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 opmanu-

al_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 '#opman-
ual_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 '#opman-
ual_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

Step 2: XXX

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

Real-time object detection: Project report

Authors :

Dishank BANSAL
Bhavya PATWA
Sophie CHAUDONNERET

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, Scope and motivation

Perception is a key component for any autonomous system. State-of-the-art autonomous driving technologies have object detection as part of their perception method. Unfortunately, benefits of object detection were never fully leveraged for Duckietown as this method runs into one big obstacle: real-time performance. Real time performance is crucial for robotics and as object detection is quite computationally expensive, its performance on the duckiebot is limited. The goal of this project is to propose a method for real-time object detection and tracking that can be run on a duckiebot with acceptable performance using a Jetson Nano.

1) Existing solution

For this work, and with this difficult context, we were not able to build our own dataset to train our different models. Fortunately, a previous project built a whole dataset for to implement object detection in Duckietown. We also found this project that aimed at implementing an object detector in DuckieTown. Unfortunately, this project seems to be deprecated, so we did not use it. Finally, we used the exercise 3 structure to implement our final object detector.

2) Opportunity

As stated before, no object detectors were implemented in the DuckieTown pipeline for one good reason : it could not run on the Raspberry Pi in real time. This year, we were lucky to also have a Jetson Nano that has a more powerful GPU. We therefore decided to try and implement an object detector that : - run faster with Tracking while maintaining a good accuracy, - can run on the Jetson Nano at reasonable speed.

To do this, we compared and used different objection detection methods to find out which offers the best compromise between performance and accuracy in the DuckieTown setting. To increase the speed of the object detection, we also used tracking between two object detections. With the final OD real-time pipeline, we tried to implement a way to avoid detected obstacles.

3.3. Background and Preliminaries

In this report a few preliminary knowledge is needed.

The main mathematical difficulty that the reader can encounter is the Kalman Filter. Kalman filter is used in this project in the Tracking step. The main idea of the Kalman filter is that, given a model of evolution of our state, its noise model and the measurement and noise measurement model of our system, we can firstly predict the next step state then, with our measurement corresponding to this new step, we can update to take into account both the dynamic model and the measurement. A full lecture was given by Dr. Forbes on this subject there.

On another subject, the two neural networks that are presented are compared have two very different architectures. Indeed, there are mainly two types of object detectors. On the one hand, we have the one-stage object detectors, such a Yolo or SSD-MobileNet, which make a fixed number of predictions on grid. On the other hand, the two stages object detectors use a proposal network to find approximately objects and then use a second network to fine-tune these detections and give final predictions, such as Faster-RCNN or MaskRCNN. One stage ODs tend to have faster inference time while two stages ODs tend to have higher mean average precision. This article explains quite thoroughly the differences and similarities between the two architectures.

3.4. Object detection models : FasterRCNN vs. YOLOv5

1) FasterRCNN architecture and performance

As mentioned above, FasterRCNN is a two stage detector. The first stage is called the RPN (**R**egion **P**roposal **N**etwork), it processes the images by a feature extractor and keep only the topmost feature maps to to predict bounding box proposals. The second stage then crop features from the topmost feature maps using these bounding box proposals. The cropped features are then passed on to the FastRCNN for bounding box regression and classification.

As its name suggests, FastRCNN is a faster version of R-CNN. Its architecture is presented in figure 3.1.

In figure 3,2, you can see the 2 stages mentioned above and the FastRCNN module.

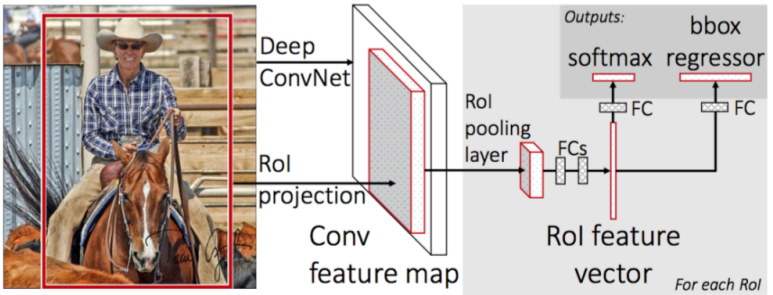

Figure 3.1. Fast-RCNN architecture (source : https://arxiv.org/abs/1504.08083)

