

leftDB=leftDB+vexRT[Ch1];

rightDF=rightDF-vexRT[Ch1];

rightDB=rightDB-vexRT[Ch1];

//Give motors values

motor[LDF]=leftDF;

motor[LDB]=leftDB;

motor[RDF]=rightDF;

motor[RDB]=rightDB;

}

}

This was what I put together quickly to make the first robot competition worthy. Integers are declared early on which act as a ‘buffer’ to hold values sent by the controller before they are assigned to the motors

# Version 2 – Time V2

Date: 05/07/2014

Compiler: RobotC

Sensors Setup:

#pragma config(I2C\_Usage, I2C1, i2cSensors)

#pragma config(Sensor, in1, LL\_pot, sensorPotentiometer)

#pragma config(Sensor, in2, RL\_pot, sensorPotentiometer)

#pragma config(Sensor, I2C\_1, RDB\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_2, RDF\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_3, LDF\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_4, LDB\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Motor, port1, RDF, tmotorVex393, openLoop, encoder, encoderPort, I2C\_2, 1000)

#pragma config(Motor, port2, RDB, tmotorVex393, openLoop, encoder, encoderPort, I2C\_1, 1000)

#pragma config(Motor, port3, LLU, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port4, LLD, tmotorVex393, openLoop)

#pragma config(Motor, port5, RLU, tmotorVex393, openLoop)

#pragma config(Motor, port6, RLD, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port7, RIN, tmotorVex393, openLoop)

#pragma config(Motor, port8, LIN, tmotorVex393, openLoop)

#pragma config(Motor, port9, LDB, tmotorVex393, openLoop, reversed, encoder, encoderPort, I2C\_4, 1000)

#pragma config(Motor, port10, LDF, tmotorVex393, openLoop, encoder, encoderPort, I2C\_3, 1000)

After the first scrimmage, many more sensors were added onto the cortex. This included two potentiometers and the addition of integrated encoders on the motors. On the programming side, this meant that they needed to be added to the config. I focused on getting the autonomous working properly for this version of the program, as it came to the stage where most robots would be built and would have autonomous for the next scrimmage.

I also added experimental functions to this version of the code. Experimental functions are ideas that I deem to be worth trying out, but have unstable or unverified code. The two I have in this program is the PID control for robot drive and also the logarithmic drive code to increase controller sensitivity at lower values at the cost of sensitivity at higher values. I believe the log drive will help driving, since at high speeds it’s unlikely that a driver would be concentrating on precision, so the loss of sensitivity will not matter. The more precise control could assist in moving small distances, such as when trying to place a skyrise into the slot. The PID however, I could not get to function properly so it was not activated in this program. Both these experimental functions have a switch, which is a constant Boolean variable that can be true or false, either allowing it to run or disabling it respectively.

## Autonomous

The first thing that I worked on for autonomous was code that would let the robot know how far it travels using encoders, and stop itself at a specified value for autonomous.

void drive\_move(int encoder\_distance,int speed)

{

clear\_previous\_error();

clear\_encoders();

bool LDB\_goal\_reached = false;

bool RDB\_goal\_reached = false;

bool LDF\_goal\_reached = false;

bool RDF\_goal\_reached = false;

//Data validation

speed = abs(speed);

while ((LDB\_goal\_reached == false)||(RDB\_goal\_reached == false)||(LDF\_goal\_reached == false)||(RDF\_goal\_reached == false))

{

if (abs(nMotorEncoder[LDB]) < abs(encoder\_distance))

{

motor[LDB] = speedstep\_drive(encoder\_distance,speed,nMotorEncoder[LDB],3);

}

else

{

motor[LDB] = 0;

LDB\_goal\_reached=true;

}

if (abs(nMotorEncoder[RDB]) < abs(encoder\_distance))

{

motor[RDB] = speedstep\_drive(encoder\_distance,speed,nMotorEncoder[RDB],0);

}

else

{

motor[RDB] = 0;

RDB\_goal\_reached=true;

}

if (abs(nMotorEncoder[LDF]) < abs(encoder\_distance))

{

motor[LDF] = speedstep\_drive(encoder\_distance,speed,nMotorEncoder[LDF],2);

}

else

{

motor[LDF] = 0;

LDF\_goal\_reached=true;

}

if (abs(nMotorEncoder[RDF]) < abs(encoder\_distance))

{

motor[RDF] = speedstep\_drive(encoder\_distance,speed,nMotorEncoder[RDF],1);

}

else

{

motor[RDF] = 0;

RDF\_goal\_reached=true;

}

}

}

The main feature in autonomous is a while loop that checks each motor’s encoder individually to see if they have reached the target distance as set when the code is written. If it has, then it only switches off that individual motor, whilst the rest will keep going until they reach their respective targets. I find this is more accurate, especially with turning, which uses a variation of this code with two motors reversed.

