Team Lucid

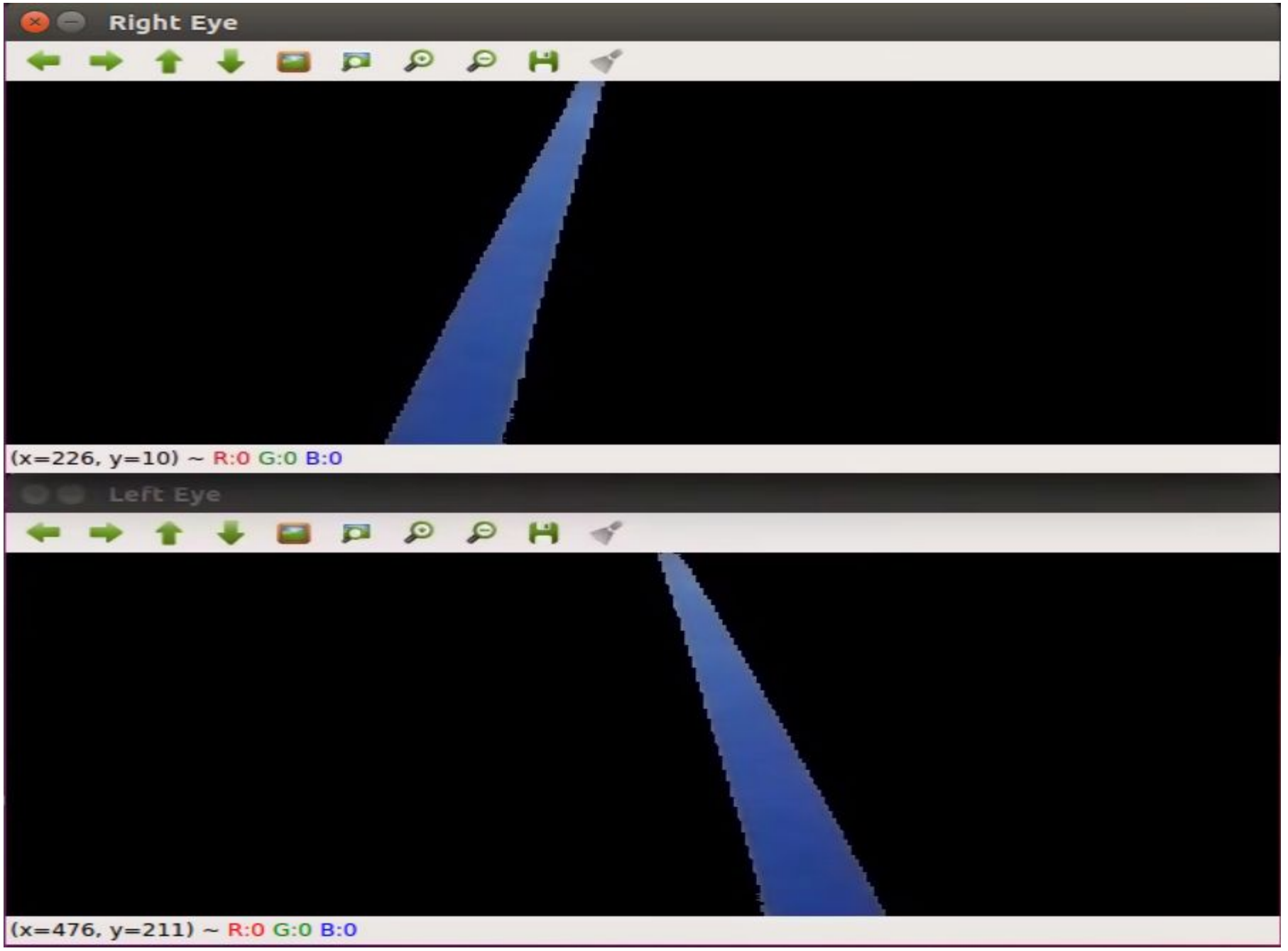
>>Renegade

