Part A

Montreal IFT6757 2020 Project Reports: Final Reports

Contents

UNIT A-1

Group name: Project report

Before starting, you should have a look at some tips on how to write beautiful Duckiebook pages.

The objective of this report is to bring justice to your hard work during the semester and make so that future generations of Duckietown students may take full advantage of it. Some of the sections of this report are repetitions from the preliminary design document (PDD) and intermediate report you have given.

1.1. The final result

Let's start from a teaser.

- Post a video of your best results (e.g., your demo video): remember to have duckies on the robots or something terrible might happen!

You might want to add as a caption a link to your instructions to reproduce to reproduce these results. Moreover, add a link to the readme.txt of your code.

1.2. Mission and Scope

Now tell your story:

Define what is your mission here.

1) Motivation

Now step back and tell us how you got to that mission.

- What are we talking about? [Brief introduction / problem in general terms]
- Why is it important? [Relevance]

2) Existing solution

- Describe the "prior work"

3) Opportunity

- What was wrong with the baseline / prior work / existing solution? Why did it need

improvement?

Examples: - there wasn't a previous implementation - the previous performance, evaluated according to some specific metrics, was not satisfactory - it was not robust / reliable - somebody told me to do so (/s) (this is a terrible motivation. In general, never ever ever say "somebody told me to do it" or "everybody does like this")

- How did you go about improving the existing solution / approaching the problem? [contribution]

Examples: - We used method / algorithm xyz to fix the gap in knowledge (don't go in the details here) - Make sure to reference papers you used / took inspiration from, lessons, textbooks, third party projects and any other resource you took advantage of (check here how to add citations in this document). Even in your code, make sure you are giving credit in the comments to original authors if you are reusing some components.

1.3. Background and Preliminaries

- Is there some particular theorem / "mathy" thing you require your readers to know before delving in the actual problem? Briefly explain it and links for more detailed explanations here.

Definition of link: - could be the reference to a paper / textbook - (bonus points) it is best if it is a link to Duckiebook chapter (in the dedicated "Preliminaries" section)

1.4. Definition of the problem

Up to now it was all fun and giggles. This is the most important part of your report: a crisp, possibly mathematical, definition of the problem you tackled. You can use part of the preliminary design document to fill this section.

Make sure you include your: - final objective / goal - assumptions made - quantitative performance metrics to judge the achievement of the goal

1.5. Contribution / Added functionality

Describe here, in technical detail, what you have done. Make sure you include: - a theoretical description of the algorithm(s) you implemented - logical architecture - software architecture - details on the actual implementation where relevant (how does the implementation differ from the theory?) - any infrastructure you had to develop in order to implement your algorithm - If you have collected a number of logs, add link to

where you stored them

Feel free to create subsections when useful to ease the flow

1.6. Formal performance evaluation / Results

Be rigorous!

- For each of the tasks you defined in you problem formulation, provide quantitative results (i.e., the evaluation of the previously introduced performance metrics)
- Compare your results to the success targets. Explain successes or failures.
- Compare your results to the "state of the art" / previous implementation where relevant. Explain failure / success.
- Include an explanation / discussion of the results. Where things (as / better than / worst than) you expected? What were the biggest challenges?

1.7. Future avenues of development

Is there something you think still needs to be done or could be improved? List it here, and be specific!

UNIT A-2

Instructions template

Before starting, you should have a look at some tips on how to write beautiful Duckiebook pages.

This is the template for the description of a what we should do to reproduce the results you got in your project. The spirit of this document is to be an operation manual, i.e., a straightforward, unambiguous recipe for reproducing the results of a specific behavior or set of behaviors.

It starts with the "knowledge box" that provides a crisp description of the border conditions needed:

- Duckiebot hardware configuration (see Duckiebot configurations)
- Duckietown hardware configuration (loops, intersections, robotarium, etc.)
- Number of Duckiebots
- Duckiebot setup steps

For example:

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Duckiebot in configuration `DB19`

Requires: Duckietown without intersections

Requires: Camera calibration completed

2.1. Video of expected results

First, we show a video of the expected behavior (if the demo is successful).

Make sure the video is compliant with Duckietown, i.e. : the city meets the appearance specifications and the Duckiebots have duckies on board.

2.2. Laptop setup notes

Does the user need to do anything to modify their local laptop configuration?

2.3. Duckietown setup notes

Here, describe the assumptions about the Duckietown, including:

- Layout (tiles types)
- Infrastructure (traffic lights, WiFi networks, ...) required
- Weather (lights, ...)