The lift code autonomous function works in a simplified manner, as it was decided to only use one potentiometer due to them both having different scaling values. The value of the left and right potentiometers did not match, and the rate of change of those values were different as well. Basically the lift function brought the lift up or down towards a certain potentiometer value. It stopped the lift motors when the target is reached.

The intake was easy to program, as it only involved activating the two intake motors by setting them to a value, and then back to 0 when I wanted them to stop.

## User Control

The user control section of this program was harder to program due to the fact that values on the controller needs to be constantly updated. This means that the main usercontrol task’s while loop cannot be clogged up with anything.

As previously, to assign values to motors, the program checks for the controller values and then stores them in a variable, before the motor is set to equal that variable. Trigger (5U&D, 6U&D) values are multiplied by 127, which corresponds to the maximum motor speed or -127, for the minimum speed. Left triggers control lift, right triggers control intake.

Because the lift kept falling down when a load was applied, a trim was added that constantly kept the lift motors running at 30, to prevent it from sliding back down when not being controlled. However, there was another problem with rubber bands not letting the lift be at its lowest location. To solve this problem, I thought of a trim that would switch to keeping the lift down when the lift was below a certain height. I found this ‘threshold’ using RobotC and the potentiometers on the lift. As a result, the lift did stay at the height it was intended to be at.

Something new I added on to version two of the program was a preset lift height system. When a button is pressed, the lift would automatically move towards a set height, by checking the current potentiometer value, deciding whether to go up or down, and moving there. These would be activated by controller buttons, with potentially 8 presets. However, only 6 were programmed in, with two being reserved for a possible skyrise intake.

The skyrise intake however did pose a major problem. Moving the lift with it on meant that the side that it was attached on tilted due to its weight. The motors applied a certain amount of force, but because of the uneven weight distribution, it would be unable to lift the side with the claw as fast as the side without, creating this tilt. In trying to solve this, the side that the claw was not attached on had its lift motors underpowered, to compensate the slowness of the other side of the lift. In going down, the reverse was true for the lift. The side without the claw was going down faster, and so is underpowered. To detect which side the claw was on, a potentiometer was added to the robot so that we could manually indicate which side the claw was attached, so that the program would know which motors to underpower when lifting. When the potentiometer was disconnected, I found it had a value of 240-250, and that is the range I set for the indication of no skyrise intake.

## Autonomous Strategy

Because the skyrise claw that was designed was hard to use in autonomous, it was decided that autonomous would consist of cube scoring only. Two different autonomous strategies were devised, so four total routines were written in code, two for each side. One was a floor goal scorer, dragging one cube onto the mat. The other attempted to score on a post. It was decided to use the floor goal scoring as the default as it was reliable.

|  |  |
| --- | --- |
| C:\Users\Aeon\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Skyrise 920p.png  **1**  Figure 1 Outer floor autonomous | C:\Users\Aeon\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Skyrise 920p.png  **1**  Figure 2 Inner floor goal autonomous |
| C:\Users\Aeon\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Skyrise 920p.png  **1**  **2**  Figure 3 Outer post scorer | C:\Users\Aeon\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Skyrise 920p.png  **2**  **1**  Figure 4 Inner post scorer |

On the red side, both of these would be reversed as the field is mirrored. Essentially the inner scorer becomes the outer and vice versa.

