CS 378 - Autonomous Vehicles in Traffic I

Week 9a - Debugging
Programming Assignment 4

- There was a bit of delay in programming assignment 4. The assignment is now due 3/28
  - Programming Assignment 4
  - Assignment 5 is still due 4/4
  - OpenCV is not the easiest library to use. Search for solutions to any problems you might be facing online.
  - Sometimes a solution may be given using the C API. It is easy enough to use the corresponding C++ functions.
  - Let's take a look at one such example problem next frame
cv::cvtColor

- If you use edge detection, you will have to convert the image into grayscale
- Open up the [OpenCV C++ reference](http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html)
- Type in the function in the C API, by removing the first `cv`  
  - For instance search for `cvtColor` instead of `cvCvtColor`
  - Take a look at the documentation and adapt the code you find online

- I don't have any experience with the Python reference, you might have to work a bit hard to find solutions using it.
OpenCV - helpful links

- I found this list of tutorials very recently:
  - These tutorials use the new OpenCV C++ API, and should be extremely useful

- In particular these tutorials should be useful:
  - `cv::Mat` The basic image container
    - [http://opencv.itseez.com/doc/tutorials/core/mat%20-%20the%20basic%20image%20container/mat%20-%20the%20basic%20image%20container.html](http://opencv.itseez.com/doc/tutorials/core/mat%20-%20the%20basic%20image%20container/mat%20-%20the%20basic%20image%20container.html)
  - If you plan on using edge detection
    - [http://opencv.itseez.com/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html](http://opencv.itseez.com/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html)
Projects

• I will start replying back to your initial emails today.
• The goal of the research proposal on Wednesday is for you to:
  ◦ form a group
  ◦ bring the list of the projects you are interested in down to 1 or 2
• We'll try to finalize some basic groups at the end of Wednesday's class
• Is there anyone who has formed a group already, but has not emailed me yet?
• By the end of Wednesday, send one email per group to cs378-spr12-submit with your group members, and the projects you are interested in.
• More information is better!
Debugging

• Hopefully while doing the assignments, all of you have now gotten a bit of experience handling compile time issues.

• In reality, debugging is the main thing that can make C++ particularly painful to use.

• Unlike other languages such as Python or Java, C++ code runs directly on the machine.
  ◦ There is nothing to catch an error when something goes wrong (and things can go horribly wrong)
  ◦ At the same time, it allows for your code to run a lot more efficiently
Debugging

• There are 2 tools that are popular for debugging C++ code
  ○ *gdb*
  ○ *valgrind*

• In class today, we'll go through *gdb*.
  ○ For the most part, *gdb* should suffice for most of your debugging needs.
  ○ I do occasionally use *valgrind*, when *gdb* is unable to solve the problem. Additionally, *valgrind* can help detect memory problems with your program.

• A overview of coding practices with regards to debugging is available below. I think this needs some prior experience.
  ○ [http://www.gamedev.net/page/resources/_/technical/general-programming/c-debugging-r1344](http://www.gamedev.net/page/resources/_/technical/general-programming/c-debugging-r1344)
Examples

- We will see 2 examples in today's class
- Example from today's lectures have been taken from here
  - [http://www.tutorialspoint.com/gnu_debugger/gdb_debugging_examples.htm](http://www.tutorialspoint.com/gnu_debugger/gdb_debugging_examples.htm)
- I checked them in a new package called `debugging` inside the `art_examples` stack. You can do use `svn up` to get them.
#include <iostream>

int divint(int, int);

int main() {
    int x = 5, y = 2;
    std::cout << divint(x, y);
    x = 3; y = 0;
    std::cout << divint(x, y);
    return 0;
}

int divint(int a, int b) {
    return a / b;
}


```cpp
#include <iostream>

void setint(int*, int);

int main() {
    int a;
    setint(&a, 10);
    std::cout << a << std::endl;

    int* b;
    setint(b, 10);
    std::cout << *b << std::endl;
    return 0;
}

void setint(int* ip, int i) {
    *ip = i;
}
```
Debugging - g++

• Let's assume you are writing a plain old C++ program outside the ROS ecosystem.

• To enable debugging, you run the following command to compile your code:
  ○ `g++ -g -O0 -o example1 example1.cpp`

• The following flags are useful while debugging:
  ○ `-g` - Produces debugging information
  ○ `-O0` - disables optimizations (used by default)
  ○ `-Wall` - enable a number of warnings that you should not produce in your program

• You can now run `gdb` using the following command:
  ○ `gdb example1`
Debugging - ROS

- A few things are different when you are inside the ROS Ecosystem
  - The ROS CMake script enables `-Wall` by default
  - You can use set the `ROS_BUILD_TYPE` to `Debug` in the `CMakeLists.txt` to get the `-g -O0` flags.
  - Remember to run the following command to recompile your code:
    - `make clean && make`
  - You can then run gdb using the following command:
    - `gdb bin/example1`
  - Once you are done debugging, remember to change the flag back to `Release`, or comment out the line.
Important gdb commands

• `run <program flags here>`
• `list`
• `backtrace OR where`
• `inspect OR print`
• `break`
• `next`
• `step`
• `quit`

A bit of explanation is here:
• [http://www.tutorialspoint.com/gnu_debugger/gdb_debugging_example1.htm](http://www.tutorialspoint.com/gnu_debugger/gdb_debugging_example1.htm)
• [http://www.tutorialspoint.com/gnu_debugger/gdb_debugging_example2.htm](http://www.tutorialspoint.com/gnu_debugger/gdb_debugging_example2.htm)