Challenge One – Line Detection

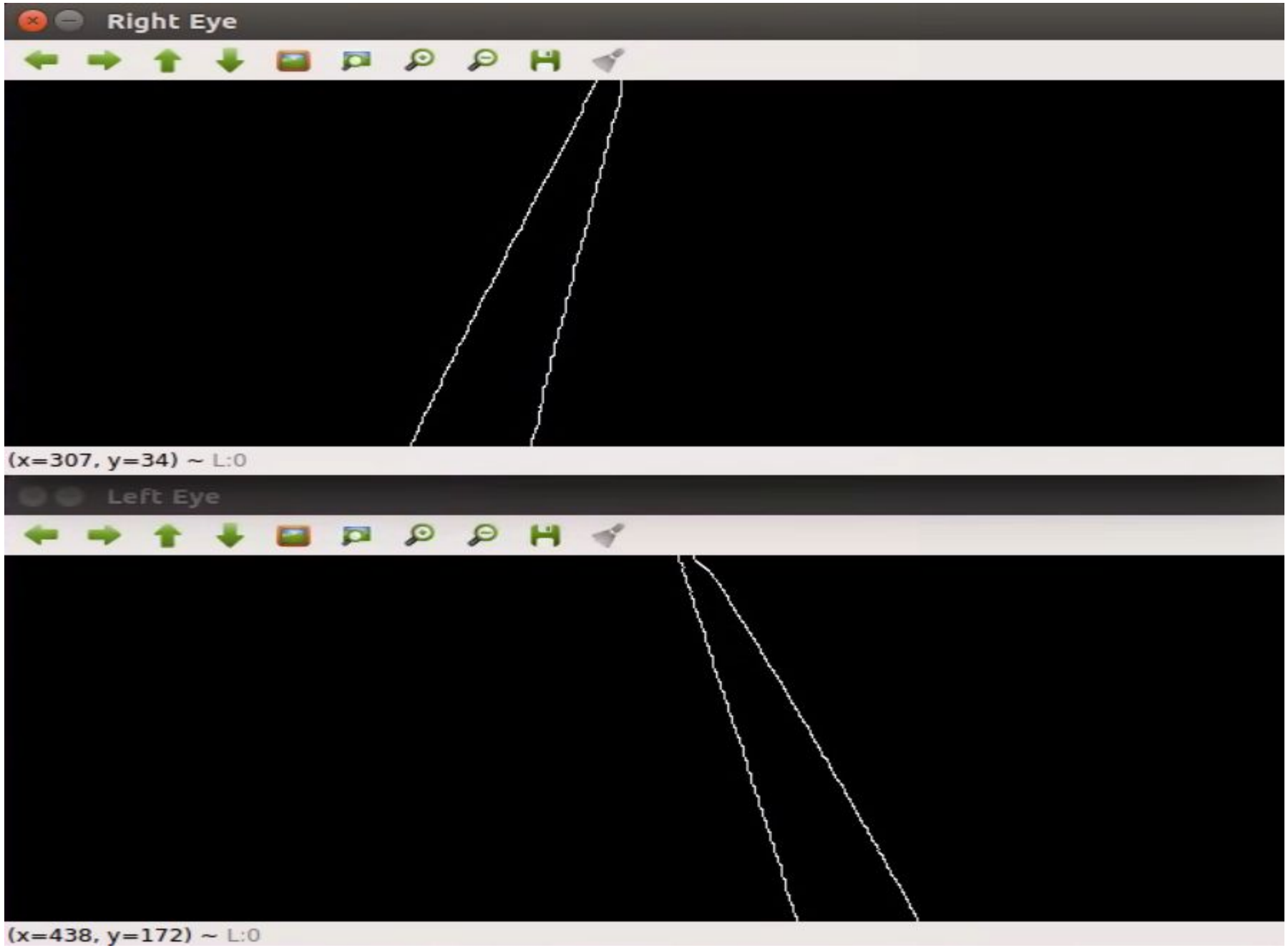
Designed By:

