Document on MSC with AR-Drone 2

Zhangyuan Wang

05/18/2016

This document introduces how to set up the environment to run MSC on AR-Drone 2 on Windows and other platform in general. A brief summary of previous work is presented. And the work to be done in the future is discussed.

Set up on Windows

A. Set up Map Seeking Circuit

This part is mostly done by the Visual Studio configuration. All the compiled opencv file are included in the github repository.

1. Install Visual Studio 2015 Community.
2. Git clone from https://github.com/nnnrpq/msc
3. Add ***MSC_ROOT/msc/opencv_2411/bin/Debug*** and ***MSC_ROOT/msc/opencv_2411/bin/Release*** to system environment.
4. Open msc.sln. It should be ready to use in Visual Studio, both in debug mode and release mode. F5 to build and start.

B. Set up node-ar-drone

We need to manually setup this part, since the packages needs to be installed in the correct place.

1. Install **nodejs** (current version v4.4.5)

2. cd to ***MSC_ROOT/msc***

3. Install ar-drone through nodejs:

   npm install git://github.com/felixge/node-ar-drone.git

4. Install node-arraybuffer and nan, to convert nodejs object to void* in C.

   npm install node-arraybuffer –save

   npm install nan –save

5. Install node-gyp, native addon build tool for nodejs. Do as the github says:
   a) Install ***npm install -g node-gyp***
   b) Install python 2.7 and run `npm config set python python2.7`
   c) Launch cmd, ***npm config set msvs_version 2015***

6. Create (binding) addon project for VS

Since we need the packages in nodejs and still want to configure the whole project in VS2015, we first create a "hello" project with node-gyp and change the files and settings of the created project.

In ***MSC_ROOT/msc,*** there's already a binding.gyp and hello.cc. Run ***node-gyp configure install*** and node-gyp will create a VS project in the ***MSC_ROOT/msc/build*** directory.

7. Add files and configure the project

Open **binding.sln** or **MSC_drone.vcxproj**. Change to Release mode. Add the source files from ***MSC_ROOT/msc/src*** as in the previous MSC project. Delete **hello.cc** from project and **jsmsc.cc** instead**.**

Edit the properties of project. Open View-Others-Property Manager and in the Release part, add existing property sheet **opencv2411r.props.** This should load the existing property page. If not, follow any tutorial online to set up OpenCV environment and project property and add the source code (***MSC_ROOT/msc/src***) of the MSC project.

I'm not sure why this happens, but if it cannot find **Username\.node-gyp\4.4.4\Release,** copy the folder ***MSC_ROOT*\.node-gyp\4.4.2\Release** to the place. Also, you might need to add the folder of **..\ pthreads-w32-2-9-1-release\lib** to **Project – Properties – linker – general – additional library directories**.

Build the solution, which produces in a **MSC_drone.node** file for nodejs.

After configuring the **binding.sln** with the source code in ***MSC_ROOT/msc/src*** for the first time, you should be able to use the project in the future versions, since the ***MSC_ROOT/msc/build*** directory is already excluded from the github repository and will not update.

8. Call the node from nodejs and fly the drone

Get the drone ready. Connect to its wifi from PC.

Go to ***MSC_ROOT/msc/js.*** We need to use binary library **ffmpeg** to get the image stream.

Open **ff-prompt.bat.** Go back to the upper directory (cd ..). Type **node jstest.js**, which will run the js script in node.

Demo done!

General method to set up

A. Set up Map Seeking Circuit

Generally, to set up the MSC is to set up an OpenCV project. The source code is in ***MSC_ROOT/msc/src.*** You can find what is used in msc.vcxproj file. Note that when configuring Linux based platform, pthread package for windows (**pthreads-w32-2-9-1-release**) is no longer needed.

It might be possible to convert the VS project directly to makefile. Here's a hint of what can be used.

B. Set up node-ar-drone

Configure the interface to the drone should be familiar. Just follow the corresponding instructions for the platform you need. The method is inspired by the tutorial on this website.

1. Install **nodejs** (current version v4.4.5)

2. Install ar-drone through nodejs:

npm install git://github.com/felixge/node-ar-drone.git

3. Install node-arraybuffer and nan, to convert nodejs object to void* in C.

npm install node-arraybuffer –save

npm install nan –save

4. Install node-gyp. This part is different for Unix or Max OS. Follow the instruction on GitHub.

5. Set up the MSC makefile with node dependency enabled.

One way is to follow the same procedure as in VS, i.e. do the binding with a "hello world" code and modify the makefile.

