

# LabVIEW Robot

## Pasco-Ray Programming Guide

Team RUSH 27 LabVIEW

**Matt Pasco**

**Michael Ray**

# **Robot**

## **Table of Contents**

1. Making a robot project

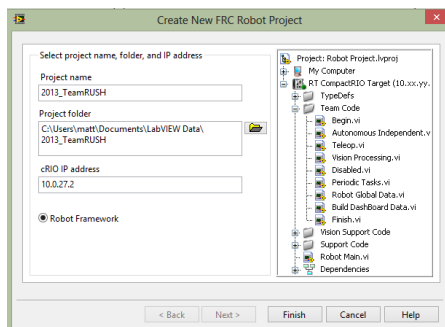
# Setup a Robot Project

1. Open up LabVIEW and select FRC cRIO Robot Project



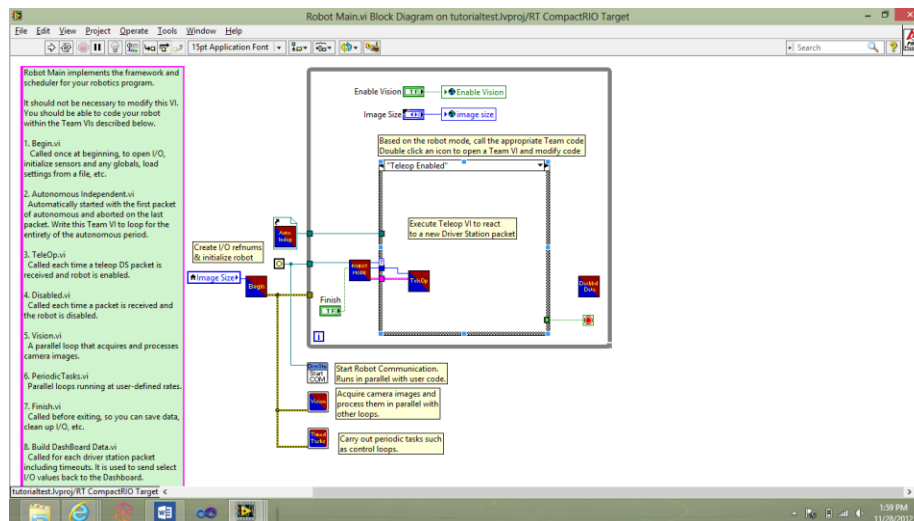
2. In the create new FRC robot project

- a. Project name: *year\_teamname*
- b. IP 10. TE.AM.2 so for Team Rush since our number is 0027 it would be 10.0.27.2



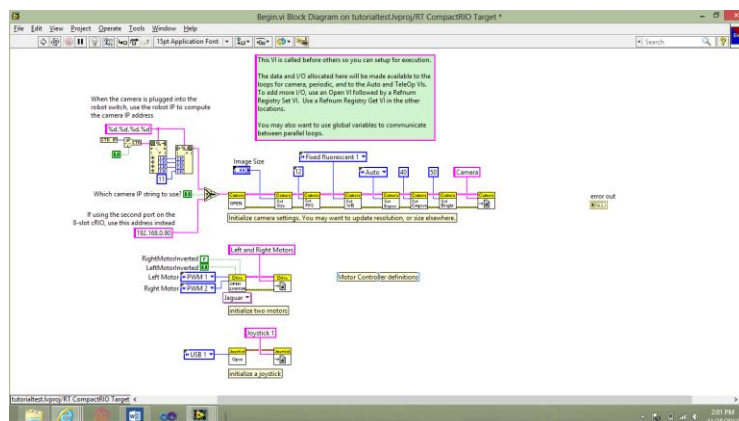
Now you have a robot project just select *Robot Main* to go into the code

# Inside the Robot Project



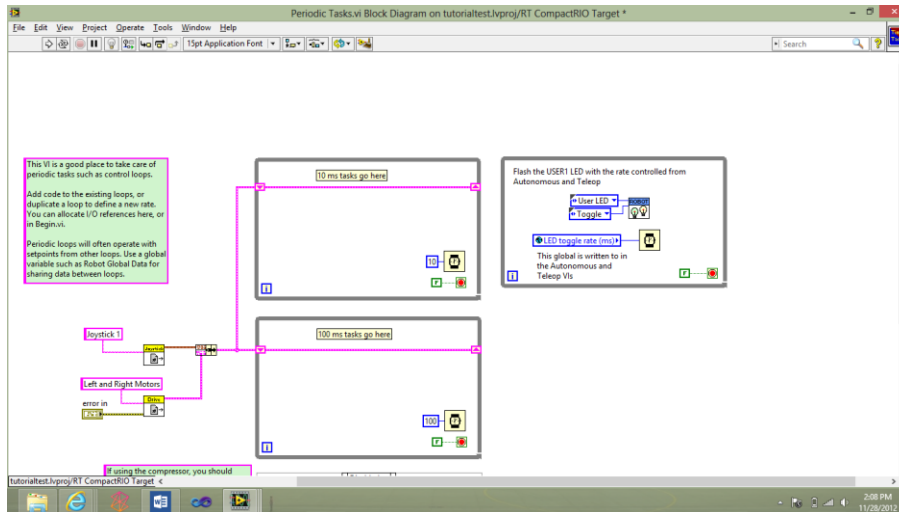
There are many “blocks” on the block diagram these are called VI’s

## Begin



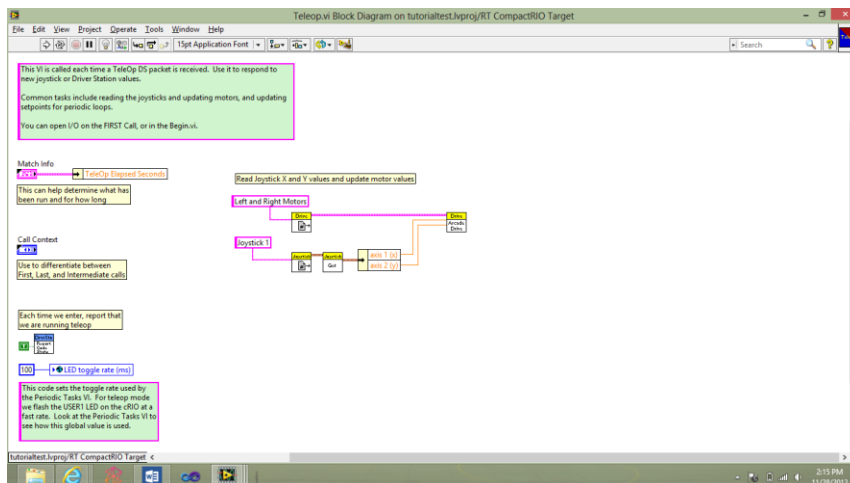
In begin we define or declare all of our refnums for Camera, motors, joystick, and more. As well as we import our CSV files from the robot which will be discussed in later topics

## Periodic Tasks



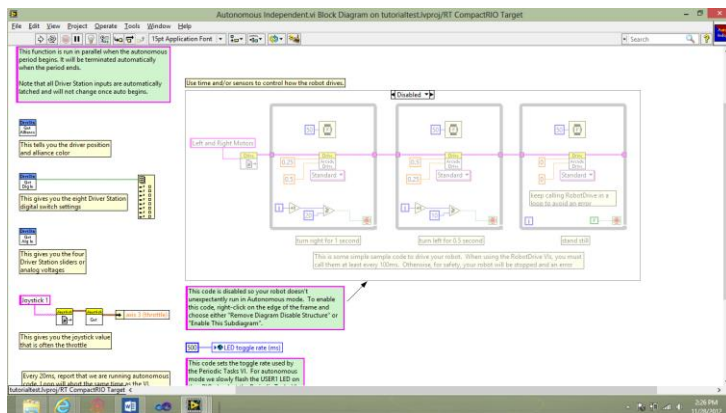
We read from sensors we do **no motor control in here** only sensor reading

## Teleoperated



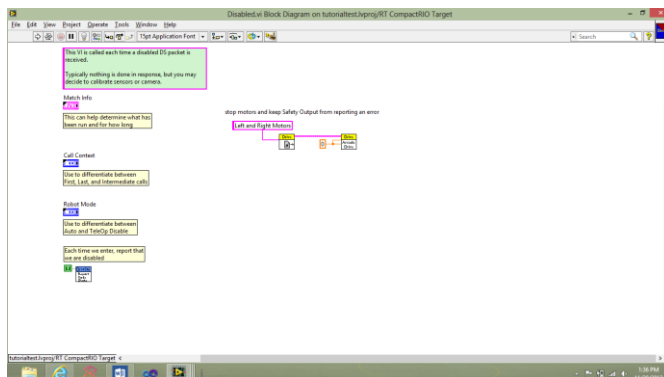
In Teleoperated (human control) this is where we read from all of our joysticks and put out motor values to the power and do logic for teleoperated in here as well as import sensor input from periodic tasks.

# Auton



This is where we do out logic for auton with motor control with logic for drive train and use data from sensor reading.

# Disabled



This is what runs when the robot is disabled where we make sure all motors are set to 0

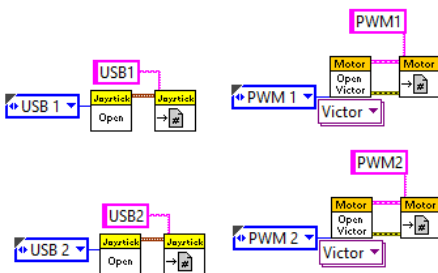
## **Short Cuts**

- Ctrl + T opens shows both screens side by side
- Ctrl + E shows other window
- Ctrl + spcbr opens up quick search
- Ctrl + C copy
- Ctrl + X cut
- Ctrl + V paste
- Ctrl + Z undo
- F1 run/deploys code

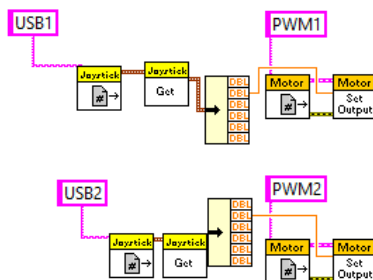
## Refnum & Open

Refnums are used to declare motors, sensors and joysticks and further use them. There are two things you do with refnums

- Get
- Set



Set- is used with open, we use set to tell the cRIO we have a motor. With open we give it a name like "PWM 2" or "Right\_Drive" then wire it into the get and tell it which PWM port the motor controller is connected to. We do the same thing for sensors and joysticks.

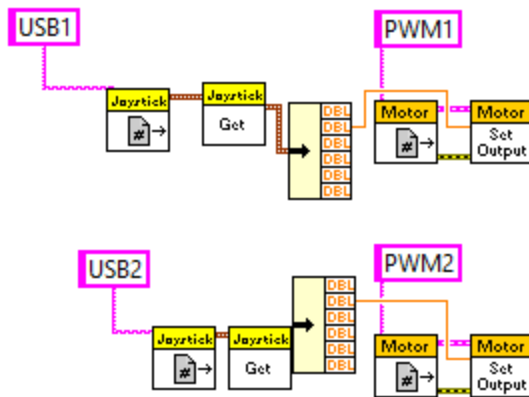


Get- is used with setting the power using the *setoutput* block. So we use the get block using the same name we used in the *Set* block and then we can send a value between -1 and 1 from the joysticks to the *setoutput* block.

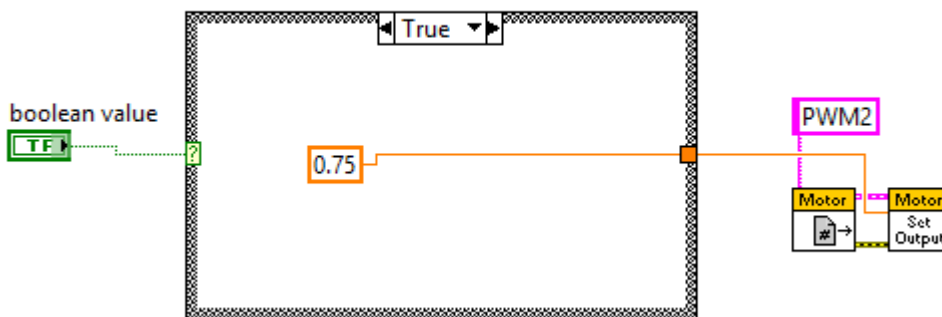
*Note: the axes and buttons of the joysticks are in an cluster (like an array) so we had to unbundle it*



# Motors

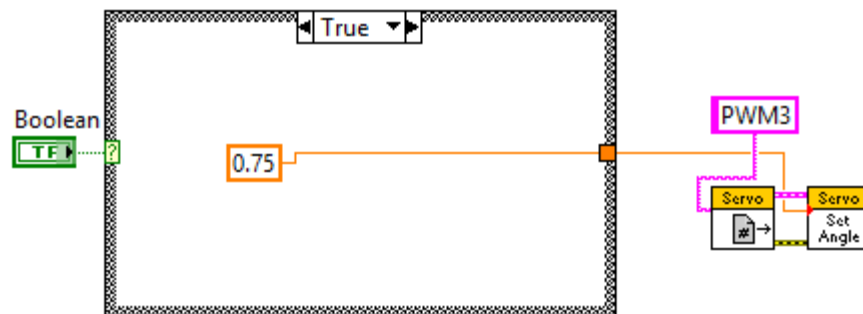


First you must use refnums to declare them in the *begin.vi* as either a victor or a jaguar motor controller. Then in the *teleoperated.vi* we pull the value (referencing it using get) from the joysticks and feed them to the set output motor.