- Justin McGettigan (Team Lead, Lead Developer)
- David Ciccarello (Hardware integration, Programmer)
- Alex Olinger (Programmer, Debugging)

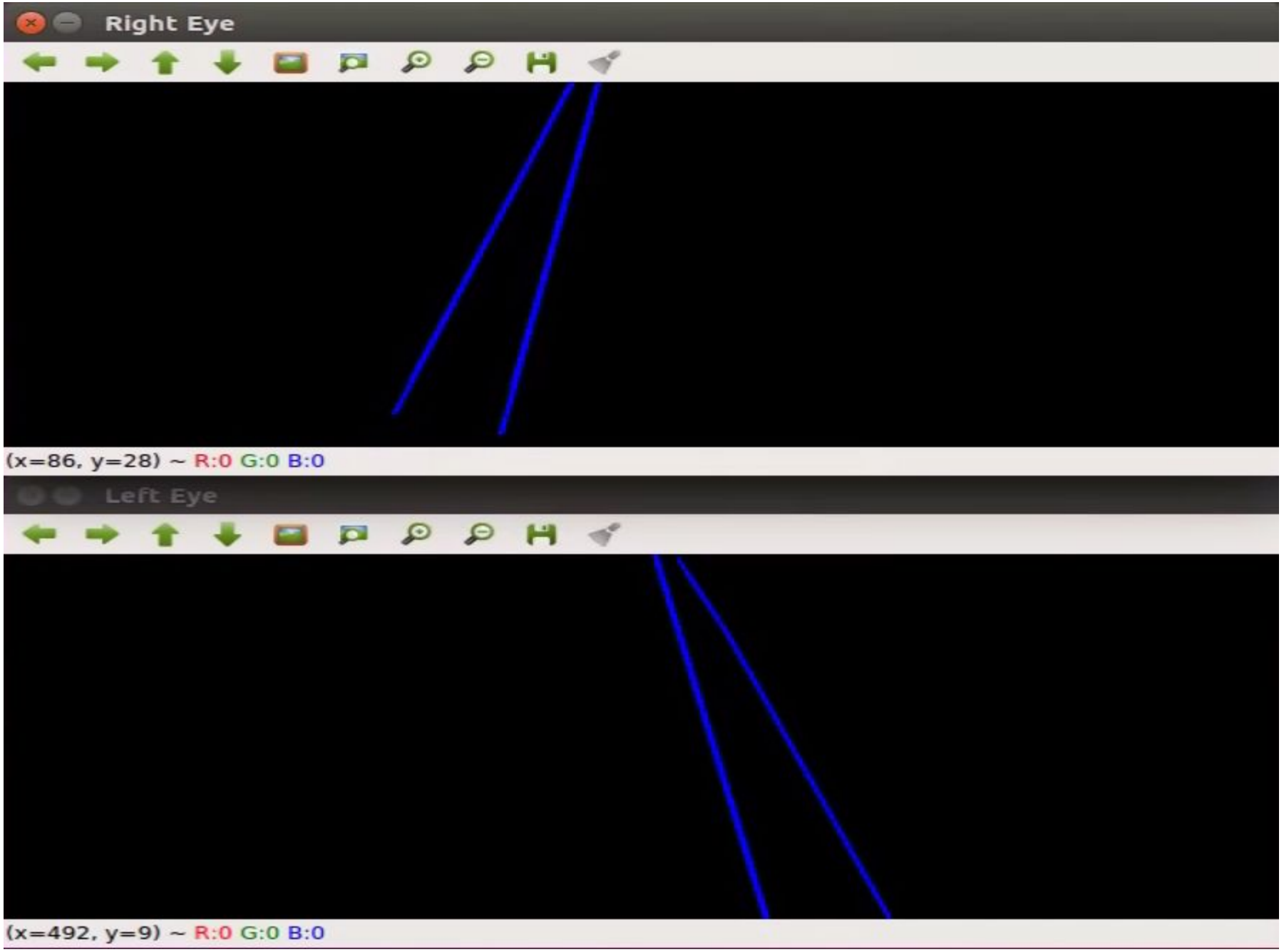
- Color Masking
- Isolate a defined color (blue) from the ZED’s left and right eyes