Do not write instructions on how to build the city here, unless you are doing something very particular that is not in the Duckietown operation manual (unknown ref opmanual_duckietown/duckietowns)

warning next (1 of 8) index

warning

```
I will ignore this because it is an external link.

 > I do not know what is indicated by the link '#op-
manual_duckietown/duckietowns'.
```

Location not known more precisely.

Created by function `n/a` in module `n/a`.

. Here, merely point to them.

2.4. Duckiebot setup notes

Write here any special setup for the Duckiebot, if needed.

Do not repeat instructions here that are already included in the Duckiebot operation manual (unknown ref opmanual_duckiebot/opmanual_duckiebot)

previous warning next (2 of 8) index

warning

```
I will ignore this because it is an external link.

 > I do not know what is indicated by the link '#op-
manual_duckiebot/opmanual_duckiebot'.
```

Location not known more precisely.

Created by function `n/a` in module `n/a`.

.

2.5. Pre-flight checklist

The pre-flight checklist describes the steps that are sufficient to ensure that the demo will be correct:

Check: operation 1 done

Check: operation 2 done

2.6. Instructions

Here, give step by step instructions to reproduce the demo.

Step 1: XXX

task next (1 of 7) index

task

The following was marked as "status-XXX".

> Step 1: XXX

Location not known more precisely.

Created by function `n/a` in module `n/a`.

Step 2: XXX

previous **task** next (2 of 7) index

task

The following was marked as "status-XXX".

> Step 2: XXX

Location not known more precisely.

Created by function `n/a` in module `n/a`.

Make sure you are specifying where to write each line of code that needs to be executed, and what should the expected outcome be. If there are typical pitfalls / errors you experienced, point to the next section for troubleshooting.

2.7. Troubleshooting

Add here any troubleshooting / tips and tricks required, in the form:

> Symptom: The Duckiebot flies

Resolution: Unplug the battery and send an email to info@duckietown.org

> Symptom: I run `this elegant snippet of code` and get this error: `a nasty line of gibberish`

Resolution: Power cycle until it works.

2.8. Demo failure demonstration

Finally, put here video of how the demo can fail, when the assumptions are not respected.

You can upload the videos to the Duckietown Vimeo account and link them here.

UNIT A-3

Group name: Project report

3.1. The final result

Let's start from a teaser.

- Post a video of your best results (e.g., your demo video): remember to have duckies on the robots or something terrible might happen!

You might want to add as a caption a link to your instructions to reproduce to reproduce these results. Moreover, add a link to the readme.txt of your code.

3.2. Mission and Scope

Our mission is to create a controller that is capable of using images from the duckiebot's camera to follow another duckiebot and stop a certain distance away, in line with its rear. We would like to implement this controller in the duckiebot and also a similar controller that would be able to only stop behind another duckiebot using the simulator.

Our duckiebot has a camera with fisheye lens installed on it. This configuration is called eye-in-hand because it is rigidly mounted on the robot. Our target is at a certain distance of the bumper of other duckiebot. The bumper has a circle's pattern: 8 columns by 3 rows on simulator and 7 columns by 3 rows on the duckiebot.

3.3. Motivation

This project is a visual servo controller that is a technique of using visual sensors to control the movements of a robot.

This project is of great importance as it has many applications. Visual servoing can be used to pick up an object from one place and place it in another, or to follow and track an object by an autonomous vehicle.

3.4. Existing solution

Visual servoing is the use of visual sensors in order to control the motion of the robot. There are three main approach to tackle this problem. One is called positional-based visual servo (PBVS), another is called Image-based visual servo (IBVS) and the third one is an hybrid that combines the two first. For more information about both methods [1] and a comparison in results [2]

1) S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," IEEE Transactions on Robotics and Automation, vol. 12, no. 5, pp. 651–670, 1996.

2) G. Palmieri, M. Palpacelli, M. Battistelli, and M. Callegari, "A Comparison between Position-Based and Image-Based Dynamic Visual Servoings in the Control of a Translating Parallel Manipulator"

1) Positional-based visual servo (PBVS)

The positional-based visual servo control the task is defined in 3D Cartesian frame. The visual data tries to reconstruct the 3D pose of the camera and the kinematic error is generated in Cartesian space.

In order to perform a PBVS, the follow steps is necessary:

1) 3D camera calibration: - Intrinsic parameters that deppends exclusively on the camera, such as its optical center and focal length; - Extrinsic parameters represents the location of the camera in the 3D scene. 2) Estimate the POSE at certain frequency since it deppends on the motion of the camera and/or the target. 3) Control the motion of the robot in order minimize the kinematic error.

This method is very sensitive to calibrations errors.