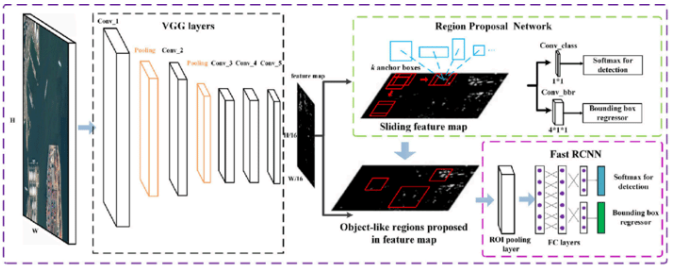

Figure 3.2. Faster-RCNN architecture (source : https://www.researchgate.net/figure/The-architecture-of-Faster-R-CNN_fig2_324903264)

2) Yolo architecture and performance

YOLOv5 is a one stage object detector, like any one stage detector, it is made of three main parts :

- model backbone
- model neck
- model head

The model backbone is used in object detection to extract the most important features : the richest and most distinctive ones. In YOLOv5, the backbone used is CSPNet which stands for **C**ross **S**tage **P**artial **Net**works.

Model neck is used in object detectors to build feature pyramids in order to detect an object of different sizes and scales. There are many different feature pyramid techniques available. YOLOv5 uses PANet, which stands for **P**ath **A**ggregation **Net**work.

The YOLOv5 model head is the same as in the previous version of Yolo.

Figure 3.3 gives an overall representation of YOLOv5 architecture.

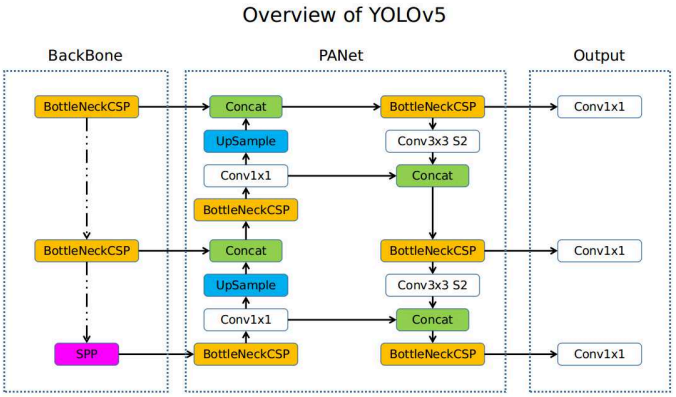

Figure 3.3. YOLOv5 architecture (source : https://github.com/ultralytics)

3.5. Tracking

1) Motivation

Now that we have found the detector that provides the best compromise between performance and accuracy, we wanted to be able to speed up the detection process by tracking the detected bounding boxes between two detections in order to be able to skip frames in our object detector.

Moreover, Tracking can help to recover dropped detection for in-between frames.

2) Kalman filter

To track a bounding over frames, we will be working in pixel space. For tracking, we assume here that the bounding boxes moves at constant speed in pixel space.

We will use a Kalman filter to track our bounding boxes. Let $\mathbf{X_k}$ be the state vector that represents the bounding boxe coordinates and their velocities.

$$\mathbf{X_k} = [x_1, y_1, x_2, y_2, v_{x,1}, v_{y,1}, v_{x,2}, v_{x,2}]$$

The motion model of the system that will be used for prediction is quite simple (as velocity is assumed constant):

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & 0 & dt & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & dt & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The bounding box detection by object detector is used as measurement in the update

step. The measurement is the bounding box coordinates :

$$\mathbf{z_k} = [x_1, y_1, x_2, y_2]$$

The measurement model of the system is then :

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The final Kalman filter equations for bounding box tracking is :

- Prediction step :

$$\begin{aligned} \hat{\mathbf{x}}'_k &= \mathbf{F}\hat{\mathbf{x}}_{k-1} \\ \mathbf{P}'_k &= \mathbf{F}\mathbf{P}'_{k-1}\mathbf{F}^T + \mathbf{Q} \end{aligned}$$

- Update step :

$$\begin{aligned} \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}'_k + \mathbf{K}_k\left(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}'_k\right) \\ \mathbf{P}_k &= \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}\right)\mathbf{P}'_k \\ \mathbf{K}_k &= \mathbf{P}'_k\mathbf{H}^T\left(\mathbf{H}\mathbf{P}'_k\mathbf{H}^T + \mathbf{R_k}\right)^{-1} \end{aligned}$$

3) Hungarian filter

In the previous section, we only talk about one bounding box that we track through frames.

