# CS492 - The Santa Maria Project

# Introduction

An inherent part of the 2010 JanPlan Columbus project was making the proof-of-concept application that would *just work*. The goal was achieved, and a great software package (at last compiled for ARM processors) was prepared. With small changes, as the world's first robot controller running on iPod, it was even featured on the Colby College news feed, as one of the recent student achievements.
However, the implementation of Columbus was done very much at the whim. Substantial parts of the IPC library (or its event handling), used typically in *nav-devel* were axed to make coding as easy as possible. No more. The goal of CS492 was to create a grown-up platform, on which IPC/Mage-based software can be easily implemented. Name it - a wrapper, a thin client, an external library. It is Santa Maria - a 300 kilobytes of goodness, that allows all iPhones, iPads and Macs in the world rejoice and implement *snmd* natively, as a normal class supplemented by a library. Furthermore, a simple application, Easy Robot, leveraging this library was created - thus allowing to explore the way certain constructs and concepts are implemented.
A set of guidelines and snippets for the future Colby iPhone and iPad programmers was also prepared, in total enveloping whereto the Columbus/Santa Maria ecosystem should grow, and what technologies and concepts it should heed.

# Assumptions

It is expected, that the readers of this document are familiar with C and IPC library. While no other experience is strictly required, some reading on Objective-C and Cocoa before browsing the code is advised, as the conceptual differences of Objective-C, as well as Cocoa's dominating presence in the code are fairly intimidating at first. The development Mac should be running at least Snow Leopard, as SDK for iPhone OS 3.2 or 4.0 is required to build iPad apps.

# Download

The library and Easy Robot can be downloaded from a github repository as a complete XCode project, with all the dependencies. Readers are more than welcome to fork the project and send the pull requests.

# The Santa Maria Project

## Robot Library

### Library construction

The Robot is the main part of this CS492 project. Set up as an XCode target, Robot compiles down into a set of two files - `libRobot.a` and `Robot.h`. The first one is a multi-platform library (at the moment supporting i386, x86_64 and ARMv6) that contains also IPC and GCM libraries, while the other one is a header for the library. Since IPC and GCM generally do not have to be recompiled, the package contains their multi-platform binaries, that can be later embedded in the Robot library. To run Robot, needed are also files `ipc.h`, `GCM.h`, `GCM_IPC.h`, and `ppmIO.h`, as they are linked from `Robot.h`. `Robot.h` itself consists of 5 pragma sections - for initializing and connecting, for handling properties, for video support, and for motion management.

### How to use it?

The Robot was created as an NSObject derived class, that represents one real life robot. To create an instance of the robot, one has to attach the library to the project and import the headers. Later, a robotic entity can be brought to life by simply typing:

```
Robot* robot = [[Robot alloc] init];
```

The standard `init` presumes that the connection to `localhost`, and there is an additional method for initializing to any given URL, `initWithURL:[NSURL URLWithString:@"hereIsURL"]`.

After the `robot` class instance has been created, one can connect to the robot at the URL simply by typing:

```
[ robot connectToTheRobot ];
```

Significantly easier than the standard IPC. Similarly, `[ robot setUpCamera ]` is enough to carry out all the setup of camera system on the robot (if changes are necessary, there is an additional method, while `[ robot startStreamingVideo:display ]` is a one-liner that will provide a constant video stream in a separate thread to the UIImageView defined as display.

The remainder of options and methods is similar in the concept - wrapping up large IPC calls and data structures in simple Objective-C structures (like NSMutableArray for the waypoints), while making the client coding much, much easier.

### Why bother?

Good question. IPC is a relatively old-school library, that was not written with Objective-C in mind. While one important concept of Objective-C is a strict subsetness of the language, by which all C code works with Obj-C code, Objective-C is largely complemented by Cocoa, which is based on the rule *as little C as possible*, wrapping all possible calls and reference in its own special wrappers.
Handling mixed C and Obj-C code is a big pain. The data structures are very often incompatible (or require huge invocations from the user to convert them safely), and by default C functions cannot call on Objective-C class methods. The last problem is a significant issue for IPC, as many of the function calls in IPC are actually executed directly from pointers to C functions. As such, it was necessary to create a closely connected and automatically synchronizing network of settings, in which Objective-C and C can interface together.
Another issue, is that `snmd-control` was built with Linux/UNIX systems in thought. The `dispipc` is an XWindow application, while the entire communication occurs over the standard UNIX sockets. Cocoa's support for these two is very limited. Because of that, the entire video reception support for Robot was rewritten from scratch to support fast CoreGraphics rendering, multithreading (to not hamper the main loop during prolonged wait periods) and take as little resources as possible, especially by taking great care of garbage collection. With medium settings, on the Transatlantic connection from Poland, `Easy Robot` was able to render about 4-5 fps, what is very promising in terms of speed of the transmission. Also, due to improved garbage collection, even if a Robot-based app is running for many hours, its memory footprint will remain tiny - three hours of transmission from the lab effected in a little more than 15 megabytes of real memory being occupied by the app. In comparison, Columbus would crash after a transmission longer than about 120-190 seconds.
Furthermore, with the new library, creating a fairly advanced robot control app becomes a plaything in Interface Builder and can be implemented with less than 50 lines of code. Such control app is more powerful than Columbus was and is about 10 times shorter. Also, expanding Robot library is easy, as it basically covers up the entire IPC aspect beautifully wherever possible.