3.5. Background and Preliminaries

• Is there some particular theorem / "mathy" thing you require your readers to know before delving in the actual problem? Briefly explain it and links for more detailed explanations here.

Definition of link: - could be the reference to a paper / textbook - (bonus points) it is best if it is a link to Duckiebook chapter (in the dedicated “Preliminaries” section)

3.6. Definition of the problem

Up to now it was all fun and giggles. This is the most important part of your report: a crisp, possibly mathematical, definition of the problem you tackled. You can use part of the preliminary design document to fill this section.

Make sure you include your: - final objective / goal - assumptions made - quantitative performance metrics to judge the achievement of the goal

3.7. Contribution / Added functionality

Describe here, in technical detail, what you have done. Make sure you include: - a theoretical description of the algorithm(s) you implemented - logical architecture - software architecture - details on the actual implementation where relevant (how does the implementation differ from the theory?) - any infrastructure you had to develop in order to implement your algorithm - If you have collected a number of logs, add link to where you stored them

Feel free to create subsections when useful to ease the flow

3.8. Formal performance evaluation / Results

Be rigorous!

- For each of the tasks you defined in you problem formulation, provide quantitative results (i.e., the evaluation of the previously introduced performance metrics)
- Compare your results to the success targets. Explain successes or failures.
- Compare your results to the “state of the art” / previous implementation where relevant. Explain failure / success.
- Include an explanation / discussion of the results. Where things (as / better than / worst than) you expected? What were the biggest challenges?

3.9. Future avenues of development

Is there something you think still needs to be done or could be improved? List it here, and be specific!

UNIT A-4

Instructions template

Before starting, you should have a look at some tips on how to write beautiful Duckiebook pages.

This is the template for the description of a what we should do to reproduce the results you got in your project. The spirit of this document is to be an operation manual, i.e., a straightforward, unambiguous recipe for reproducing the results of a specific behavior or set of behaviors.

It starts with the "knowledge box" that provides a crisp description of the border conditions needed:

- Duckiebot hardware configuration (see Duckiebot configurations)
- Duckietown hardware configuration (loops, intersections, robotarium, etc.)
- Number of Duckiebots
- Duckiebot setup steps

For example:

KNOWLEDGE AND ACTIVITY GRAPH

> **Requires:** Duckiebot in configuration `DB19`
> **Requires:** Duckietown without intersections
> **Requires:** Camera calibration completed

4.1. Video of expected results

First, we show a video of the expected behavior (if the demo is successful).

Make sure the video is compliant with Duckietown, i.e. : the city meets the appearance specifications and the Duckiebots have duckies on board.

4.2. Laptop setup notes

Does the user need to do anything to modify their local laptop configuration?

4.3. Duckietown setup notes

Here, describe the assumptions about the Duckietown, including:

- Layout (tiles types)
- Infrastructure (traffic lights, WiFi networks, ...) required
- Weather (lights, ...)

Do not write instructions on how to build the city here, unless you are doing something very particular that is not in the Duckietown operation manual (unknown ref opmanual_duckietown/duckietowns)

previous warning next (3 of 8) index

warning

```
I will ignore this because it is an external link.

 > I do not know what is indicated by the link '#op-
manual_duckietown/duckietowns'.
```

Location not known more precisely.

Created by function `n/a` in module `n/a`.

. Here, merely point to them.

4.4. Duckiebot setup notes

Write here any special setup for the Duckiebot, if needed.

Do not repeat instructions here that are already included in the Duckiebot operation manual (unknown ref opmanual_duckiebot/opmanual_duckiebot)

previous warning next (4 of 8) index

warning

```
I will ignore this because it is an external link.

 > I do not know what is indicated by the link '#op-
manual_duckiebot/opmanual_duckiebot'.
```

Location not known more precisely.

Created by function `n/a` in module `n/a`.

.

4.5. Pre-flight checklist

The pre-flight checklist describes the steps that are sufficient to ensure that the demo will be correct:

Check: operation 1 done

Check: operation 2 done

4.6. Instructions

Here, give step by step instructions to reproduce the demo.

Step 1: XXX

previous task next (3 of 7) index

task

The following was marked as "status-XXX".

> Step 1: XXX

Location not known more precisely.

Created by function `n/a` in module `n/a`.

Step 2: XXX

previous task next (4 of 7) index

task

The following was marked as "status-XXX".

> Step 2: XXX

Location not known more precisely.

Created by function `n/a` in module `n/a`.

Make sure you are specifying where to write each line of code that needs to be executed, and what should the expected outcome be. If there are typical pitfalls / errors you experienced, point to the next section for troubleshooting.