In reality it is more complex : when there are multiple detections (i.e multiple bounding boxes), how to know which observation associate with which prediction at the update step ?

A solution is to use the *Hungarian algorithm* for Data association.

Let there be N predicted bounding boxes and M observations (detected bounding boxes). The Hungarian Algorithm will match the N boxes to N observations among the M possible so that the solution is optimal over a given metric. Here, the metric used is **IoU**, which stands for **I**ntersection **o**ver **U**nion. It is computed using this formula :

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

3.6. Object avoidance

In this section, we will see how we can use the duckiebot and duckie detector to implement certain behaviours in order to make DuckieTown safe again. We had two main goal behaviours :

- Stopping in front of an obstacle
- Overtaking an obstacle

1) Compute obstacle position in lane

First of all, we need to know, from the bounding boxes detected by our detector, the position of the obstacles. This position will be used to pass two information to the lane controller :

- Is there an obstacle close enough in our lane ?
- Is there a close obstacle in the other (left) lane ?

Let's take a bounding box. The coordinate of this box is given in pixels in the distorted image (due to the camera lens). First, we need to compute the center of the obstacle on the ground. Then, we need to rectify the center coordinates so that it corresponds to the rectified image. Since the point is considered on the ground (low edge of the box), we can use the GroundProjection module (used for line detection) to estimate the real coordinates from the duckiebot's origin. Then, with the duckiebot's lane pose, we can compute the obstacle lane pose using this formula :

$$pose_y = \cos(\phi)(y + d) + \sin(\phi)x$$

A diagram given in figure 3.4 illustrates the situation, the grey rectangle being the duckiebot and the yellow cross an obstacle.

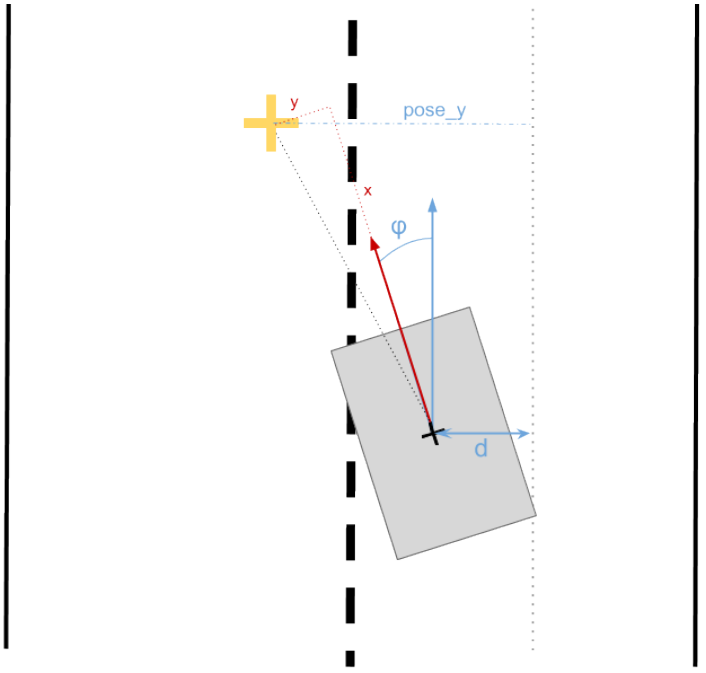

Figure 3.4. Obstacle position diagram

In the figure 3.5, the flowchart to make the decision is detailed.