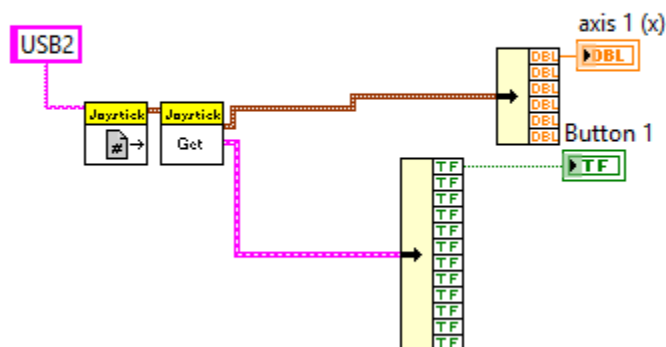
You can also use constants to drive the motors.

## Servo



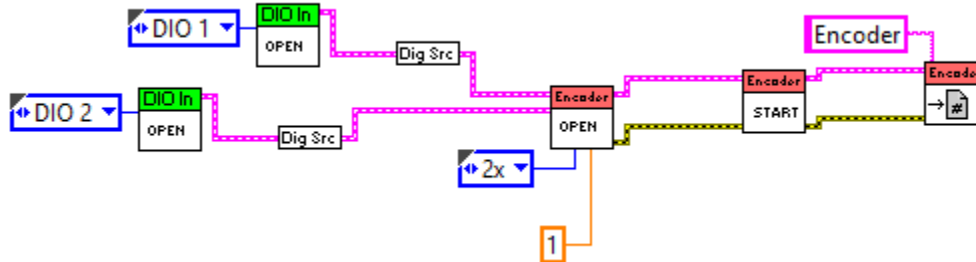
A Servo is a motor with sensors so that it will hold at a certain angle you send it automatically. We use these if we need a precise angle or something that needs to be held at a certain angle. Like with shifting in the drive train

## Joysticks

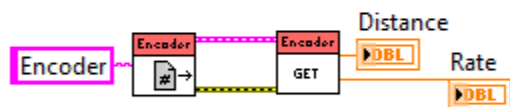
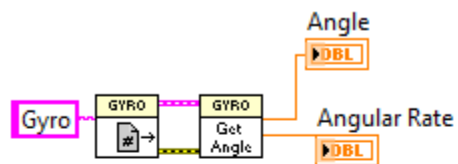


Like with motors we must create a refnum. Once it is created you unbundle the cluster (like an array) of data from the get block for the joystick. The top one is for all of the axes and the bottom for all the buttons.

# Sensors

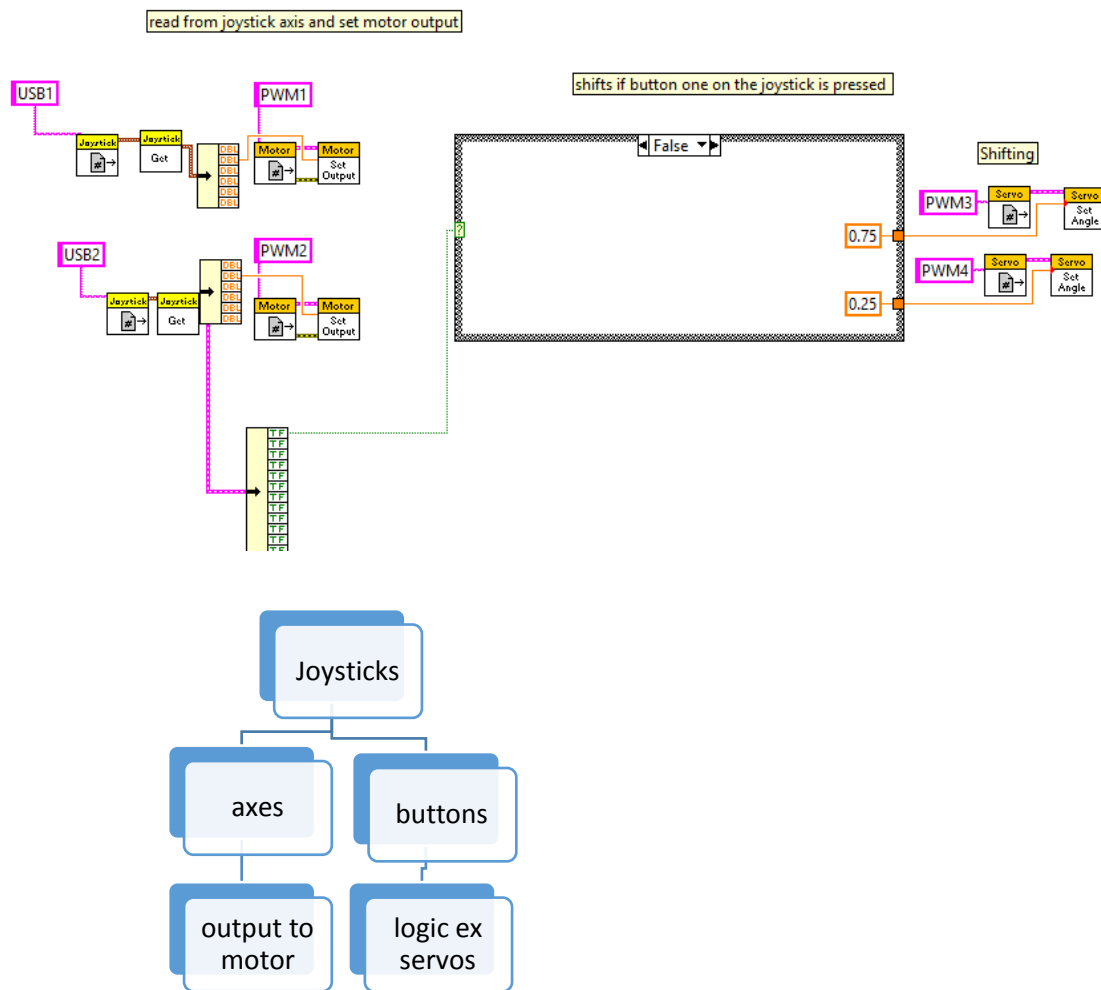


To set up the refnum of an encoder we must first open a digital io port then convert it into a digital source so that it can be imported to the encoder data. We import it into the encoder so that we can get the number of ticks/ rotation. The orange double connected is to modify the rate of the encoder. Each encoder's ticks per inch is different and will have to be modified.



To read from a sensor you use the get or getangle for the sensors. As you can see we reference the sensors name and then use the get block to get the angle or distance.

# Drivetrain

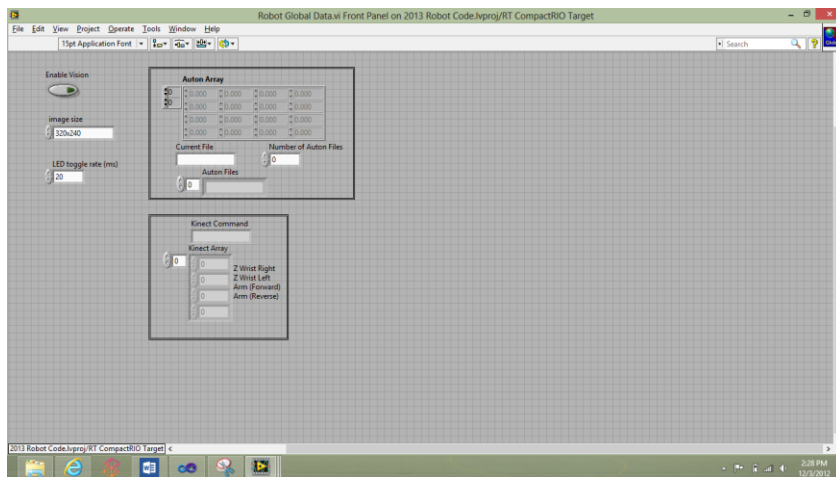


From the joysticks we get the axes and the buttons. We use the axes to control the motors and the buttons as an example to change the servo positions.

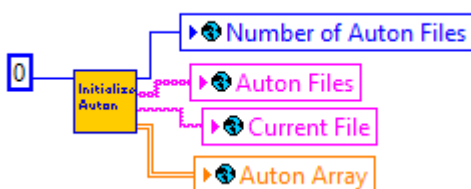
# Globals

Globals are short for Global Variables and it is a VI specifically for storing data such as integers Booleans and any other form of data. We use this so that if we need to share a value in multiple VI's we are able to.

This is the Robot Global Data.vi where everything is saved

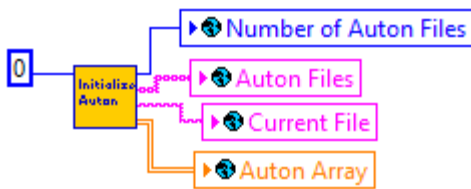
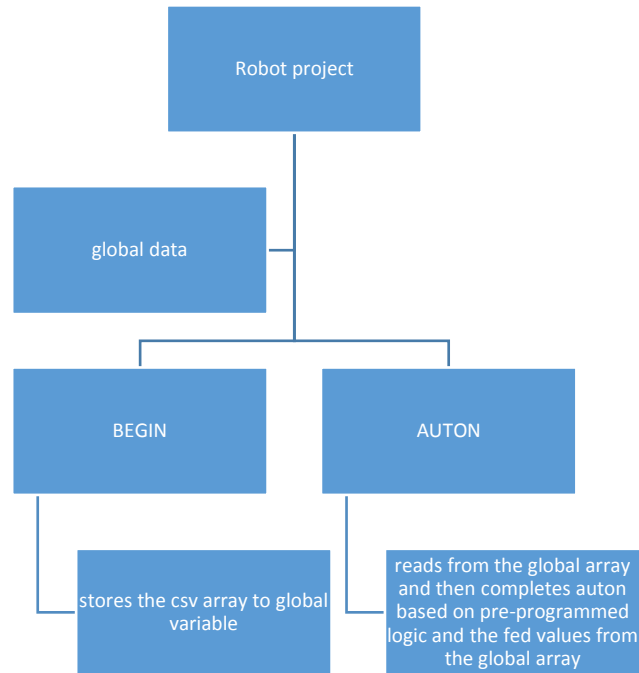