4.7. Troubleshooting

Add here any troubleshooting / tips and tricks required, in the form:

> Symptom: The Duckiebot flies

Resolution: Unplug the battery and send an email to info@duckietown.org

> Symptom: I run `this elegant snippet of code` and get this error: `a nasty line of gibberish`

Resolution: Power cycle until it works.

4.8. Demo failure demonstration

Finally, put here video of how the demo can fail, when the assumptions are not respected.

You can upload the videos to the Duckietown Vimeo account and link them here.

PART B
Caffe and Tensorflow

Contents

UNIT B-1

How to install PyTorch on the Duckiebot

PyTorch is a Python deep learning library that's currently gaining a lot of traction, because it's a lot easier to debug and prototype (compared to TensorFlow / Theano).

To install PyTorch on the Duckietbot you have to compile it from source, because there is no pro-compiled binary for ARMv7 / ARMhf available. This guide will walk you through the required steps.

1.1. Step 1: install dependencies and clone repository

First you need to install some additional packages. You might already have installed. If you do, that's not a problem.

```
sudo apt-get install libopenblas-dev cython libatlas-dev m4 libblas-dev
```

In your current shell add two flags for the compiler

```
export NO_CUDA=1 # this will disable CUDA components of PyTorch, be-
cause the little RaspberriPi doesn't have a GPU that supports CUDA
export NO_DISTRIBUTED=1 # for distributed computing
```

Then `cd` into a directory of your choice, like `cd ~/Downloads` or something like that and clone the PyTorch library.

```
git clone --recursive https://github.com/pytorch/pytorch
```

1.2. Step 2: Change swap size

When I was compiling the library I ran out of SWAP space (which is 500MB by default). I was successful in compiling it with 2GB of SWAP space. Here is how you can increase the SWAP (only for compilation - later we will switch back to 500MB).

Create the swap file of 2GB

```
sudo dd if=/dev/zero of=/swap1 bs=1M count=2048
```

Make this empty file into a swap-compatible file

```
sudo mkswap /swap1
```

Then disable the old swap space and enable the new one

```
sudo nano /etc/fstab
```

This above command will open a text editor on your `/etc/fstab` file. The file should have this as the last line: `/swap0 swap swap`. In this line, please change the `/swap0` to `/swap1`. Then save the file with CTRL+o and ENTER. Close the editor with CTRL+x.

Now your system knows about the new swap space, and it will change it upon reboot, but if you want to use it right now, without reboot, you can manually turn off and empty the old swap space and enable the new one:

```
sudo swapoff /swap0
sudo swapon /swap1
```

1.3. Step 3: compile PyTorch

`cd` into the main directory, that you clones PyTorch into, in my case `cd ~/Downloads/pytorch` and start the compilation process:

```
python setup.py build
```

This shouldn't create any errors but it took me about an hour. If it does throw some exceptions, please let me know.

When it's done, you can install the pytorch package system-wide with

```
sudo -E python setup.py install # the -E is important
```

For some reason on my machine this caused recompilation of a few packages. So this

might again take some time (but should be significantly less).

1.4. Step 4: try it out

If all of the above went through without any issues, congratulations. :) You should now have a working PyTorch installation. You can try it out like this.

First you need to change out of the installation directory (**this is important - otherwise you get a really weird error**):

```
cd ~
```

Then run Python:

```
python
```

And in the Python interpreter try this:

```
>>> import torch
>>> x = torch.rand(5, 3)
>>> print(x)
```

1.5. (Step 5, optional: unswap the swap)

Now if you like having 2GB of SWAP space (additional RAM basically, but a lot slower than your built-in RAM), then you are done. The downside is that you might run out of space later on. If you want to revert back to your old 500MB swap file then do the following:

Open the `/etc/fstab` file in the editor:

```
sudo nano /etc/fstab
```
TODO

previous task next (5 of 7) index

task

The following was marked as "todo".

TODO

Location not known more precisely.

Created by function `n/a` in module `n/a`.

please change the `/swap0` to `/swap1`. Then save the file with CTRL+o and ENTER. Close the editor with CTRL+x.

UNIT B-2

How to install Caffe and Tensorflow on the Duckiebot

Caffe and TensorFlow are popular deep learning libraries, and are supported by the Intel Neural Computing Stick (NCS).