# Version 3 – Time V3

Date: 02/08/2014

Compiler: RobotC

Sensors Setup:

#pragma config(I2C\_Usage, I2C1, i2cSensors)

#pragma config(Sensor, in1, LL\_pot, sensorPotentiometer)

#pragma config(Sensor, in2, RL\_pot, sensorPotentiometer)

#pragma config(Sensor, in3, One, sensorPotentiometer)

#pragma config(Sensor, I2C\_1, RDB\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_2, RDF\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_3, LDF\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_4, LDB\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Motor, port1, RDF, tmotorVex393, openLoop, encoder, encoderPort, I2C\_2, 1000)

#pragma config(Motor, port2, RDB, tmotorVex393, openLoop, encoder, encoderPort, I2C\_1, 1000)

#pragma config(Motor, port3, LLU, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port4, LLD, tmotorVex393, openLoop)

#pragma config(Motor, port5, RLU, tmotorVex393, openLoop)

#pragma config(Motor, port6, RLD, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port7, RIN, tmotorVex393, openLoop)

#pragma config(Motor, port8, LIN, tmotorVex393, openLoop)

#pragma config(Motor, port9, LDB, tmotorVex393, openLoop, reversed, encoder, encoderPort, I2C\_4, 1000)

#pragma config(Motor, port10, LDF, stmotorVex393, openLoop, encoder, encoderPort, I2C\_3, 1000)

This program was a cleaned up version of version 2. I got another programmer – Kristian to look through my code as he is already familiar with java, a similar language to C. I am introducing him to robotics so I thought this would be a good exercise. However due to the shortage of time, this program was never implemented as it was untested when the next scrimmage came.

A new structure was implemented by Kristian that used switches instead of if based switches.

void presetAssign()

{

switch (liftPreset)

{

case 2:

if (SensorValue[LL\_pot] < liftVal[1])

{

liftAutonVal[0] = liftVal[1];

liftPresetMonitor();

}

else

{

liftPreset = 0;

}

break;

case 1:

if (SensorValue[LL\_pot] > liftVal[0])

{

liftAutonVal[0] = liftVal[0];

liftPresetMonitor();

}

else

{

liftPreset = 0;

}

break;

……………

And so on.

# Version 4 – Time V4

Date: 06/08/2014

Sensors Setup:

#pragma config(I2C\_Usage, I2C1, i2cSensors)

#pragma config(Sensor, in1, PotentiometerLL, sensorPotentiometer)

#pragma config(Sensor, in2, PotentiometerRL, sensorPotentiometer)

#pragma config(Sensor, in3, , sensorGyro)

#pragma config(Sensor, in4, LineL, sensorLineFollower)

#pragma config(Sensor, in5, LineM, sensorLineFollower)

#pragma config(Sensor, in6, LineR, sensorLineFollower)

#pragma config(Sensor, dgtl1, UltrasonicL, sensorSONAR\_mm)

#pragma config(Sensor, dgtl3, UltrasonicR, sensorSONAR\_mm)

#pragma config(Sensor, dgtl12, PistonClaw, sensorDigitalOut)

#pragma config(Sensor, dgtl6, EIN, sensorQuadEncoder)

#pragma config(Sensor, I2C\_1, RDB\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_2, RDF\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_3, LDF\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_4, LDB\_encoder, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Motor, port1, RDF, tmotorVex393, openLoop, encoder, encoderPort, I2C\_2, 1000)

#pragma config(Motor, port2, RDB, tmotorVex393, openLoop, encoder, encoderPort, I2C\_1, 1000)

#pragma config(Motor, port3, LLU, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port4, LLD, tmotorVex393, openLoop)

#pragma config(Motor, port5, RLU, tmotorVex393, openLoop)