To bind the hello.cc in Unix, the same command ***node-gyp configure install*** is used. This will result in a Makefile for your platform. Refer to the nodejs document about addon for more.

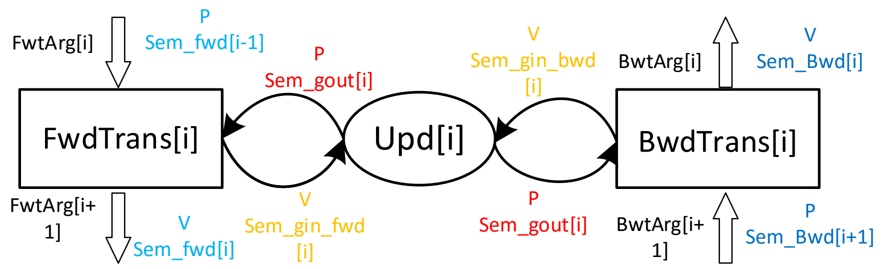
6. Download **ffmepg.** Choose the appropriate version and run **node** with **ffmepg**, call the js file to fly the drone.

Resources for drone

1. A step by step tutorial for flying the drone. http://www.instructables.com/id/Autonomous-AR-Parrot-Drone-20-Flying/
2. The git repository of the nodejs interface, with a brief summary of available functions. https://github.com/felixge/node-ar-drone
3. The official document for nodejs. In exact, the addon part, which bind C/C++ code to produce node module. https://nodejs.org/dist/latest-v4.x/docs/api/addons.html.
4. A more advanced introduction to build C/C++ addon for nodejs. https://blog.scottfrees.com/c-processing-from-node-js.
5. The official document of v8, which could help to write the C/C++ addon. https://v8docs.nodesource.com/node-4.2/inherits.html.

Work summary

1. Clean up the "vanilla code"
2. **Complete a test bench**, which apply random affine transformation to image.
3. Explore the tricks to **update competition**. One necessary element is **scaling factor**. Currently *countnonzero(dst)/countnonzero(src)* is used as scaling factor to balance the effect of scale up and scale down. Also, **boosting the k value** is important for speeding up the convergence. Other tried but not effective enough tricks include tuning the g update with power (1-q/qmax)^p, and preventing the rotation and scaling layer from updating when the translation layer hasn't converged.
4. Exploit **frame by frame continuity** to refine the selection of parameters. Currently in the case of continuous frame, 5 parameters centered around the previous solution reaches a balance between accuracy and speed.
5. Explore possible ways to **speed up transformation**. One seemingly promising approach is to calculate transformation mapping by hand in the first place, store it and do the mapping without computing each time. This should **save the redundant time of calculating mapping of pixels**. However, due to lack of optimization, i.e. not able to effectively do the parallel mapping, this cannot beat the **built-in warpaffine function**.
6. Run profiling of the code, and **optimize the memory allocation and de-allocation**. For example, use fixed length array, avoid "push_back", recycle the chunk of memory and pass the same memory block through each layer instead of assigning new memory. Also, modify code to **prevent memory leakage** caused by e.g. the vector<Mat> variables. After this, **the code is running at around 300ms per frame.**
7. Try **parallel**. Rewrite the function forward/backward transformation and update competition to accommodate pthread. The first experiment of launching both the forward and backward at the same time fails to speed up the code. Now I'm trying to work out the full pipeline design on layer level.



Update: it seems that we don't need such a complicated parallel structure, because the data flow is known. In each iteration (after the g is updated or initialized), forward/backward transformation is done layer by layer. This is sequential, except that the two direction are independent. Then, updating the g value must be done before another iteration of transformation. This should be treated as parallel.

8. **Set up the drone**. Do a demo with ar-drone package on nodejs. Figure out the way to run MSC code with drone and complete the demo with a simple control code.
9. **Clean up the code, pack it, make it easier to configure on Windows through Github, and do the documentation**.

Future work

1. Set up on Linux.
2. Experiment different methods of **pre-processing**, to adapt to the current task.
3. Tune the layer-wise parallel. If necessary, try CUDA.