2.1. Caffe

1) Step 1: install dependencies and clone repository

Install some of the dependencies first. The last command "sudo pip install ....." will cause some time.

```
sudo apt-get install -y gfortran cython
sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev li-
bopencv-dev libhdf5-serial-dev protobuf-compiler git
sudo apt-get install --no-install-recommends libboost-all-dev
sudo apt-get install -y python-dev libgflags-dev libgoogle-glog-dev li-
blmdb-dev libatlas-base-dev python-skimage
sudo pip install pyzmq jsonschema pillow numpy scipy ipython jupyter
pyyaml
```

Then, you need to clone the repo of caffe

```
cd
git clone https://github.com/BVLC/caffe
```

2) Step 2: compile Caffe

Before compile Caffe, you have to modify Makefile.config

```
cd caffe
cp Makefile.config.example Makefile.config
sudo vim Makefile.config
```

Then, change four lines from

```
'#'CPU_ONLY := 1
/usr/lib/python2.7/dist-packages/numpy/core/include
INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include
LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib
```

to

```
CPU_ONLY := 1
/usr/local/lib/python2.7/dist-packages/numpy/core/include
INCLUDE_DIRS := $(PYTHON_INCLUDE) /usr/local/include /usr/include/hdf5/
serial/
LIBRARY_DIRS := $(PYTHON_LIB) /usr/local/lib /usr/lib /usr/lib/arm-lin-
ux-gnueabihf/hdf5/serial/
```

Next, you can start to compile caffe

```
make all
make test
make runtest
make pycaffe
```

If you did't get any error above, congratulation on your success. Finally, please export pythonpath

```
sudo vim ~/.bashrc
export PYTHONPATH=/home/"$USER"/caffe/python:$PYTHONPATH
```

3) Step 3: try it out

Now, we can confirm whether the installation is successful. Download AlexNet and run caffe time

```
cd ~/caffe/
python scripts/download_model_binary.py models/bvlc_alexnet
./build/tools/caffe time -model models/bvlc_alexnet/deploy.prototxt
-weights  models/bvlc_alexnet/bvlc_alexnet.caffemodel   -iterations 10
```

And you can see the benchmark of AlexNet on Pi3 caffe.

2.2. Tensorflow

1) Step 1: install dependencies and clone repository

First, update apt-get:

```
$ sudo apt-get update
```

For Bazel:

```
$ sudo apt-get install pkg-config zip g++ zlib1g-dev unzip
```

For Tensorflow: (NCSDK only support python 3+. I didn't use mvNC on rpi3, so here I choose python 2.7)

(For Python 2.7)

```
$ sudo apt-get install python-pip python-numpy swig python-dev
$ sudo pip install wheel
```

(For Python 3.3+)

```
$ sudo apt-get install python3-pip python3-numpy swig python3-dev
$ sudo pip3 install wheel
```

To be able to take advantage of certain optimization flags:

```
$ sudo apt-get install gcc-4.8 g++-4.8
$ sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8
100
$ sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.8
100
```

Make a directory that hold all the thing you need

```
$ mkdir tf
```

2) Step 2: build Bazel

Download and extract bazel (here I choose 0.7.0):

```
$ cd ~/tf
$ wget https://github.com/bazelbuild/bazel/releases/download/0.7.0/
bazel-0.7.0-dist.zip
$ unzip -d bazel bazel-0.7.0-dist.zip
```

Modify some file:

```
$ cd bazel
$ sudo chmod u+w ./* -R

$ nano scripts/bootstrap/compile.sh
```

To line 117, add "-J-Xmx500M":

```
run "${JAVAC}" -classpath "${classpath}" -sourcepath "${sourcepath}" \
      -d "${output}/classes" -source "$JAVA_VERSION" -target "$JAVA_VERSION" \
      -encoding UTF-8 "@${paramfile}" -J-Xmx500M
```

Figure 2.1

```
$ nano tools/cpp/cc_configure.bzl
```

Place the line return "arm" around line 133 (beginning of the _get_cpu_value function):

```
...
"""Compute the cpu_value based on the OS name."""
return "arm"
...
```

Figure 2.2

Build Bazel (it will take a while, about 1 hour):

```
$ ./compile.sh
```

When the build finishes:

```
$ sudo cp output/bazel /usr/local/bin/bazel
```

Run bazel check if it's working:

```
$ bazel
```

```
Usage: bazel <command> <options> ...

Available commands:
  analyze-profile     Analyzes build profile data.
  build               Builds the specified targets.
  canonicalize-flags  Canonicalizes a list of bazel options.
  clean               Removes output files and optionally stops the server.
  dump                Dumps the internal state of the bazel server process.
  fetch               Fetches external repositories that are prerequisites to the targets.
  help                Prints help for commands, or the index.
  info                Displays runtime info about the bazel server.
  mobile-install      Installs targets to mobile devices.
  query               Executes a dependency graph query.
  run                 Runs the specified target.
  shutdown            Stops the bazel server.
  test                Builds and runs the specified test targets.
  version             Prints version information for bazel.

Getting more help:
  bazel help <command>
                   Prints help and options for <command>.
  bazel help startup_options
                   Options for the JVM hosting bazel.
  bazel help target-syntax
                   Explains the syntax for specifying targets.
  bazel help info-keys
                   Displays a list of keys used by the info command.
```