- Canny Edge
- Detect edges of the color masked region



- Hough lines
- Draw the lines of the canny edges using hough space



- The red lines are the slopes that we are adding together to find our error.
- The y values in the slope are constant: the top & bottom of the screen.
- One x value is constant in the center of both eyes. The other x is always at the destination of the line.

Controller

- Our Error value is the sum of the slopes of the average slope of the huff lines for each individual eye. Here, it is **ZEDvalue**.
- For challenge 1, we are only using the current error to determine the steering value.
- This script allows us to take the previous history into account using a history array if we want.

```
global turningValues
turningValues = [0]
HISTORYSIZE = 10


def turn_control(ZEDvalue):
        length = len(turningValues)
        if (length > HISTORYSIZE-1): #if list is larger than the desired history size -1, remove item at index 
                turningValues.pop(0)

        #turningValues.append(ZEDvalue)
        turningValues.append(ZEDvalue) #adds new ZEDvalue to last position in array

#       print turningValues #debug, shows all values in the list
        P = I = 0
        for x in turningValues:
                I += x

        #angleControlPyUsesToTurn
        const = (0.3/2.5)
        P = const * turningValues[length-1]
        I = const * (I/length) #averages values

        Kp = 1#0.9
        Ki = 0#1 - Kp
        output = (Kp * P) + (Ki * I)
        return output
```

Speed Control

- Using the Sum of the slopes we added a weight of 0.12 to make the range of values usable for steering.
  -This is found in the Controller.py script

```
def controller():
    global leftSlope, rightSlope
    print "================="
    print "(Left|Right): (" + str(rightSlope) + "|" + str(leftSlope) + ")"
    print "Sum: " + str(leftSlope + rightSlope)
    control = con.turn_control(leftSlope + rightSlope)
    if control > 0.3:
        control = 0.3
    if control < -0.3:
        control = -0.3
    direction = ""
    if control > 0.02:
        direction = "Left"
    elif control < -0.02:
        direction = "Right"
    else:
        direction = "Center"
    print "(Control|Direction): (" + str(control) + "|" + str(direction) + ")"
    speed_limit = 0.6
    speed_control = speed_limit * (1 - abs(control))**1.13678
    #speed_control = 0.4
    if f.getLinesExist:
        apply_control(speed_control, control)
    #else:
        #stop()
        #apply_control(1, 0)
```

Special Functionality

- When no lines are viewed. Use the last set of hough lines that the ZED captured and repeat

```
global linesExist, lastHoughLinesLeft, lastHoughLinesRight, stop
if houghLines is not None:
    stop = False
    linesExist = True
    if side == 'left':
        lastHoughLinesLeft = houghLines
    if side == 'right':
        lastHoughLinesRight = houghLines
    return houghLines
elif lastHoughLinesLeft is not None and lastHoughLinesRight is not None:
    #print "Couldn't find lines."
    print "Using last instance of houghLines as reference."
    stop = False
    linesExist = False
    if side == 'left':
        return lastHoughLinesLeft
    if side == 'right':
        return lastHoughLinesRight
else:
    stop = True
    linesExist = False
```

Lessons learned

- Having a greater understanding of hough lines and image manipulation.
- Learned how to publish variable speed and turning values using a PID controller.
- MAKE SURE CAR IS SECURE BEFORE STARTING SCRIPTS!