This is an example of a global in the begin.vi

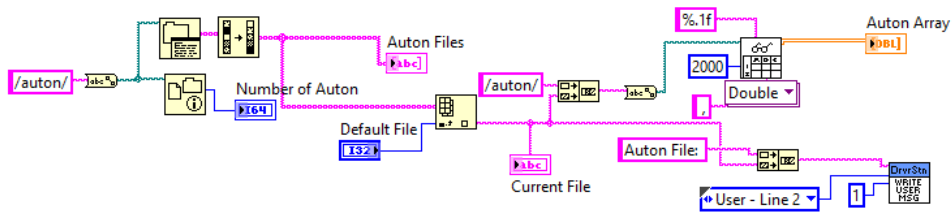


To declare go to the robotglobaldata.vi and right click and add or

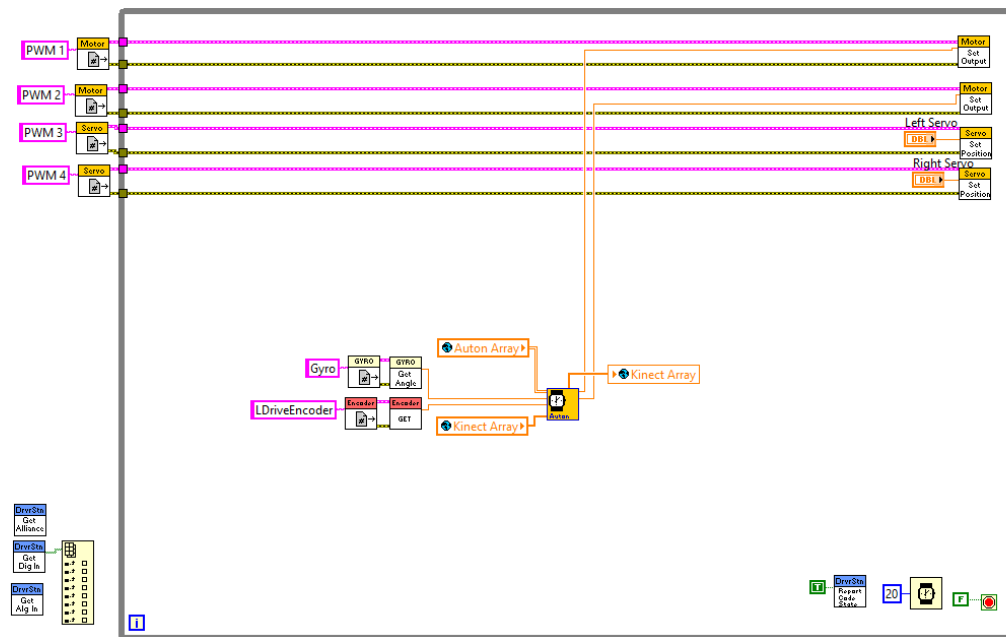
# Auton



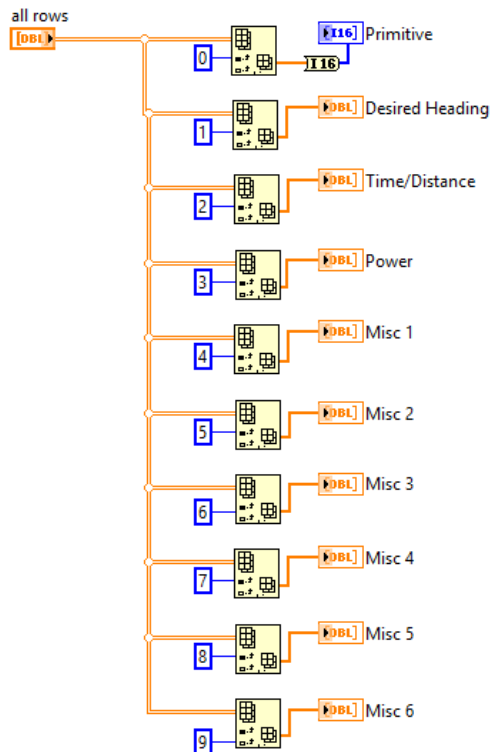
This code below is inside the “intialize auton” block



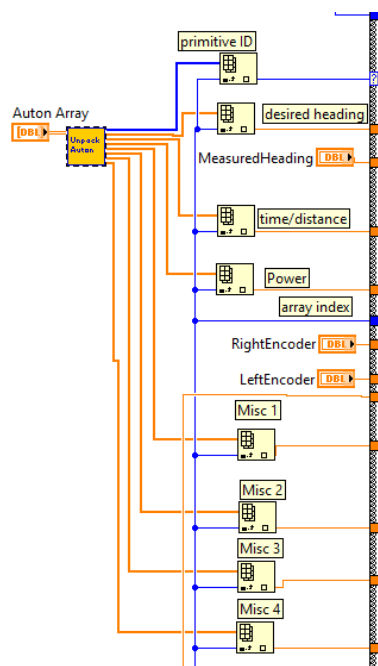
This will pull all CSV files and store them into an array called “auton files”. Then we select one CSV file and read from that CSV and store them into an array called, “auton array”. For example our CSV file we created will be in the /auton/ subfolder and then we select which one we want to be stored into the array. Also displays onto the dashboard of the driver station.



This is the very top level of auton. We use our get refnum for our motors and sensors and run it in a 20ms loop, and read from the saved auton files.

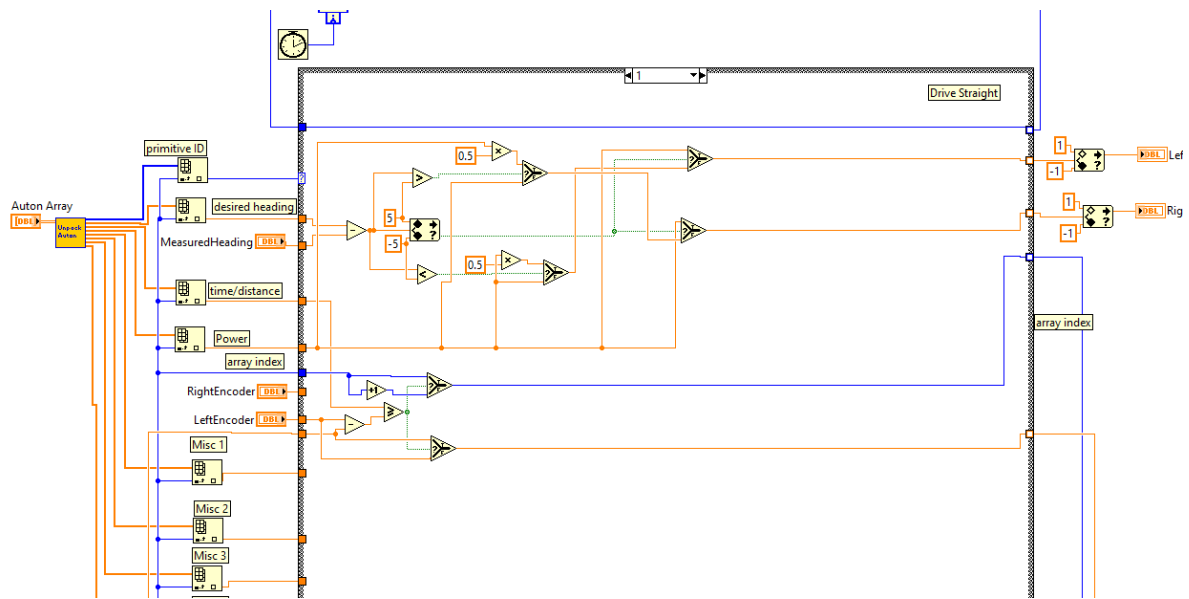


This reads from the main 2-D array and then index' them into 1-D Arrays

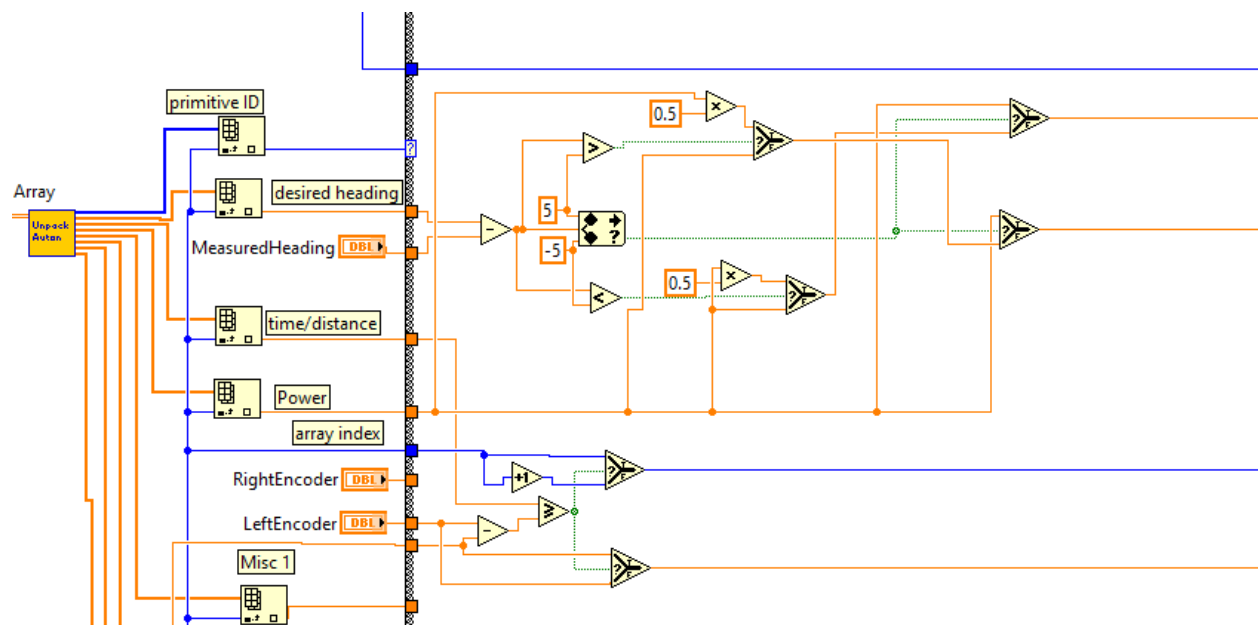


Inside of the unpack auton is the image above and then we read from those 1-D arrays by indexing them by the current loop iteration.



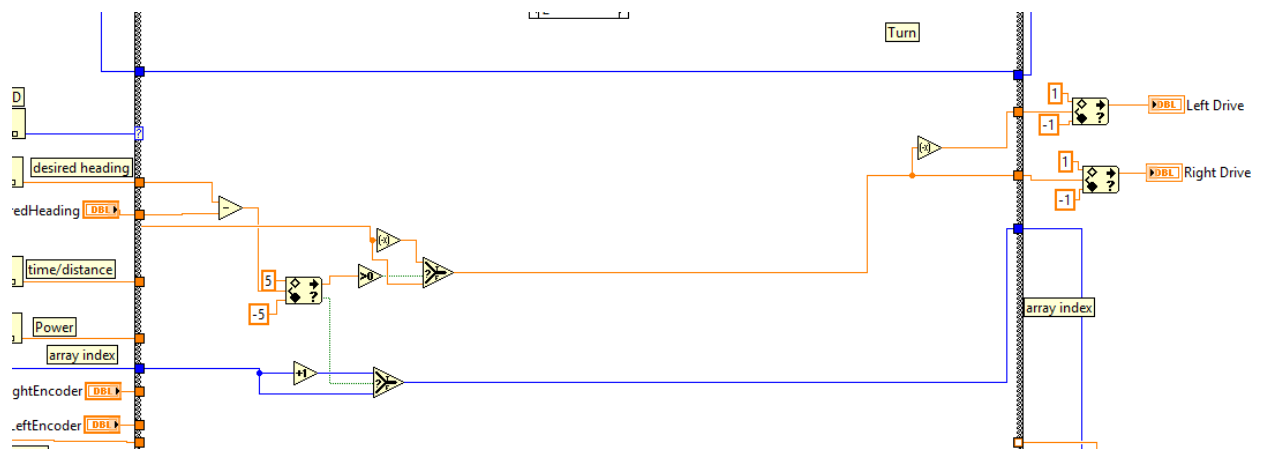


This is the full image of auton with case 1 which is our drive straightening and then after our logic is put in we output our motor values.



## Drive Straight Auton

In drive straightening we find the delta angle (desired heading-current) and then if we're heading right and we should be going straight, the robot will halve the power on the left so that it will begin to turn left and straight out. Even though you cannot physically see it, the robot is wiggling to the right and left so that it is going almost perfectly straight.



## Turning Auton

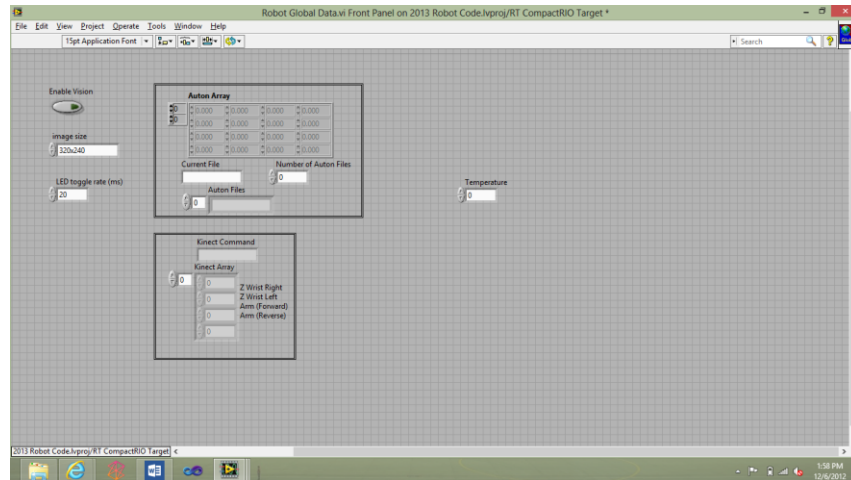
In the turning auton we find the delta angle and then determine if we need to turn right or turn left and then flip the value (\*-1) of the motor that needs to be negated so we can turn and then it increments to the next index of the array.