Figure 2.3

3) Step 3: compile Tensorflow

Clone tensorflow repo (here I choose 1.4.0):

```
$ cd ~/tf
$ git clone -b r1.4 https://github.com/tensorflow/tensorflow.git
$ cd tensorflow
```

(Incredibly important) Changes references of 64-bit program implementations (which we don't have access to) to 32-bit implementations.

```
$ grep -Rl 'lib64' | xargs sed -i 's/lib64/lib/g'
```

Modify the file platform.h:

```
$ sed -i "s|#define IS_MOBILE_PLATFORM|//#define IS_MOBILE_PLATFORM|g"
tensorflow/core/platform/platform.h
```

Configure the build: (important) if you want to build for Python 3, specify /usr/bin/python3 for Python's location and /usr/local/lib/python3.x/dist-packages for the Python library path.

```
$ ./configure
Please specify the location of python. [Default is /usr/bin/python]: /usr/bin/python
Please specify optimization flags to use during compilation when bazel option "--config=opt"
Do you wish to use jemalloc as the malloc implementation? [Y/n] Y
Do you wish to build TensorFlow with Google Cloud Platform support? [y/N] N
Do you wish to build TensorFlow with Hadoop File System support? [y/N] N
Do you wish to build TensorFlow with the XLA just-in-time compiler (experimental)? [y/N] N
Please input the desired Python library path to use. Default is [/usr/local/lib/python2.7/di
Do you wish to build TensorFlow with OpenCL support? [y/N] N
Do you wish to build TensorFlow with CUDA support? [y/N] N
```

Figure 2.4

Build the Tensorflow (this will take a LOOOONG time, about 7 hrs):

```
$ bazel build -c opt --copt="-mfpu=neon-vfpv4" --copt="-funsafe-math-op-
timizations" --copt="-ftree-vectorize" --copt="-fomit-frame-pointer" --
local_resources 1024,1.0,1.0 --verbose_failures tensorflow/tools/
pip_package:build_pip_package
```

After finished compiling, install python wheel:

```
$ bazel-bin/tensorflow/tools/pip_package/build_pip_package /tmp/tensor-
flow_pkg
$ sudo pip install /tmp/tensorflow_pkg/tensorflow-1.4.0-cp27-none-lin-
ux_armv7l.whl
```

Check version:

```
$ python -c 'import tensorflow as tf; print(tf.__version__)'
```

And you're done! You deserve a break.

4) Step 3: try it out

Suppose you already have inception-v3 model (with inception-v3.meta and inception-v3.ckpt)

Create a testing python file

```
$ vim test.py
```

Write the following code:

```python
import tensorflow as tf
import numpy as np
import cv2
import sys
import time

def run(input_image):
    tf.reset_default_graph()
    with tf.Session() as sess:
        saver = tf.train.import_meta_graph('./output/inception-v3.meta')
        saver.restore(sess, 'inception_v3.ckpt')

        softmax_tensor = sess.graph.get_tensor_by_name('Softmax:0')
        feed_dict = {'input:0': input_image}
        classification = sess.run(softmax_tensor, {'input:0': input_image})    #first run fo warm-up

        start_time = time.time()
        classification = sess.run(softmax_tensor, {'input:0': input_image})
        print 'predict label:', np.argmax(classification[0])
        print 'predict time:', time.time() - start_time, 's'

if __name__=="__main__":
    args = sys.argv
    if len(args) != 2:
        print 'Usage: python %s filename'%args[0]
        quit()
    image_data =tf.gfile.FastGFile(args[1], 'rb').read()
    image = cv2.imread(args[1])
    image = cv2.resize(image, (299,299))
    image = np.array(image)/255.0
    image = np.asarray(image).reshape((1, 299, 299, 3))
    run(image)
```

Figure 2.5

Save, and excute it

```
$ python test.py cat.jpg
```

Then it will show the predict label and predict time.

UNIT B-3

Movidius Neural Compute Stick Install

3.1. Laptop Installation

install based on [ncsdk website]

```
git clone http://github.com/Movidius/ncsdk
cd ~/ncsdk
make install
make examples
```

test installation

```
cd ~/ncsdk/examples/app/hello_ncs_py/
make run
```

3.2. Duckiebot Installation

you only need to install the NCSDK but there is also the option of installing Caffe and/or Tensorflow as well, in order to perhaps speed up the development cycle. I would recommend against it, as it can be a bigger problem than it solves.