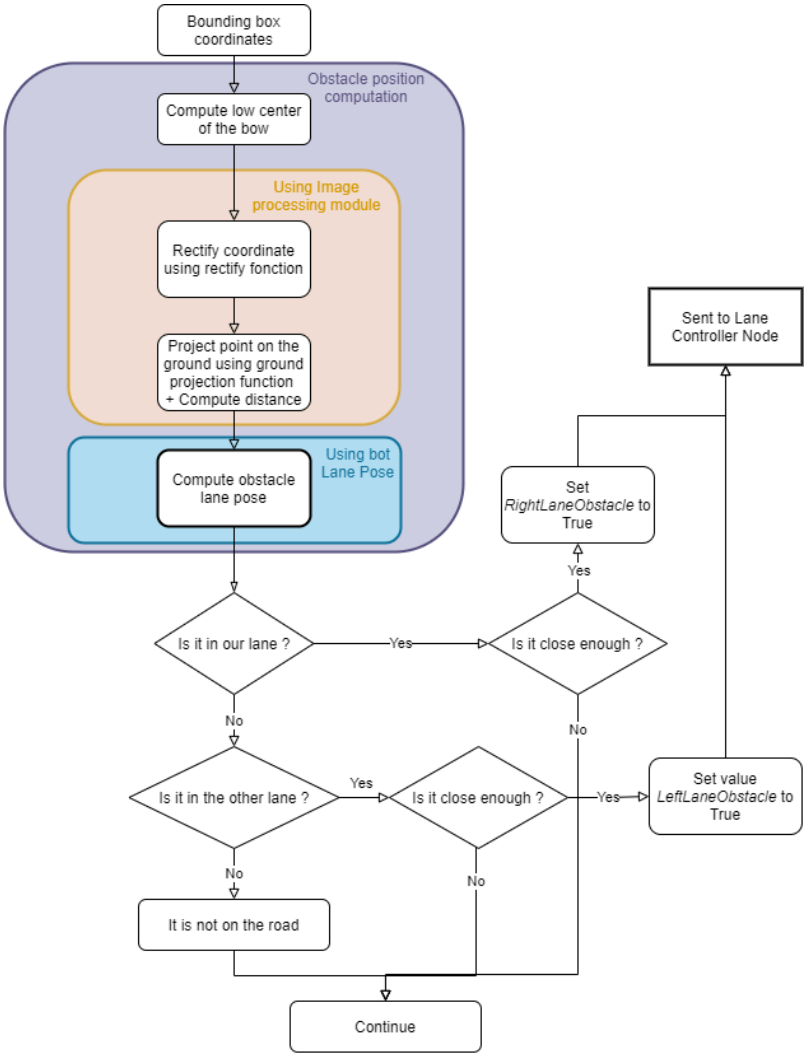

Figure 3.5. Obstacle position flowchart

2) Stopping in front of an obstacle

The first behaviour is quite straight forward : if an obstacle (duckiebot of duckie) is detected in our lane close enough from the bot, the lane controller passes $\boldsymbol{v} = \mathbf{0}$ to the wheel command. Figure 3.6 details the algorithm used to stop in front of an obstacle.

Algorithm 1: Stop at obstacle

```
Input : a list of N boxes and a list of N corresponding labels;
for idx in labels do
    label = labels[idx];
    if label == class["duckie"] OR label == class["bot"] then
        (x_min, y_min, x_max, y_max) = boxes[idx];
        low_center = [(x_min + x_max)/2, y_max];
        center_rectified = Rectifier(low_center) ;
        center_projected = GroundProjection(center_rectified) ;
        if ObjectIsInLane(center_projected) then
            dist = sqrt(center_projected.x^2 + center_projected.y^2) ;
            if dist <= d_safe(safedistance) then
                v = 0
            end
        end
    end
end
```

Figure 3.6. Algorithm to stop in front of an obstacle

3) Overtaking an obstacle

Here the problem is more challenging : avoiding an obstacle. In the literature, obstacle avoidance is well researched and most of the solutions proposed use graphs in which the vehicle must find the shortest path while respecting constraints or path planning.

In DuckieTown, we thought it would be simpler to overtake the obstacle by switching lane.

Our solution is to change the d_off parameter used in the Lane Controller Node to make the duckiebot believe it is not in the right lane.

First, we need to increase the d_off parameter so that the bot moves to the left lane, then keep it increased while it passes the obstacle and finally decrease it to switch back to the right lane.

The decision process to know when to overtake is detailed in figure 3.5. In figure 3.7, the flowchart to overtake the obstacle is explained. The corresponding algorithm is also detailed in figure 3.8.