## Editing Driverstation

### Subject to change with 2013 driverstation

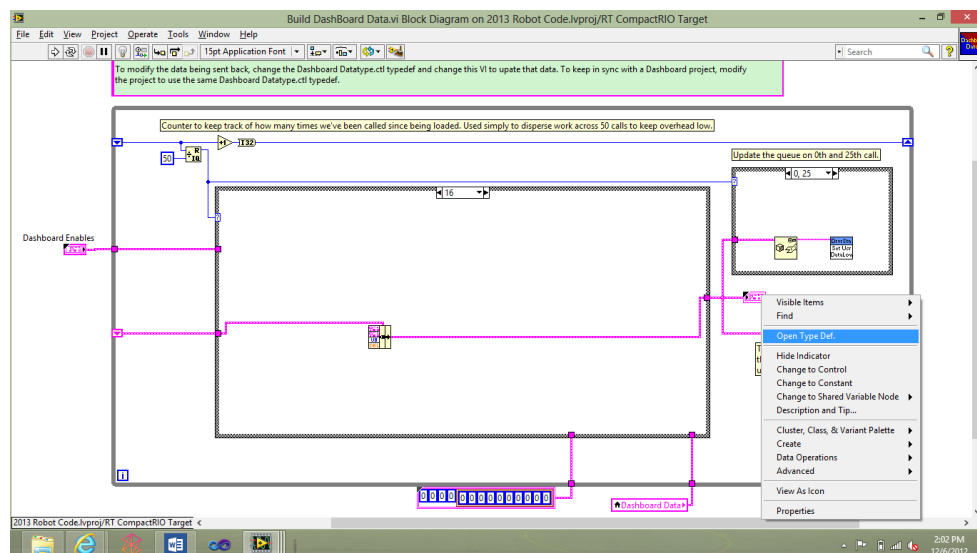
1. Create a global variable (right now I am using Temperature)

a.



2. Plug whatever logic you want into your variable in whatever vi you want (ex. read temperature in periodic tasks and set the temp value to the temperature global)
3. Open build dashboard

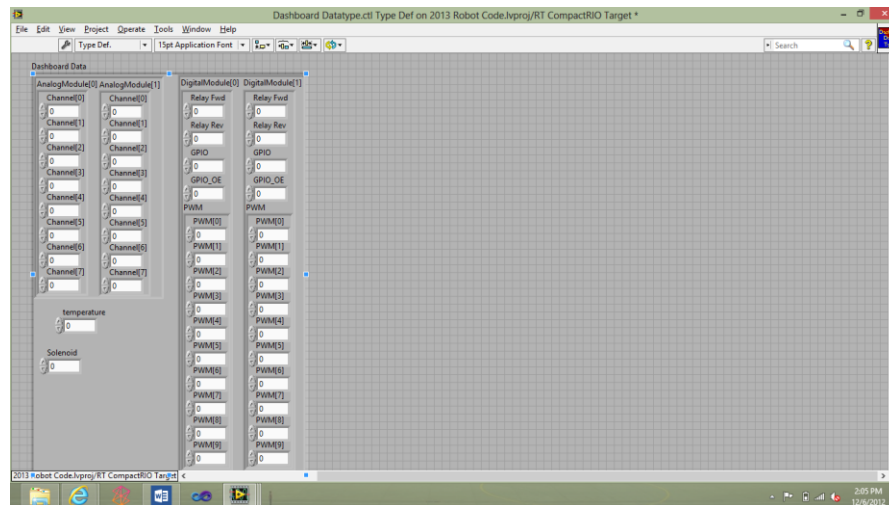
a.



- b. Right click on dashboard data type (the pink weird box) and select open type def

4. Right Click and add a variable (temperature)

a.

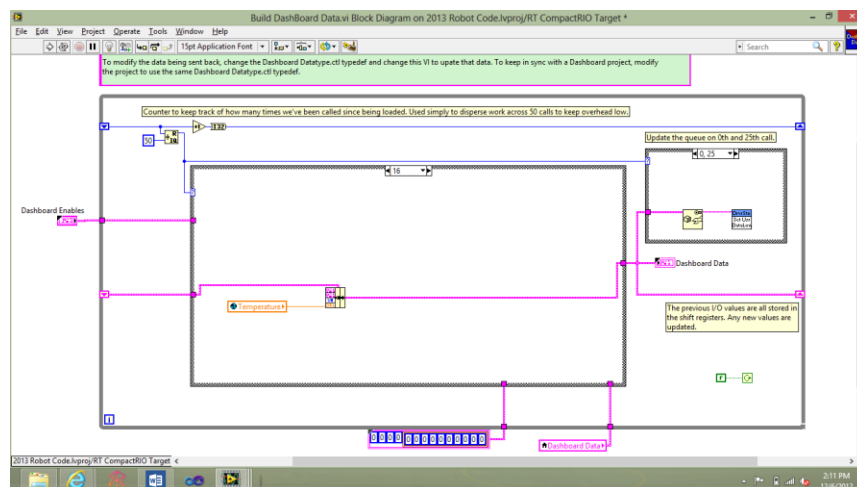


b. Save and close this vi

5. Now show the build dashboard data type front panel (you should see temperature there)

6. Add a new case in the case structure it doesn't matter what number it is just add case after

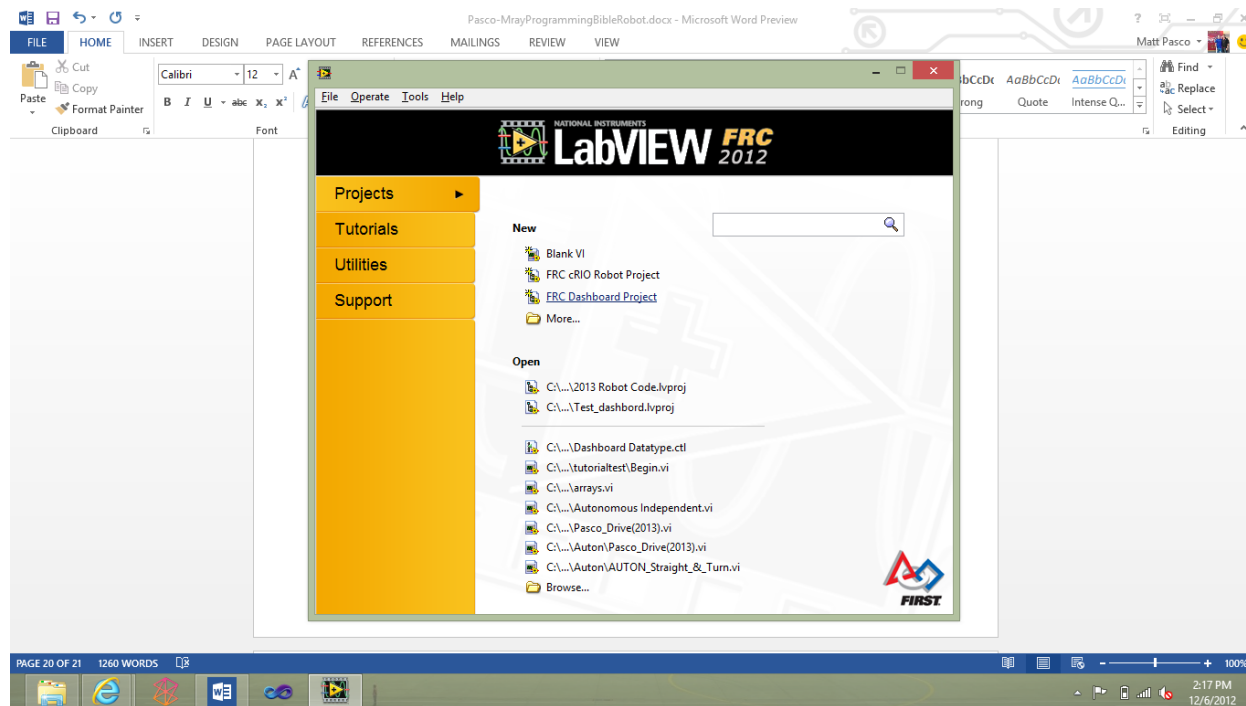
a.



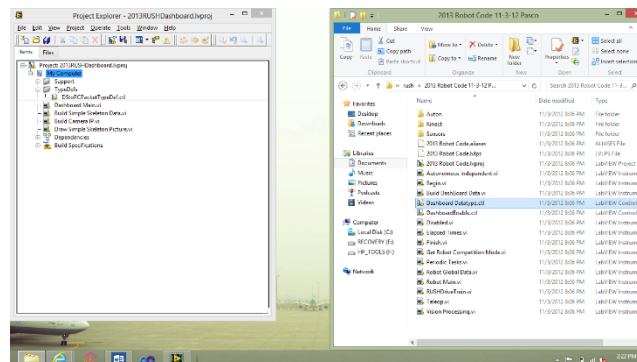
b. Insert bundle and insert the cluster into the bundle (on top where it say cluster)

c. You should see variables on the bundle (you'll see your added variable)

- d. Plug your global into the cluster with the corresponding text
7. Save all and close the robot project
8. Now on the front page of LabVIEW click on new dashboard project
  - a.

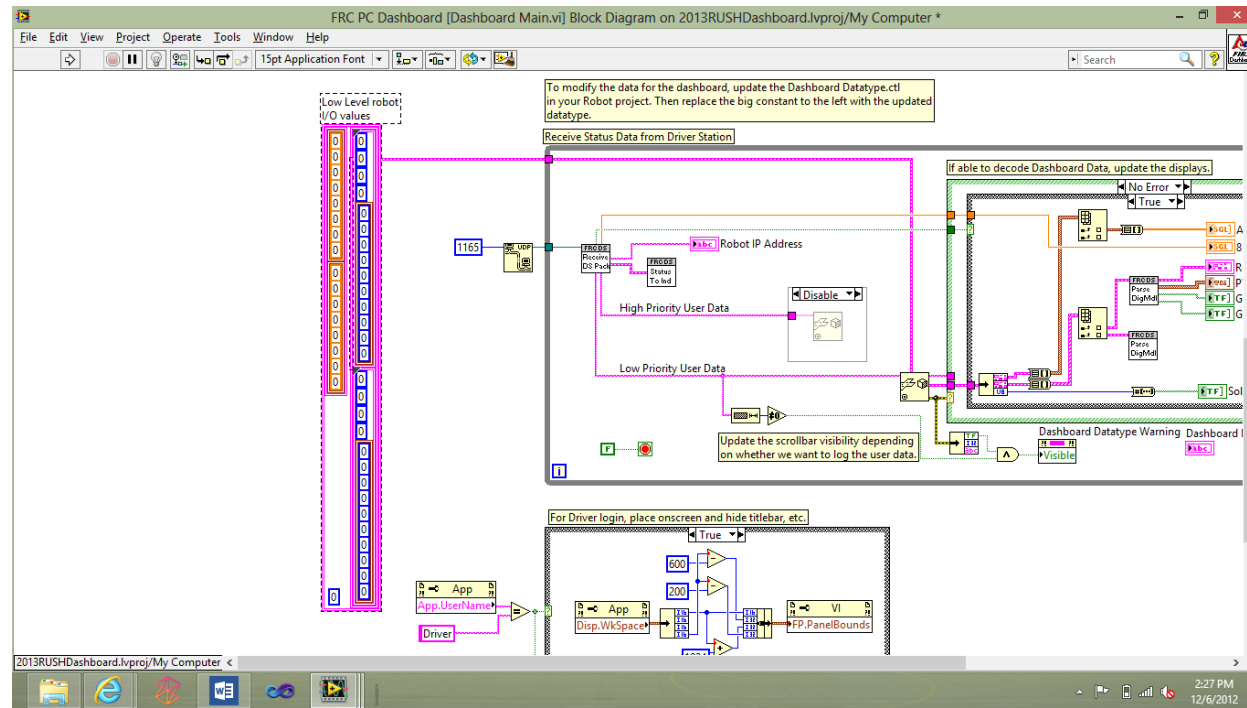


9. Name it whatever you want (2013RUSHDASHBOARD)
10. Now open up windows explorer and find where your normal robot code is (C:\Users\username\Documents\LabVIEW Data\)
- a. Now copy dashboard datatype.ctl to the typedefs folder