1) Barebones Install (recommended)

you don't need tensorflow, caffe, or any tools in order to run the compiled networks and not installing them will save you a lot of hassle

on duckiebot:

```
git clone http://github.com/Movidius/ncsdk
cd ~/ncsdk/api/src
make
sudo make install
```

2) Caffe/Tensorflow Install

> **Note:** if you want to be able to compile your models on the duckiebot itself, install tensorflow or caffe beforehand and remember to install for python 3 (`pip3`)

follow directions here

make sure caffe and tensorflow as installed

```
python3 -c 'import tensorflow as tf; import caffe'
```

install sdk:

```
git clone http://github.com/Movidius/ncsdk
cd ~/ncsdk
make install
make examples
```

UNIT B-4

How To Use Neural Compute Stick

4.1. Workflow

create and train model in tensorflow or caffe (brief note on configuration)
save tensorflow model as a `.meta` (or caffe model in `.prototxt`)

```
saver = tf.train.Saver()
...
saver.save(sess, 'model')
```

compile the model into NC format (documentation here)

```
mvNCCompile model.meta -o model.graph
```

move model onto duckiebot

```
scp model.meta user@robot name:~/path_to_networks/
```

run the compiled model

```
with open(path_to_networks + model.meta, mode='rb') as f:
    graphfile = f.read()
graph = device.AllocateGraph(graphfile)
graph.LoadTensor(input_image.astype(numpy.float16), 'user object')
output, userobj = graph.GetResult()
```

4.2. Benchmarking

get benchmarking (frames per second) from their app zoo

```
git clone https://github.com/movidius/ncappzoo
cd ncappzoo/apps/benchmarkncs
./mobilenets_benchmark.sh | grep FPSk
```

Part C

ETH Autonomous mobility on Demand 2019: Final Reports

Contents

UNIT C-1

Group name: final report

Before starting, you should have a look at some tips on how to write beautiful Duckiebook pages (unknown ref duckumentation/contribute)

previous **warning** next (5 of 8) index

warning

```
I will ignore this because it is an external link.

 > I do not know what is indicated by the link '#duck-
umentation/contribute'.
```

Location not known more precisely.

Created by function `n/a` in module `n/a`.

.

The objective of this report is to bring justice to your extraordinarily hard work during the semester and make so that future generations of Duckietown students may take full advantage of it. Some of the sections of this report are repetitions from the preliminary design document (PDD) and intermediate report you have given.

1.1. The final result

Let's start from a teaser.

- Post a video of your best results (e.g., your demo video): remember to have duckies on the robots or something terrible might happen!

Add as a caption: see the operation manual to reproduce these results. Moreover, add a link to the readme.txt of your code.

1.2. Mission and Scope

Now tell your story:

Define what is your mission here.

1) Motivation

Now step back and tell us how you got to that mission.

- What are we talking about? [Brief introduction / problem in general terms]
- Why is it important? [Relevance]

2) Existing solution

- Was there a baseline implementation in Duckietown which you improved upon, or did you implemented from scratch? Describe the "prior work"

3) Opportunity

- What was wrong with the baseline / prior work / existing solution? Why did it need improvement?

Examples: - there wasn't a previous implementation - the previous performance, evaluated according to some specific metrics, was not satisfactory - it was not robust / reliable - somebody told me to do so (/s) (this is a terrible motivation. In general, never ever ever say "somebody told me to do it" or "everybody does like this")

- How did you go about improving the existing solution / approaching the problem? [contribution]

Examples: - We used method / algorithm xyz to fix the gap in knowledge (don't go in the details here) - Make sure to reference papers you used / took inspiration from, lessons, textbooks, third party projects and any other resource you took advantage of (check here how to add citations in this document). Even in your code, make sure you are giving credit in the comments to original authors if you are reusing some components.

4) Preliminaries

- Is there some particular theorem / "mathy" thing you require your readers to know before delving in the actual problem? Briefly explain it and links for more detailed explanations here.

Definition of link: - could be the reference to a paper / textbook - (bonus points) it is best if it is a link to Duckiebook chapter (in the dedicated "Preliminaries" section)

1.3. Definition of the problem

Up to now it was all fun and giggles. This is the most important part of your report: a crisp, possibly mathematical, definition of the problem you tackled. You can use part of the preliminary design document to fill this section.