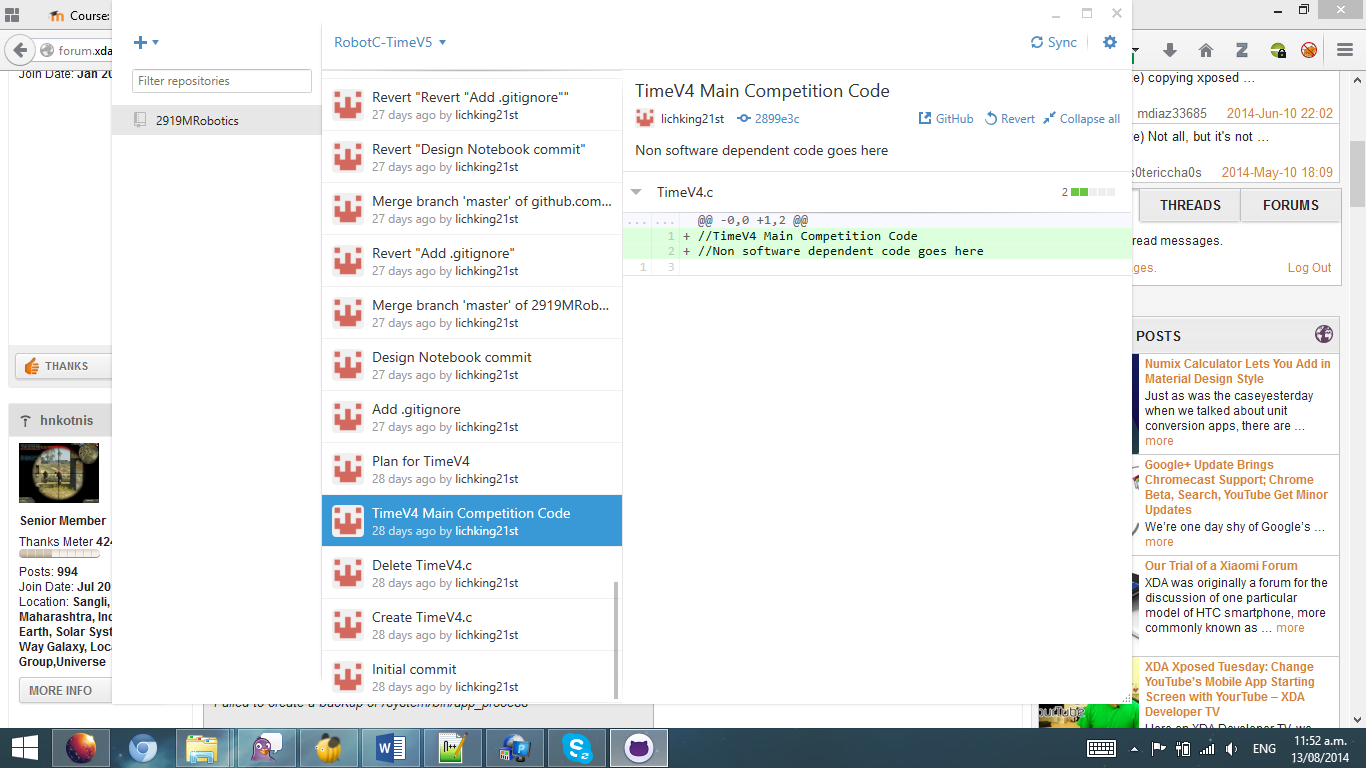
#pragma config(Motor, port6, RLD, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port7, RIN, tmotorVex393, openLoop)

#pragma config(Motor, port8, LIN, tmotorVex393, openLoop)

#pragma config(Motor, port9, LDB, tmotorVex393, openLoop, reversed, encoder, encoderPort, I2C\_4, 1000)

#pragma config(Motor, port10, LDF, tmotorVex393, openLoop, encoder, encoderPort, I2C\_3, 1000)



Version four of the program was a huge step forward in terms of program structure. I started to use header files and includes to clear up clutter on the main file. The main RobotC file is now a list of imports. The advantage I have in doing this is that I can make changes to individual parts of the program such as usercontrol, without having to look through the whole entire program.

I also added this program to a github repository that I set up, this means that other VEX programmers are able to see my work if they search for it, and also so that the other programmer on my team can easily contribute towards parts of the program. Git uses a revision feature, keeping track of changes in code files. This means with each update, I have to submit what is called a ‘commit’, allowing me to record the development of the program, which helps in the design notebook.

The directory structure of TimeV4 goes as follows

* TimeV4.c – Main file
  + Main.h – main header file for code
  + Definitions.h – contains custom macros and definitions
  + Preauton.c – contains preautonomous tasks
  + Usercontrol.c – contains user control tasks
  + Autonomous.c – contains autonomous tasks

The features I implemented was no different to Time V2, however I implemented the features in differing ways. Within this program, I used tasks more than one control loop, especially with autonomous where I have dedicated tasks for setting motor speeds and controlling the lift. I had to manage the different tasks with wait commands so that one task did not take up all the CPU time and essentially jam up the program.