- b.
11. Open Dashboard Main.vi

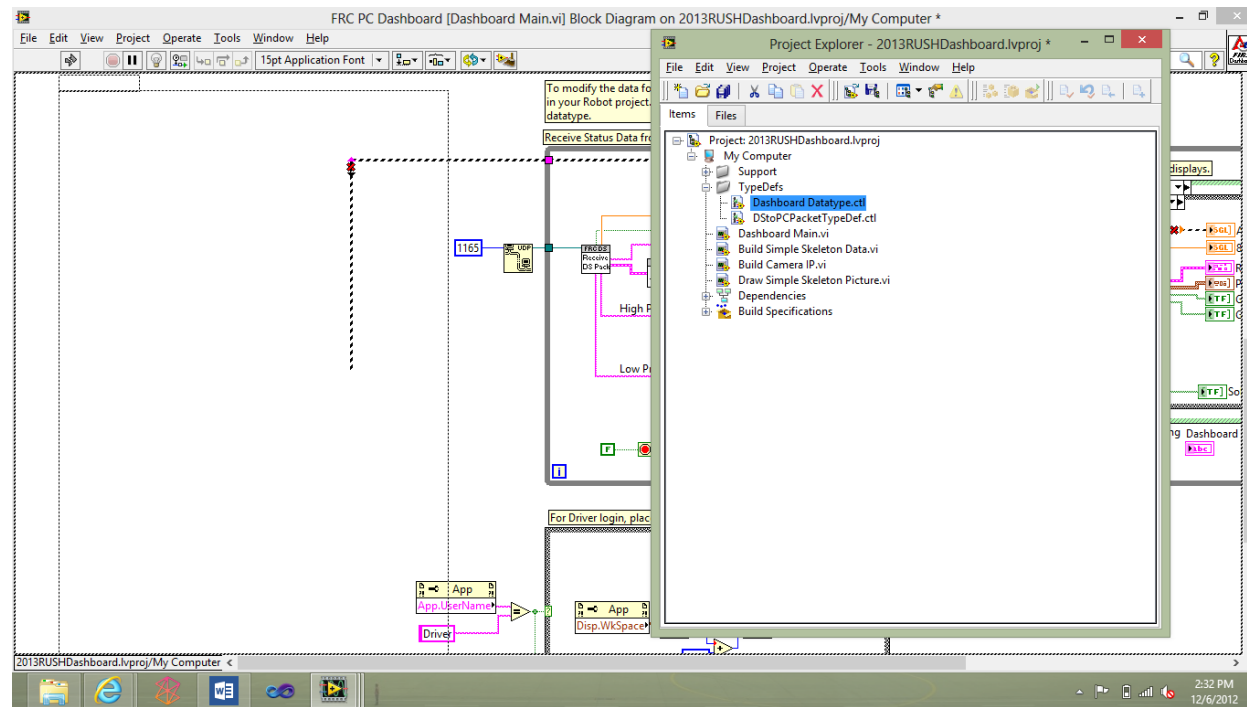
## 12. Delete



a.

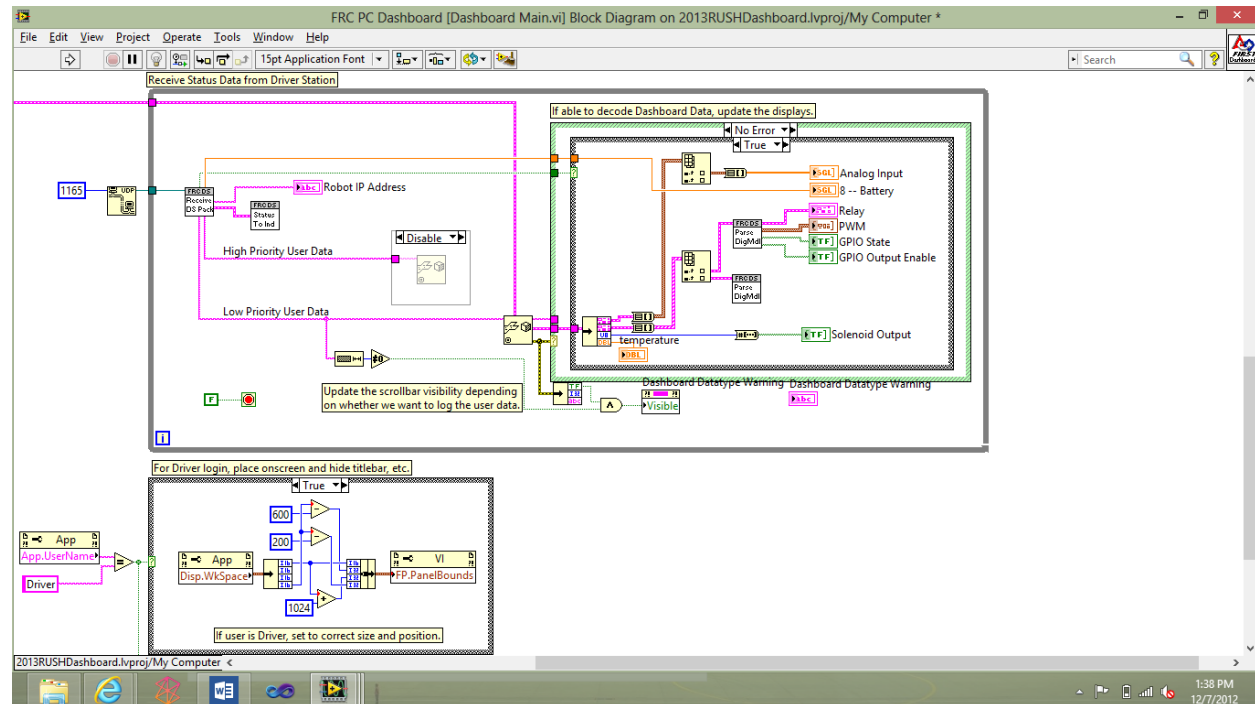
## 13. Now drag dashboard datatype.ctl

a.

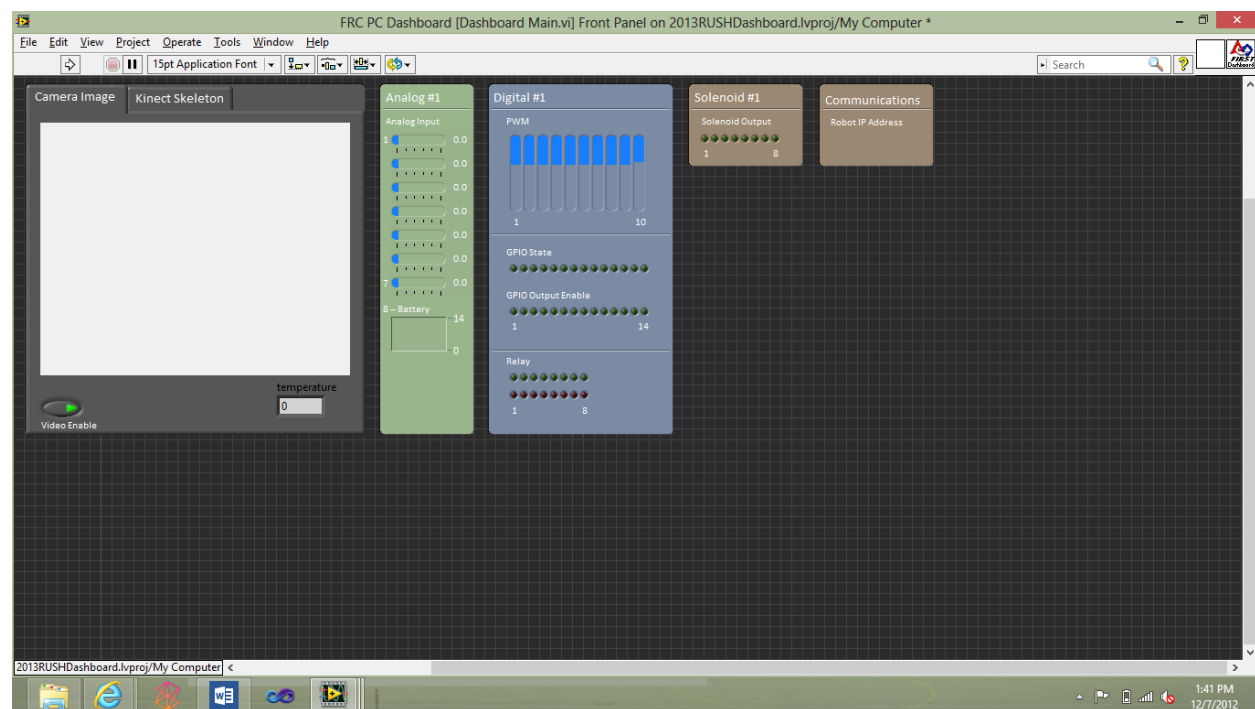


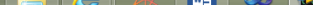
b. Hook the existing wire back up

14. Now you should see your variable here

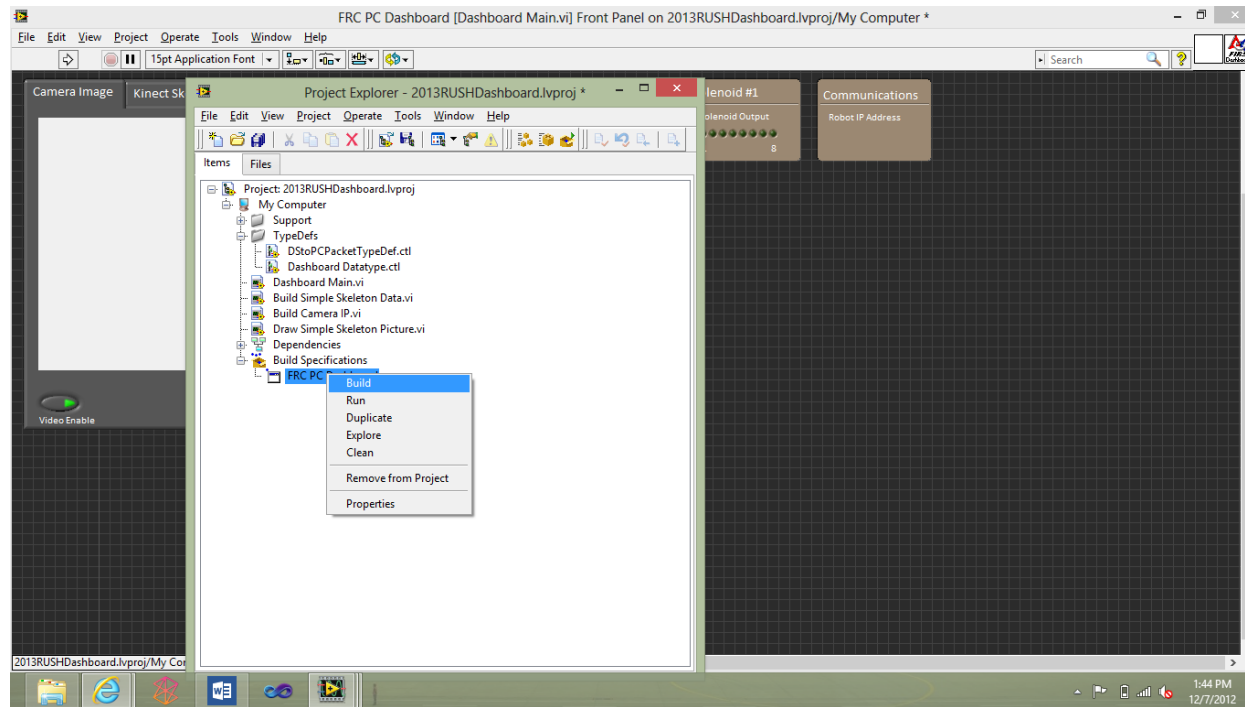


- a. 