## Easy Robot App

Easy Robot App is a basic robot controller built upon the `snmd-control` port supposed to be used in the Colby College Museum in mid-May. The entire app is implemented only using The Robot calls and presents some of the notions that should be heeded in the future development of Columbus and Santa Maria.

# Quo vadis?

## To implement in The Robot soon

Important missing feature is Cocoa-condoned error throwing. Right now, majority of errors is signified by either modified Boolean variables, inactivity or invalid return values. All of this can be caught using Objective-C `@try-@catch-@else-@finally` construct. Furthermore, the application will not quit or crash on any error other than running out of memory or calling the function the wrong way, what improves the general stability.
Furthermore, Apple recommends if the application contains the network status and reachability check. It is trivial to implement as a part of NSURL module, but I did not add this due to the lack of usability in this situation.
The setters and getters need to be confirmed as functional throughout the function. Basically, it is entirely possible to set all the needed values from within the class, but some of them are protected according to the basic usage. This might change with versions.

## To implement in The Robot/Columbus soon

The Robot and Columbus are missing out on a planned functionality of Finger Mapper. The Finger Mapper became an answer to the last year's

research problem in constructing the optimal path through the waypoints. The way to handle it this time would be to overlay the waypoints over the laser map obtained by the device and ask the user to slide the finger in an unbroken path through all the marked dots. Since the movement would be smooth, one could extrapolate the total path of the finger and take the robot over it, making basically the Finger Mapper a touchpad, and robot a mouse cursor, if a simile is needed.
Furthermore, the Robot/Columbus does not support mapping waypoints on a canvas yet (while it supports intake of waypoints in a n2 NSArray, where n is the number of waypoints and xy is the coordinates of the waypoints). However, all technical code (waypoint import and robot localization) are there already, so this functionality should be easy to build.

## Santa Maria/Columbus guidelines

1. The app should provide visual feedback on the connection.
    a. A top bar with the title indicating the host is advised.
    b. The property `setPrompt:prompt` is to be used to inform the user about the ongoing activity, i.e. connecting to the server or replacing data chunks.
    c. Network activity (transmissions) should be indicated by a spinning wheel, implemented using `[UIApplication sharedApplication].networkActivityIndicatorVisible=YES;`
2. The user should be notified about the issues with the robot and application.
    a. Application should notify the user about both low memory and impossibility to connect or obtain the video feed.
    b. Application should notify the user if the brakes/velocitySpace are acting as limiting factors (easy by comparing the x and y values)
3. The application should follow Apple's Human Interface Guidelines:
    a. The app should use the concepts of tabbed interface as rarely as possible. Split buttons are OK, tabs (but not views!) are not.
    b. The video should be relocated to the bottom left corner of a typical split-view application and be presented on a separate layer on request, vide most of the prompted actions on iPhone OS for iPads
    c. The app should be able to function in both horizontal and vertical orientation and provide different interfaces for different functionalities.

# Conclusions

While I did not manage to take the project as far as I planned this term, (Burn. Last time I overachieved in comparison with the expectations) I still produced a decent, well-thought library that is a substantial and fundamental for any future development of iPod/iPad - basically touch - Robot Controllers at Colby. Some of my work chunks, featured to Apple earned me a scholarship to attend WWDC this year on a student scholarship, so I guess the work could not have been too shabby. I plan on continuing the development actively, as quickly as possible - I have recently received a proposition to showcase it to Apple students and professors at WWDC.

Objective-C and Cocoa make for a great developer ecosystem. Frustrating at times. Full of features, but also full of concepts understandable only to Apple engineers or people with eons of experience in Apple development. (hi, Bruce!) This library finally allows to wrap everything in an easy, understandable platform, that can be a stepping stone to phenomenal projects down the line. I am very excited to see how it develops over the summer and during my year-long absence and I am quite certain it will be something awesome. 

# Acknowledgements

A lot of thanks goes to Professor Maxwell, for his unending assistance and Q&A sessions during which a lot of solutions or concepts was brainstormed. Definitely a word of thank you is due to Professor Eastwood, for his help in conceptualizing math behind the Finger Mapper. The Apple Developer websites were indispensable, as well as books titled "Unleashing XCode", "Programming in Objective-C 2.0", "iPhone Programming: The Big Nerd Ranch Guide", "Cocoa(R) Programming for Mac(R) OS X (3rd Edition)" and countless other websites, among them these of Jeff LaMarche, Wil Shipley and Scott Stevenson, the first of whom I hope to meet for lunch on WWDC :]
All in all, this was a big project that involved a lot of Odwallas, Google and indie rock. So 'hi' to these three as well.

kthxbai. see you in 2011.