Within the sensor declarations, I added many placeholder sensors for anticipated sensors that would be attached to the robot. These include the three line sensors and the two ultrasonic ones.

This program received less testing than V2 however, and as a result the final results were worse than the previously built program, even though the structure of the code in this program is significantly better.

# Version 5 – Time V5

Date:

Sensors Setup:

#pragma config(I2C\_Usage, I2C1, i2cSensors)

#pragma config(Sensor, in1, potLL, sensorPotentiometer)

#pragma config(Sensor, in2, potRL, sensorPotentiometer)

#pragma config(Sensor, in3, turningGyro, sensorGyro)

#pragma config(Sensor, in4, lineL, sensorLineFollower)

#pragma config(Sensor, in5, lineM, sensorLineFollower)

#pragma config(Sensor, in6, lineR, sensorLineFollower)

#pragma config(Sensor, dgtl1, ultraL, sensorSONAR\_mm)

#pragma config(Sensor, dgtl3, ultraR, sensorSONAR\_mm)

#pragma config(Sensor, dgtl12, piston, sensorDigitalOut)

#pragma config(Sensor, dgtl6, encoderIN, sensorQuadEncoder)

#pragma config(Sensor, I2C\_1, encoderRDB, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_2, encoderRDF, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_3, encoderLDF, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Sensor, I2C\_4, encoderLDB, sensorQuadEncoderOnI2CPort, , AutoAssign)

#pragma config(Motor, port1, driveRF, tmotorVex393, openLoop, encoder, encoderPort, I2C\_2, 1000)

#pragma config(Motor, port2, driveRB, tmotorVex393, openLoop, encoder, encoderPort, I2C\_1, 1000)

#pragma config(Motor, port3, liftLU, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port4, liftLD, tmotorVex393, openLoop)

#pragma config(Motor, port5, liftRU, tmotorVex393, openLoop)

#pragma config(Motor, port6, liftRD, tmotorVex393, openLoop, reversed)

#pragma config(Motor, port7, intakeR, tmotorVex393, openLoop)

#pragma config(Motor, port8, intakeL, tmotorVex393, openLoop)

#pragma config(Motor, port9, driveLB, tmotorVex393, openLoop, reversed, encoder, encoderPort, I2C\_4, 1000)

#pragma config(Motor, port10, driveLF, tmotorVex393, openLoop, encoder, encoderPort, I2C\_3, 1000)

Given that Time V3 and V4 was not actually used in any scrimmages due to instability, I wanted to make a program that was stable, and that was the aim for V5. V5’s current user control disables diagonal movement due to a fault we found in the last scrimmage where motors would fail after moving diagonally for an extended amount of time. If the controller is using both the x and y axis on the left stick, then it takes the largest of the two, and uses that instead, resulting in either strafe or straight movement.

This program was built on V4, after I spent one day looking through and trying to eliminate all the bugs I could see. As such it still retains the task structure of multiple tasks controlling separate parts of the robot, rather than a single task running everything. The user control is mainly working, but presets are not for some reason. If they are not easily fixable, I will revert back to the old V2 style of presets that used the main user control task to activate them instead of the newer structure.

# Version 6 – Impulse V1

Date: 24/07/14

During the short session on this day, I worked on the autonomous part of the robot, making sure all the required functions were in place in V5 of the program. The focus was mainly on driving, with the robot now able to go back and forth with macros to call the required functions in the code quickly.

Date: 28/07/14

Attempted to put in an independent task that could hook on to other tasks during the autonomous section of matches. This task would have the ability to stop the robot if it detects a jam, so if the motor does not move, it would stop powering it to keep it from being damaged. I’m not sure how well this will work however, given that the motor itself has no sensors, so that I’m relying on things like potentiometers and encoders to give me accurate information.

Date: 29/07/14

Maximum speed of the robot was reduced on request of our driver Marco. The intakes have now been programmed on and mapped on to the controller, so that’s working well so far. I also touched up on the antijam function as it wasn’t interacting with the other tasks, hopefully it should work now. A trim switch has been added so that the robot won’t loosen up when it is in compact form, by making sure the lift moves down constantly to oppose the tension of the rubber bands. However the trim will switch to a proper one, holding the lift up when it goes above a certain point. Skyrise intake which is separate from the main intake has also been programmed and mapped.