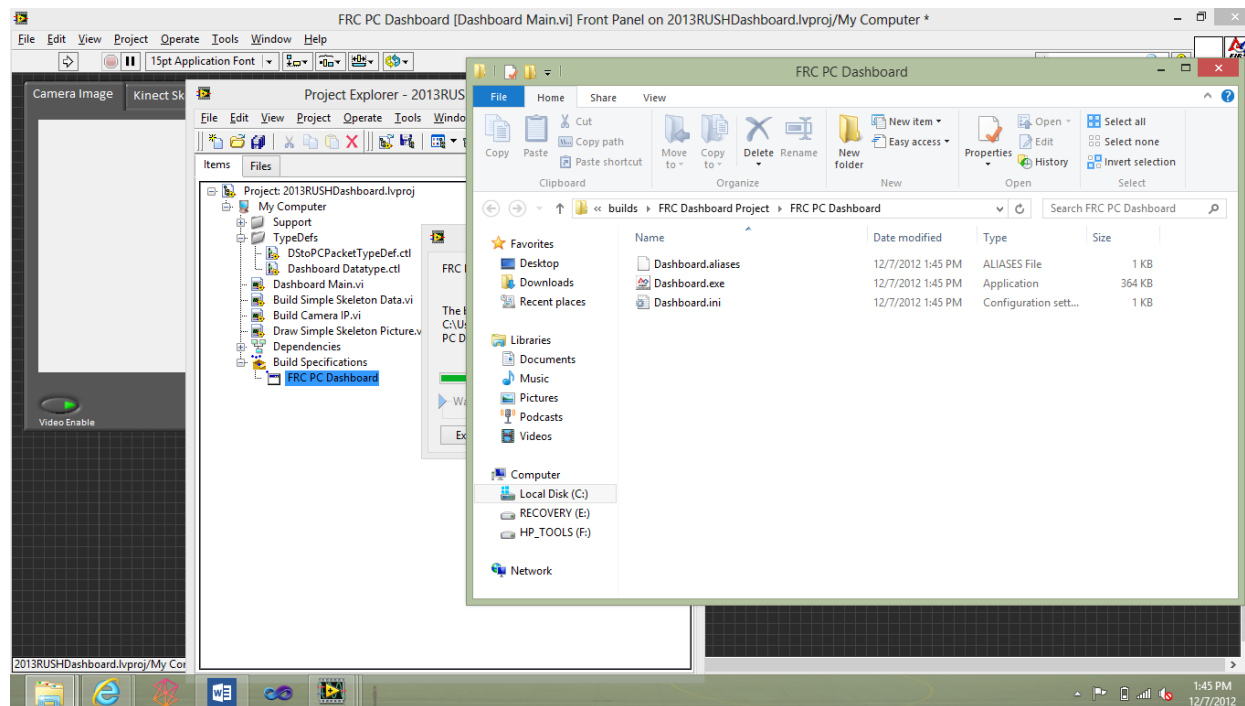


- c. 
- d. Now on your front panel you will see temperature, It will be easier to see when the code is actually running
- e. save

15. Now on the project explorer right click on FRC PC Dashboard and press build

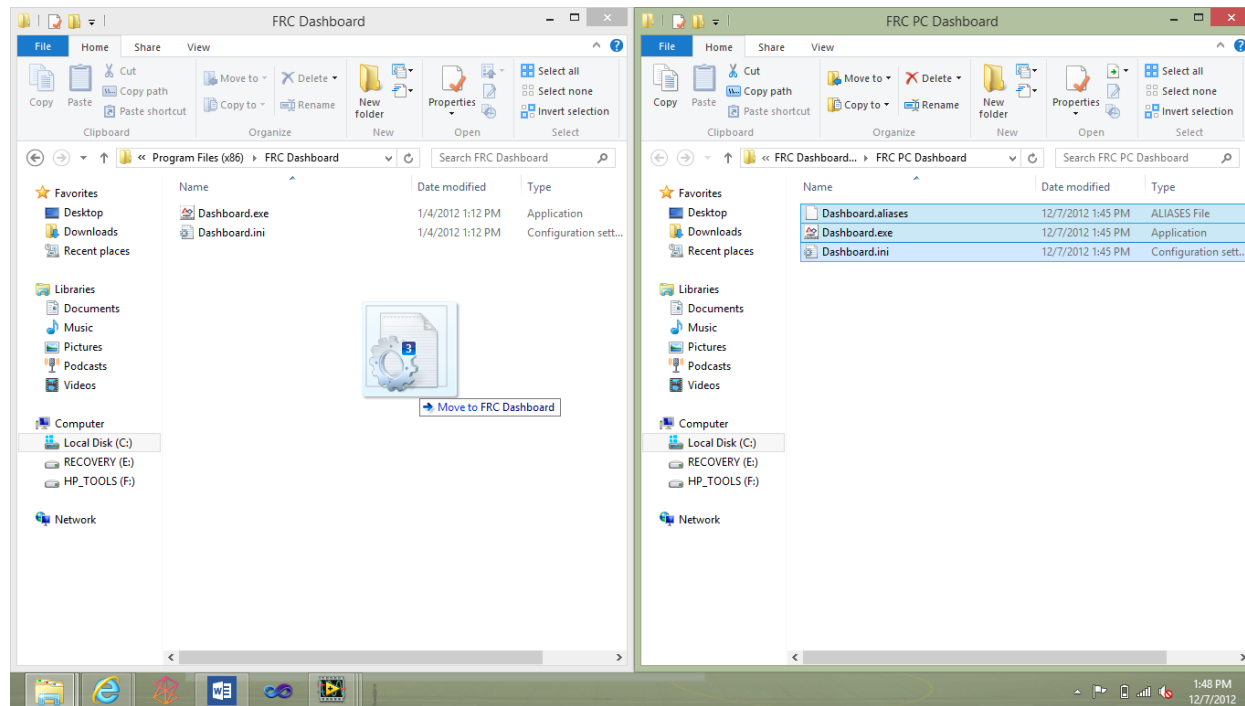


- a.
- b. Now once it is complete press explorer and it will bring up your explorer with the built exe in the folders like this:





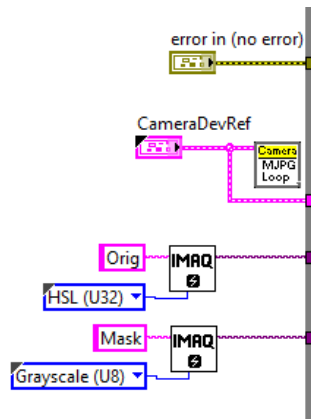
- c. Now copy these items and make a backup folder for the files with the same name in your program files (C:\Program Files (x86)\FRC Dashboard)



- d.
- e. Now the next time you run the robot with the dashboard it should show temperature.

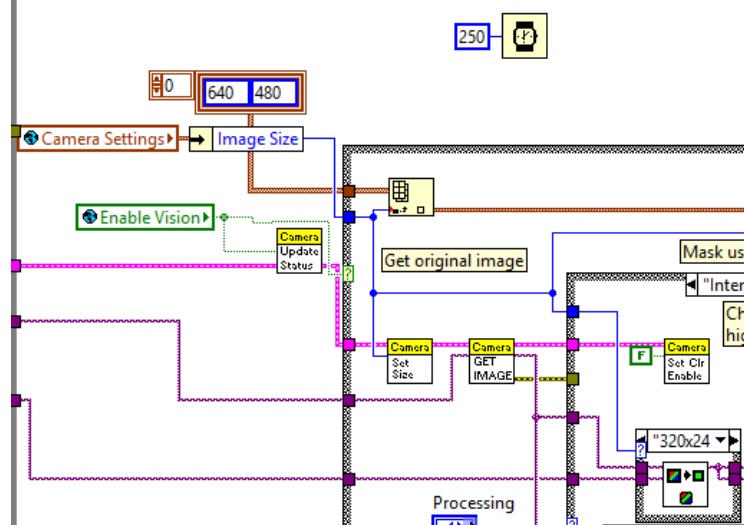
# Vision Tracking

1.



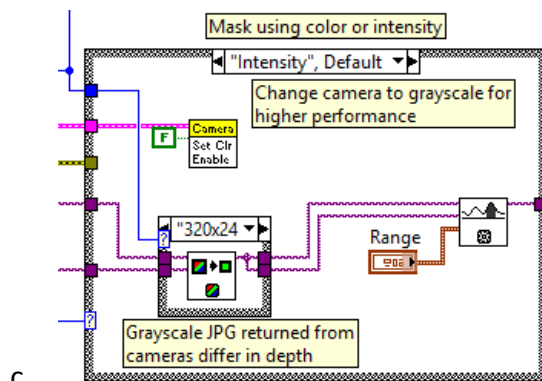
a.

Grabbing Camera Dev ref and making a grayscale Mask (we had to for processing power)

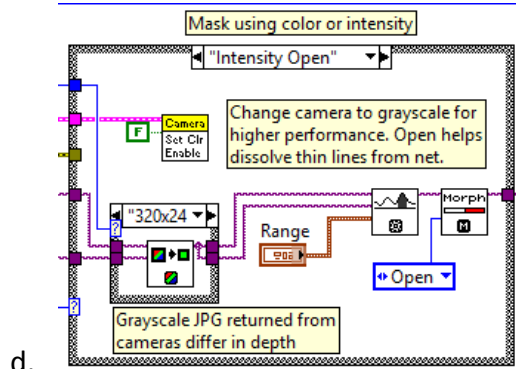


b.

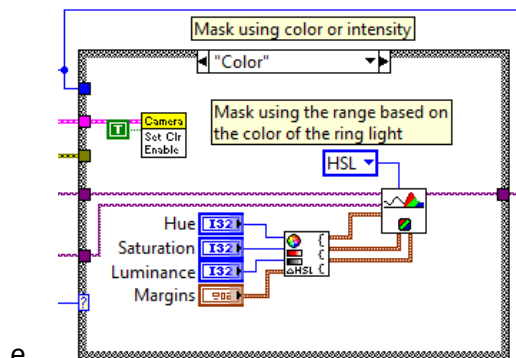
- i. We set our loop iteration to 250ms (This can be any number we set ours to 250 just for processing you can set it to whatever but any faster might lead to lag depending on what youre using)
- ii. We set the image size it should be the same size it is on your dashboard
- iii. We enable our vision
- iv. Send through our camera image



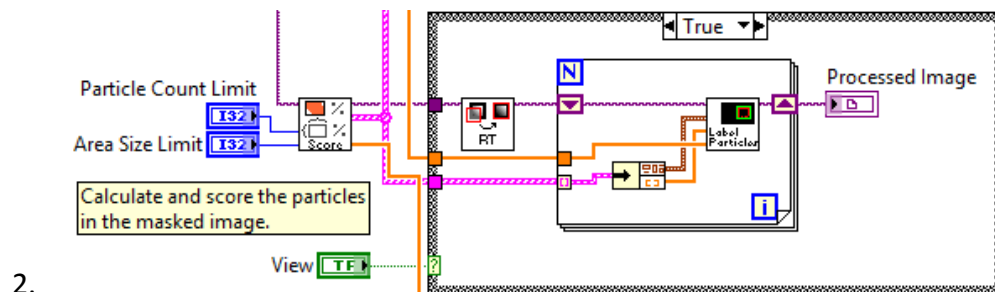
- i. Scales the image if need to and pulls preset intensity values (range) and then filters image so that it only gets the selected intensity value (gets only the bright parts of the image)
- ii. Set color enable tells it if it can display color



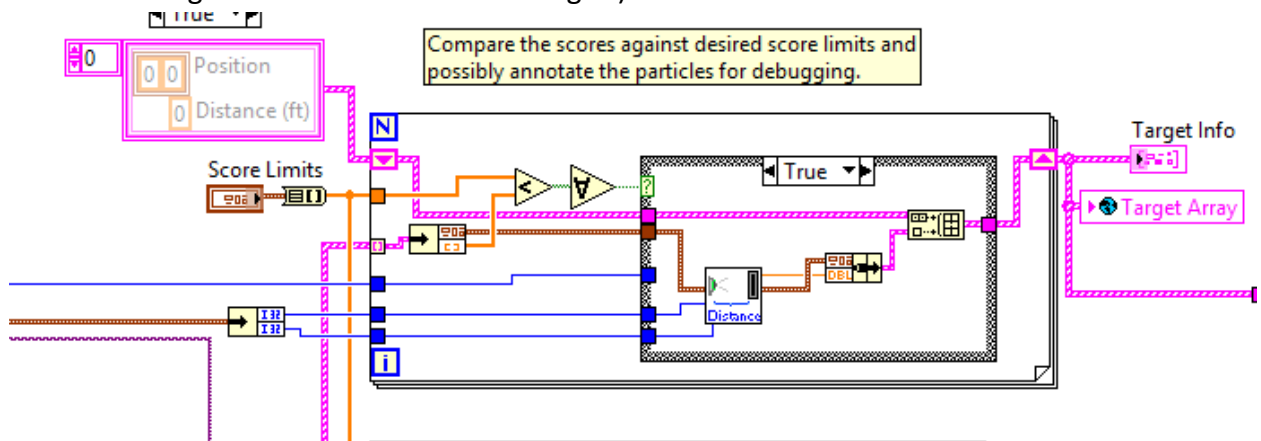
- i. Now it opens the image



- f. Applies HSL filter to original image and then sets it to the destination of mask