Make sure you include your: - final objective / goal - assumptions made - quantitative performance metrics to judge the achievement of the goal

1.4. Contribution / Added functionality

Describe here, in technical detail, what you have done. Make sure you include: - a theoretical description of the algorithm(s) you implemented - logical architecture - software architecture - details on the actual implementation where relevant (how does the implementation differ from the theory?) - any infrastructure you had to develop in order to implement your algorithm - If you have collected a number of logs, add link to where you stored them

Feel free to create subsections when useful to ease the flow

1.5. Formal performance evaluation / Results

Be rigorous!

- For each of the tasks you defined in you problem formulation, provide quantitative results (i.e., the evaluation of the previously introduced performance metrics)
- Compare your results to the success targets. Explain successes or failures.
- Compare your results to the “state of the art” / previous implementation where relevant. Explain failure / success.
- Include an explanation / discussion of the results. Where things (as / better than / worst than) you expected? What were the biggest challenges?

1.6. Future avenues of development

Is there something you think still needs to be done or could be improved? List it here, and be specific!

Part D

ETH Autonomous mobility on Demand 2019: Final Reports

Contents

UNIT D-1

Demo template

Before starting, you should have a look at some tips on how to write beautiful Duckiebook pages (unknown ref duckumentation/contribute)

previous **warning** next (6 of 8) index

warning

```
I will ignore this because it is an external link.

 > I do not know what is indicated by the link '#duck-
umentation/contribute'.
```

Location not known more precisely.

Created by function `n/a` in module `n/a`.

.

This is the template for the description of a demo. The spirit of this document is to be an operation manual, i.e., a straightforward, unambiguous recipe for reproducing the results of a specific behavior or set of behaviors.

It starts with the "knowledge box" that provides a crisp description of the border conditions needed:

- Duckiebot hardware configuration (see Duckiebot configurations)
- Duckietown hardware configuration (loops, intersections, robotarium, etc.)
- Number of Duckiebots
- Duckiebot setup steps

For example:

KNOWLEDGE AND ACTIVITY GRAPH

Requires: Duckiebot in configuration `DB18`

Requires: Duckietown without intersections

Requires: Camera calibration completed

1.1. Video of expected results

First, we show a video of the expected behavior (if the demo is successful).

Make sure the video is compliant with Duckietown, i.e. : the city meets the appearance specifications and the Duckiebots have duckies on board.

1.2. Duckietown setup notes

Here, describe the assumptions about the Duckietown, including:

- Layout (tiles types)
- Infrastructure (traffic lights, WiFi networks, ...) required
- Weather (lights, ...)

Do not write instructions on how to build the city here, unless you are doing something very particular that is not in the Duckietown operation manual (unknown ref opmanual_duckietown/duckietowns)

previous **warning** next (7 of 8) index

warning

```
I will ignore this because it is an external link.

 > I do not know what is indicated by the link '#op-
manual_duckietown/duckietowns'.
```

Location not known more precisely.

Created by function `n/a` in module `n/a`.

. Here, merely point to them.

1.3. Duckiebot setup notes

Write here any special setup for the Duckiebot, if needed.

Do not repeat instructions here that are already included in the Duckiebot operation manual (unknown ref opmanual_duckiebot/opmanual_duckiebot)

previous **warning** (8 of 8) index

warning

```
I will ignore this because it is an external link.

 > I do not know what is indicated by the link '#op-
manual_duckiebot/opmanual_duckiebot'.
```

Location not known more precisely.
Created by function `n/a` in module `n/a`.

.

1.4. Pre-flight checklist

The pre-flight checklist describes the steps that are sufficient to ensure that the demo will be correct:

Check: operation 1 done

Check: operation 2 done

1.5. Demo instructions

Here, give step by step instructions to reproduce the demo.

Step 1: XXX

previous task next (6 of 7) index
task
The following was marked as "status-XXX".

> Step 1: XXX

Location not known more precisely.
Created by function `n/a` in module `n/a`.

Step 2: XXX

previous task (7 of 7) index
task
The following was marked as "status-XXX".

> Step 2: XXX

Location not known more precisely.
Created by function `n/a` in module `n/a`.

Make sure you are specifying where to write each line of code that needs to be executed, and what should the expected outcome be. If there are typical pitfalls / errors you experienced, point to the next section for troubleshooting.

1.6. Troubleshooting

Add here any troubleshooting / tips and tricks required, in the form:

> Symptom: The Duckiebot flies

Resolution: Unplug the battery and send an email to info@duckietown.org

> Symptom: I run `this elegant snippet of code` and get this error: `a nasty line of gibberish`

Resolution: Power cycle until it works.

1.7. Demo failure demonstration

Finally, put here video of how the demo can fail, when the assumptions are not respected.

You can upload the videos to the Duckietown Vimeo account and link them here.

Other learning modules

Logo