Date: 6/08/14

Massive code revamp done on the current program version, a lot of bug fixes with the code was done, but the robot is not with me so no testing can be done yet. Theoretically the new code should run better and smarter than before.

Date: 12/08/14

Disabled movement in diagonal directions. Robot is able to move diagonally due to the mek wheels installed but will hang for some reason. Our driver does not need this function so I have disabled it, the robot now moves only forwards and back and left to right for strafing.

Date: 19/08/14

Day of autonomous tests, trying to score points in autonomous whilst the test field is set up. Skyrise auton has been abandoned as it is too hard with the robot to position it correctly. Scoring with cubes however is easier so that will be done.

Date: 30/09/14

New gyro has arrived so I have started to work out how to implement this in my code. I have got the sensor connected and working so far, I just need to tweak the sensitivity so that one turn on robot registers as one turn on the sensor.

Date: 8/10/14

Gyro has been completely integrated into the robot, and now is accurate. It provides a precise indication of where the robot is pointing relative to the starting position of the robot, which is pretty useful compared to the encoders that are less precise.

**Dev Entries from Kristian Hansen:**

15/10/14

I have started the program for the second robot. Currently, it only contains a user control function to make the robot move, but more will definitely be added as time goes on. I have chosen the same file structure as it was in TimeV5 and TimeV6, but the code itself will be completely different and not based off said versions.

15/10/14

I have now completed the autonomous macros needed to move the robot automatically by simply calling a preset function which does a specific thing like move forwards or backwards, or turn left or right. I have used the same strategy in TimeV6, which is defining a method which moves the robot, and adding in #definitions which call that method. Take a look at the example below

#define drive(x, y) (move(x, abs(y), abs(y), abs(y), abs(y)))

#define driveBack(x, y) (move(x, -abs(y), -abs(y), -abs(y), -abs(y)))

#define turnLeft(x, y) (move(x, -abs(y), -abs(y), abs(y), abs(y))

#define turnRight(x, y) (move(x, abs(y), abs(y), -abs(y), -abs(y)))

16/10/14

There is now an arcade mode within the program, which means that you can switch driving control layouts with the push of a button. I also have programmed a cooldown timer, so that you have time to take your finger off the button before the button value is read again.

23/10/14

I have implemented the framework for the autonomous routine for the robot. Currently, the major functionality so far is a calibration method for calibrating the line sensors and finding the threshold between them for easy identification if the robot is currently not on the line.

28/10/14

I have programmed the basic autonomous routine (or so I hope) that will make the robot follow a line (hopefully). Currently, the code is not executed during the autonomous routine, but rather at the press of a button on the controller in usercontrol.c. The while loop is quite simple, here it is below for reference:

if (isValid())

{

drive(10, 127);

}

else

{

correctRobot();

}

21/11/14

Today I have added the controllers for the lift. The lift motors can now be controlled using the port 5 up and down buttons. Currently, all it assigns the speed of the motors to is 127 or -127. PID’s will be needed eventually because Marco informs me that the brick handles the motors at different speeds, so for precision lift control, these will be the next step.

Following updates generated through the use of

git log --reverse --no-merges --pretty="Date: %cd%nAuthor: %an%nSubject: %s%nComments:%n%b%n" > git.log

Git log is a very handy tool for keeping track of changes and writing comments about what changed within a piece of code.

Date: Fri Nov 28 02:25:25 2014 +1300

Author: Jerry Fan

Subject: Experimental code completed

Comments:

Will start testing soon

More task oriented user control allows more control over motor values at any given time

Date: Fri Nov 28 14:17:35 2014 +1300

Author: Kristian Hansen

Subject: Pragma & Lift changes

Comments:

Added a trim to the lift, and reversed the button controls for the lift.

Also altered the #pragma's a bit as well.

Date: Fri Nov 28 20:16:05 2014 +1300

Author: Jerry Fan

Subject: Bugfixes

Comments:

Fixed ramping code bringing speed above specified speed

Improved preset efficiency

Date: Fri Nov 28 20:32:06 2014 +1300

Author: Jerry Fan

Subject: Mirror main code development

Comments:

Date: Wed Dec 3 09:18:03 2014 +1300

Author: Kristian Hansen

Subject: Lift function

Comments:

Added a lift function for autonomous. And edited trim value.