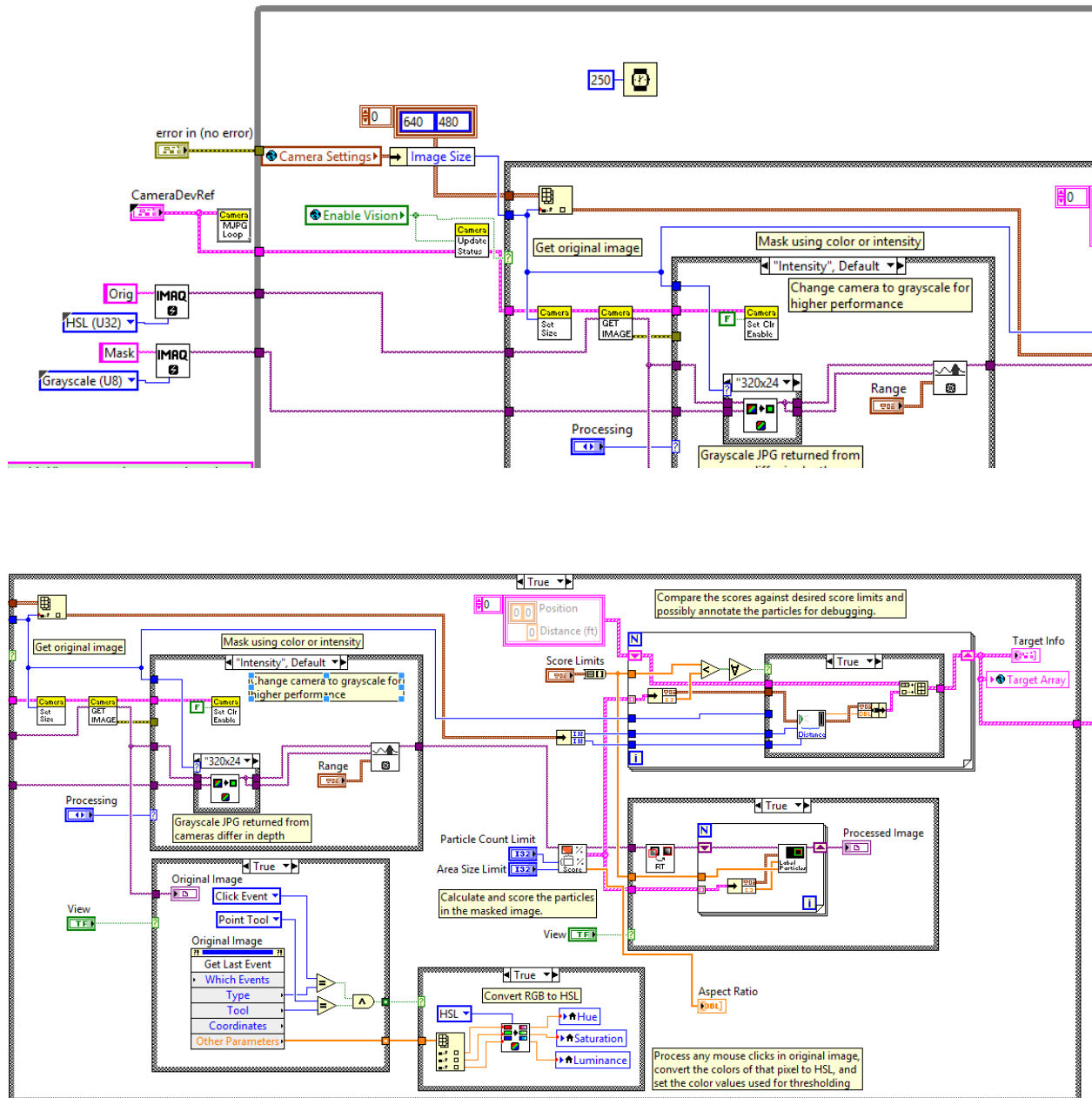


- a. Look for the rectangles, and create your aspect ratio (width/height)
- b. And then fill and color the rectangles (for example we make the high intensity rectangles we found into red rectangles)



- a. Now you can use the array to track

# Full Images

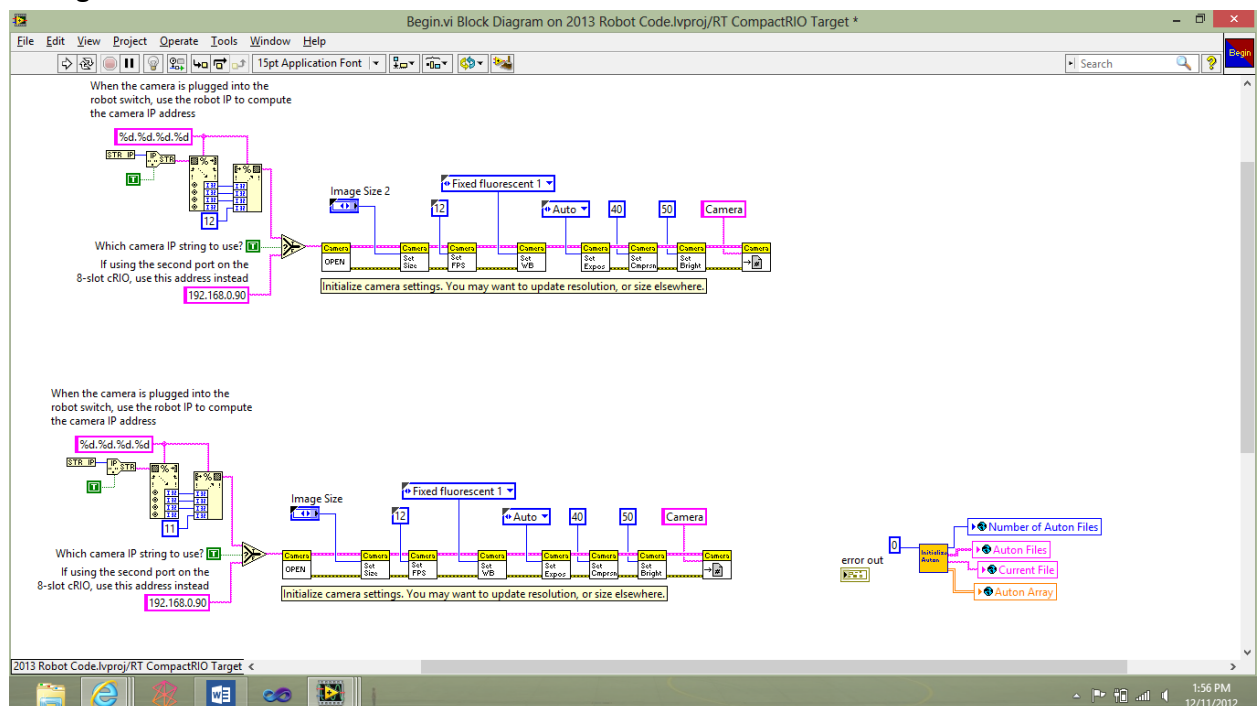


# Adding a second Camera

## For analysis

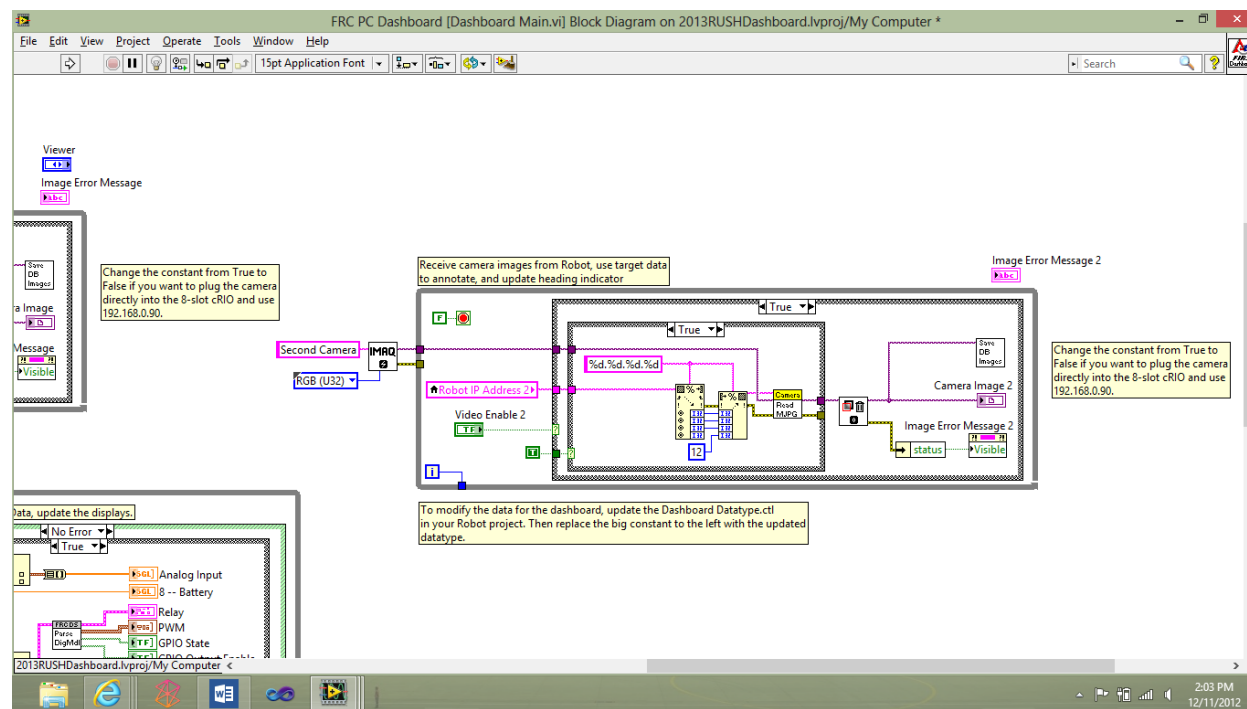
## Subject to change with 2013 driverstation

4. In begin copy and paste the declared vision setup
  - a. Change the IP to root.12



- b.
  - c. The top one is the edited version

1. In your dashboard code
2. Copy old camera Code
3. Paste it



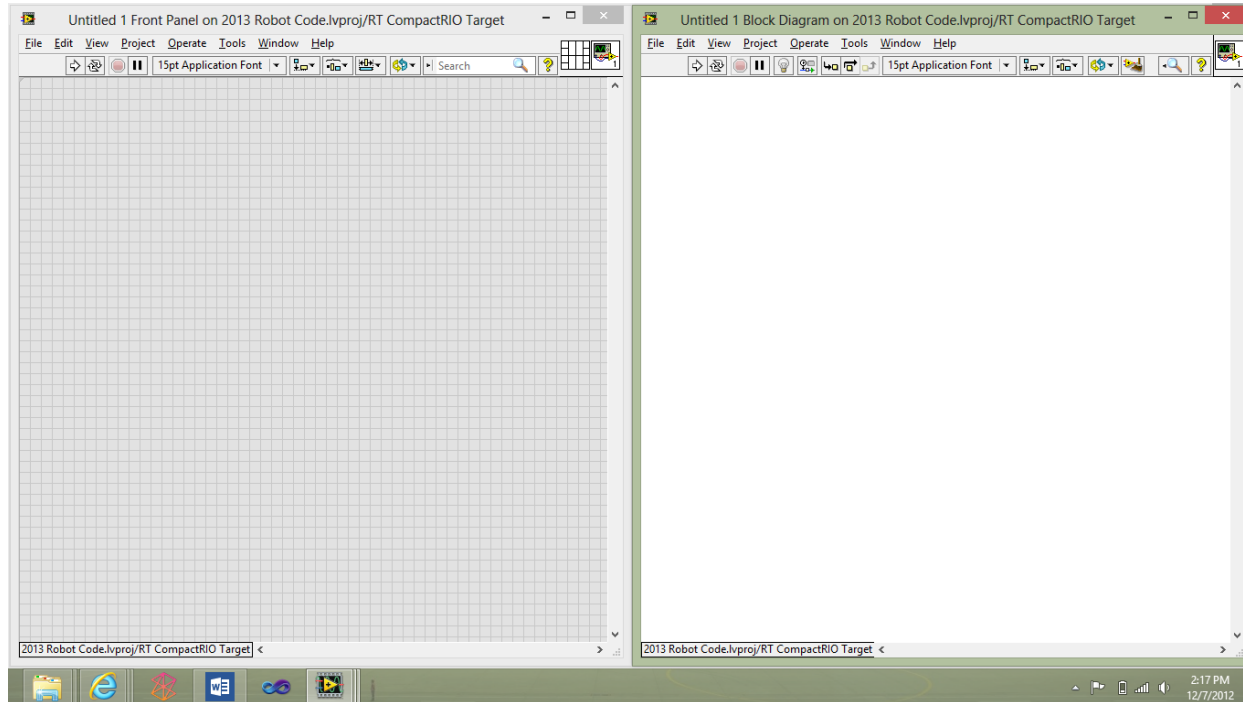
a.