Date: Mon Dec 8 17:51:38 2014 +1300

Author: Jerry Fan

Subject: Experimental code working

Comments:

Replaced old usercontrol with experimental

Date: Tue Dec 9 01:51:26 2014 +1300

Author: Jerry Fan

Subject: Update of code to include breaking and bits

Comments:

Date: Wed Dec 10 11:21:09 2014 +1300

Author: Kristian Hansen

Subject: Add extra lift motors

Comments:

Added #pragmas and stuff for the new lift motors.

Date: Wed Dec 10 19:03:33 2014 +1300

Author: Jerry Fan

Subject: Semicolons fixed, Improvements to functions

Comments:

Functions can be set default values, so I set some

Date: Wed Dec 10 20:07:49 2014 +1300

Author: Jerry Fan

Subject: Added commit logs

Comments:

Commits from the past can be generated using the git log function. This will allow us to generate a process journal on the go and whenever we need rather than filling out each entry on a word document.

Date: Thu Dec 11 01:53:00 2014 +1300

Author: Jerry Fan

Subject: fineControl has been fixed

Comments:

The function that let the controller be less sensitive at lower values as been fixed and implemented into the code. This means should the controller be moved up to a value of say 50, it will only power motor to around 20, this way there is more precise control. However should the controller be pushed up to its maximum of 127, it will still set motors to 127 so no speed is sacrificed. The problem itself lay within the floating point calculators of the C language, it somehow could not implicitly convert 127 to 127.0 when set on a float variable.

Date: Thu Dec 11 16:40:48 2014 +1300

Author: Marco Tyler-Rodrigue

Subject: Jerry Fan to the rescue!

Comments:

Fixed drive, added ultra sonics and speaker code.

Date: Tue Dec 16 00:30:59 2014 +1300

Author: Jerry Fan

Subject: Debugging Session

Comments:

I went to test the code I commited to Github lately. Ultrasonic sensors were replaced with working ones and autonomous code to go with it was testing. Testing was successful. The robot would continue to move until a condition was fufilled, in this case it was the ultrasonics. I also tested the encoder condition and it seemed to be working. The only thing giving me trouble now is the fine control, which apparently was not fixed.

Using the ringtone converter, I also managed to upload a few tunes and play them on the VEX speaker, so this might be interesting.

Date: Tue Dec 16 01:17:06 2014 +1300

Author: Jerry Fan

Subject: Emulation Debugging

Comments:

RobotC compiler is not giving me any debug values for some reason, but they are obviously there due to the fact that the robot still drives when I apply the fineControl. Thus I have concluded the problem is due to the float value being more than 0 but less than 1, which leads to a rounding of 1. I have set the outputs from -3 to 3 to return 0.

Date: Tue Jan 20 00:10:38 2015 +1300

Author: Jerry Fan

Subject: Workshop Build Session

Comments:

Today I managed to test several of the features last implemented in emulation debugging. It seems the lift doesn't quite reach the bottom of the preset I set, which is for minimum height. The fault was not in the potentiometer, and was eventually found when I removed the 'easing' code. This part will have to be redeveloped. It also seems that the exponential drive doesn't work with the ramping code so I have scrapped that completely now. It also seems the braking code for my autonomous drive only accounted for driving forward and not backwards, I am working on a fix for that.

Date: Tue Jan 20 02:10:44 2015 +1300

Author: Jerry Fan

Subject: Pre Workshop Commit

Comments:

This commit contains several key features to be used within the next workshop session. I've prebuilt code for the robot to go forth and back to simulate skyrise building. I have also included a light sensor conditional to my code so I will have that function available should I need it. The braking code also has been fixed.

Date: Wed Jan 21 01:55:29 2015 +1300

Author: Jerry Fan

Subject: Workshop Build Session

Comments:

Initilisation of robot has been tested, forwards and backwards auton has been tested and also improved on. The conditions and commands now tie into each other so there are no chances that one will miss the other and cause a slip in either conditions or commands.

Date: Wed Jan 28 10:31:09 2015 +1300

Author: Jerry Fan

Subject: Pre Workshop Commit

Comments:

Changes to pretty much every piece of code file was required as I updated the drive to not use seperate tasks anymore, as there were no loops being introduced into the drive system as there looks to be none in the future.

I fixed a problem with the autonomous not actually ending the step properly, so the next action did not stop. A new time condition has been added to add the option of waiting for a time condition to be met before commands such as move were stopped. The safeguards previously written to prevent conditions being conditions for the wrong steps were integrated into the functions themselves so I don't have to manually write them anymore.

Two new piston toggles added and remapped to the shoulder buttons on the controller.

Lifttrim downwards only triggers at the start of a match, after this there is no more downwards trim so that there's no slipping in driver control or autonomous.

Lift compensation has been added in case the lift tilts when the skyrise claw is added, this is not activated as of yet however.

Finally a bunch of new variables had to be declared to make these functions work, stored within the variables header.

Date: Wed Jan 28 10:35:23 2015 +1300

Author: Jerry Fan

Subject: Design Notebook

Comments:

Slight design notebook update.

Date: Thu Jan 29 01:26:32 2015 +1300

Author: Jerry Fan

Subject: Autonomous Overhaul

Comments:

Restored the old autonomous system with single conditional move functions. New system too hard to debug with RobotC and the constant hardware connection problems in VEX.

Added support for new buttons, one to control the needle deployment without need for the controller and another to calibrate the light sensor in different environments.

Autonomous currently should deploy the robot then drive forwards and backwards to fill in skyrise pieces. It should detect whether there is a skyrise or not in the holder with its light sensor and if so, grab it. The robot deployment has enough force to knock the preload on to the skyrise base, scoring additional points.

Date: Fri Jan 30 22:08:44 2015 +1300

Author: Kristian Hansen

Subject: Remove direct call to piston

Comments:

This is better in terms of safety and if you want to change things later

Date: Mon Feb 2 23:26:02 2015 +1300

Author: Jerry Fan

Subject: Fixes and autonUser addition

Comments:

Added ability for auton to function within userControl with second controller. Fixed the previous problem with the bumper buttons with inverted on/off values.

Date: Wed Feb 4 00:17:18 2015 +1300

Author: Jerry Fan

Subject: Workshop Auton

Comments:

The auton has been modified with tweaks to improve timing, the lift now will grab a skyrise as it lifts, which reduces the amount of up down cycles done by the lift.

Date: Fri Feb 6 00:39:17 2015 +1300

Author: Jerry Fan

Subject: Workshop Session

Comments:

Auton running into problems as older framework does not support multi threaded actions, commands and conditions are tied together so may need to rely on more tasks to get the job done.

The more advanced framework was proving too hard to debug, but other than that it suited the current needs perfectly, might see if I can restore it and fix.

Date: Tue Feb 10 13:36:31 2015 +1300

Author: Jerry Fan

Subject: Auton rework to newer framework

Comments:

I have included the newer framework again in this latest commit, I have separate each part of the function into separate mini functions so that I have more control over the code and that it is easier to debug. This is pending testing again.

Date: Tue Feb 10 18:08:53 2015 +1300

Author: Kristian Hansen

Subject: Change needle piston mode

Comments:

Changed it from a toggle to a single 'on' and 'off'. This should work for you Marco.

Date: Thu Feb 12 03:33:54 2015 +1300

Author: Jerry Fan

Subject: Code update

Comments:

This update brings about the fixes for the new auton framework, which is fully working now due to the additional debugging functions I managed to create. Because everything was so modular, I could just comment out sections and add in code bit by bit, to find the offending function. I found a few functions that were written incorrectly and corrected accordingly.

Auton essentially runs as follows:

Picks up skyrise from holder, goes back into position, lift goes down and drops skyrise into the skyrise base. Then it lifts up to the correct height for loader before moving back again. A light sensor delays the claw closing until a piece is detected in the holder, with a delay so human error is accounted for.

Other additions with this patch includes a skyrise height preset toggle, which will go up to each of the skyrise heights accordingly, and also an autoloader height preset.

Date: Thu Feb 12 04:07:35 2015 +1300

Author: Jerry Fan

Subject: Syntax Fixes

Comments:

A few errors with semicolons and such.