4. Make Sure to rename the new camera (already renamed)
5. Change the IP to root.12 (12 has already been Change)
6. Edit the visual portion of the front panel ( you may have to create a tab for the analog and digital)
7. To create it just

## Making Custom Vi's

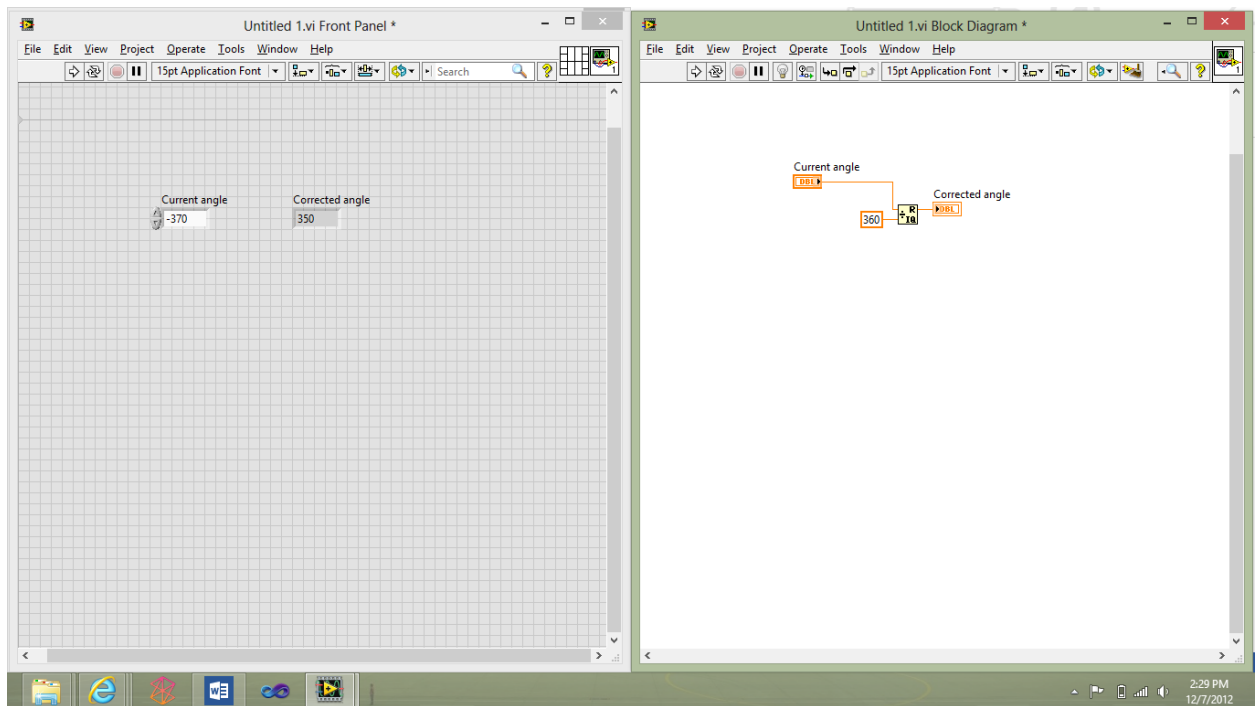
*Note: (You don't have to have a robot project open if you would like to create one and test it but you'll have to add it to the robot project later if you want to use it on the robot.)*

1. Open up LabVIEW and press new vi or if you are in a robot project press file new vi
  - a. You'll get a blank screen like this

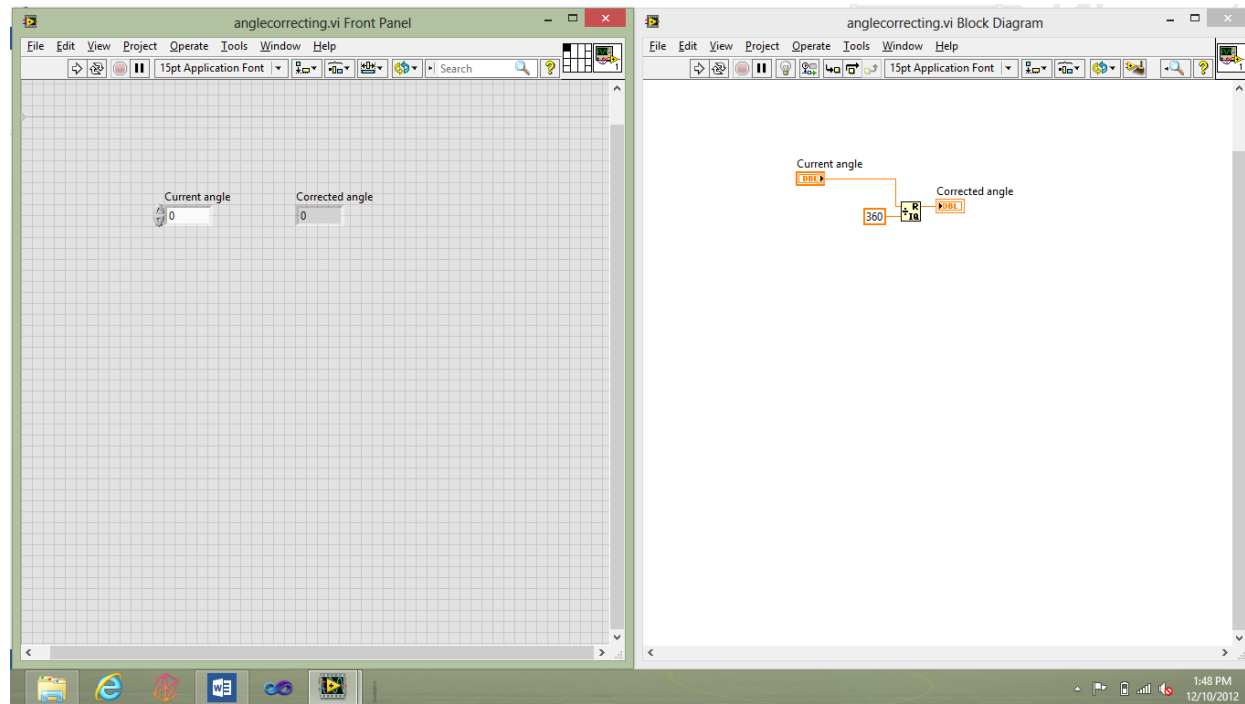


- b.
2. Now you can input whatever code you want
3. I'll show how to make a vi that if the current angle is greater than 360 (or less than -360)
4. Create a double control called current angle
5. We are going to use modulus division which gets the remainder after its divide by a certain number
  - a. Ex (54/5 you'll get a remainder of 4)

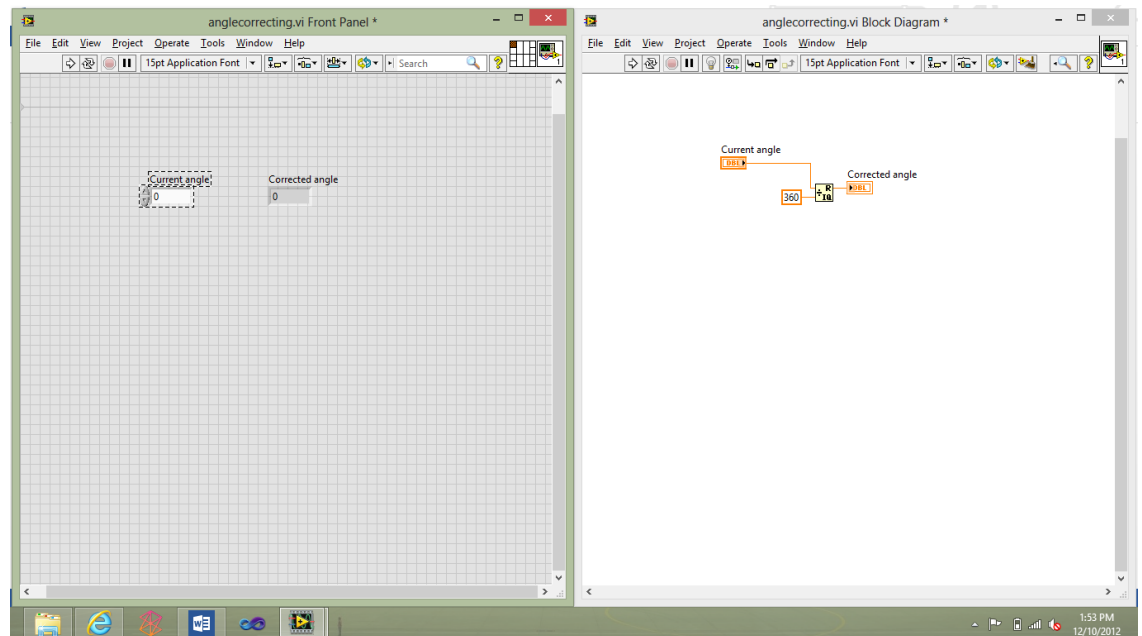




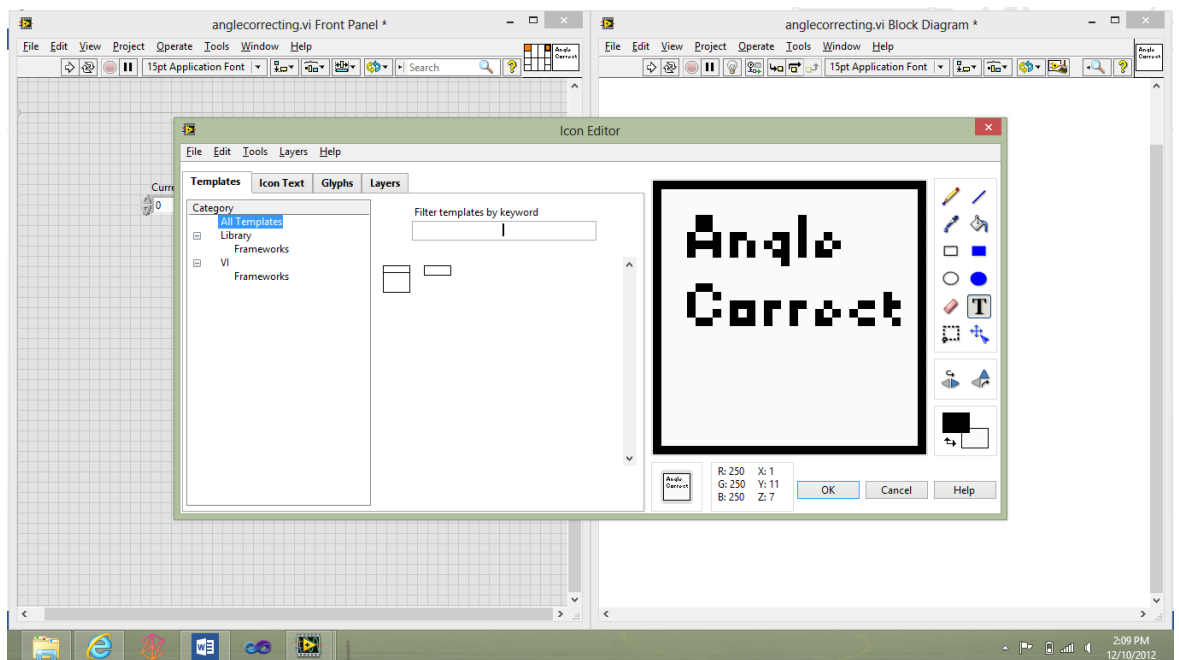
- 6.
7. So if the angle is -370 it would really be -10 degrees ( $370 \bmod 360$  is 10) which is also 350
8. Now to setup our vi for inputting an outputting
  - a. click on the grid in the top left (select a square)



- b.
- c. Then select the control (Current angle)



- d.
  - e. Now the square will be the color of the control
  - f. Do this for each control you want to input and output
9. Changing the image of the vi
- a. Double click the image right of the grid editor for the inputs



- b.
  - c. As you can see it's like paint so just edit what it looks like and says
10. Now you can insert your custom vi into the code
11. Now just plug the angle in and you will get your corrected value