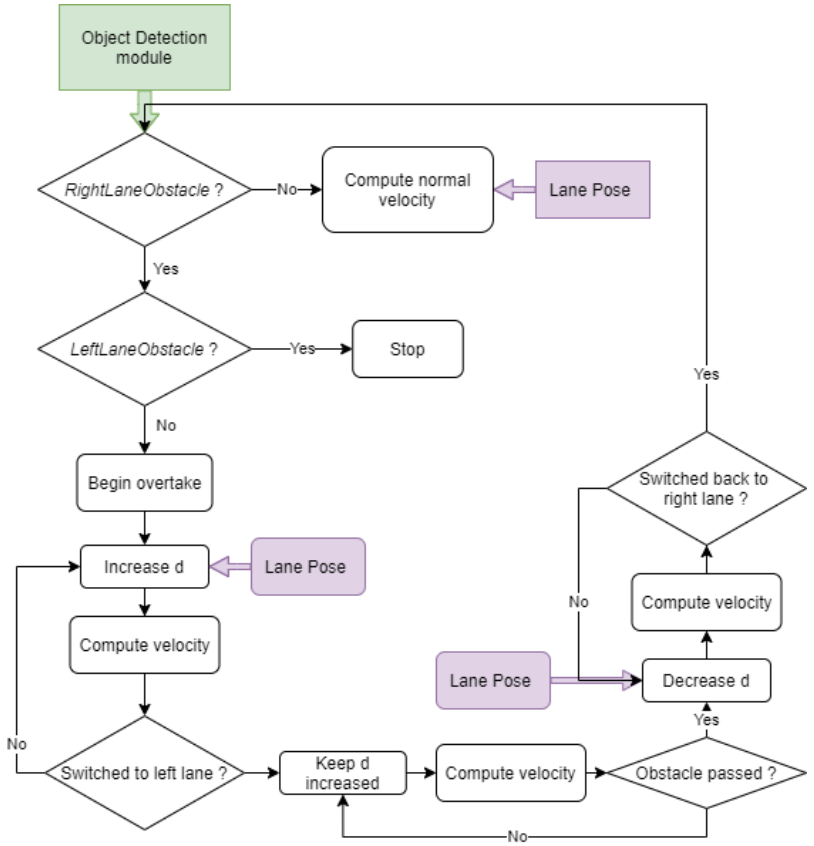

Figure 3.7. Obstacle overtaking flowchart

Algorithm 2: Overtake obstacle

Input : Information on obstacles in left lane and closest obstacle in right lane;
if *no obstacles in the left lane* **then**
  if *close enough obstacle in right lane* **then**
    begin overtaking;
    nbr_step_switch = tuned integer nbr_step = depends on obstacle;
    step = 0;
    while *step≤ nbr_step_switch* **do**
      $d\_offset = lane\_width \times \sin(\frac{step}{nbr\_step}\frac{\pi}{2})$;
      Send d_offset to Lane_controller_node;
      step += 1
    end
    Wait for nbr_step;
    step = 0;
    while *step≤ nbr_step_switch* **do**
      $d\_offset = lane\_width \times \sin(\left(1 + \frac{step}{nbr\_step}\right)\frac{\pi}{2})$;
      Send d_offset to Lane_controller_node;
      step += 1
    end
    end overtaking;
    end

Figure 3.8. Algorithm to overtake an obstacle

3.7. Formal performance evaluation / Results

We compared the different detectors using different processors to assess which will perform best on the duckiebot.

Here are some specifications regarding the material used to obtain the metrics provided above :

- CPU : AMD Ryzen Threadripper 1950X 16-Core Processor
- GPU : GeForce RTX 2080 Ti, 11 GB
- RAM : 32 GB
- Jetson Nano : specifications cqn be found here

The metrics used to assess the object detector's performance are **FPS** (**F**rames **P**er **S**econd) and **mAP** (**m**ean **A**verage **P**recision). The first one measures the detector's speed and the second one its accuracy.

We would like to remind the reader here that the following numbers are **without Tracking** and that FPS can be easily **doubled** if we skip every other image.

Here is the performance of **FasterRCNN** with two different backbones : *Resnet50* and *Resnet18*. Both were tested using the DuckieTown gym mentioned [above]{#real-time-object-detection-final-literature}.

- Using the **Resnet50** backbone :

TABLE 3.1

Proposals	FPS (on GPU)	FPS (on CPU)	mAP
300	55.5 (0.018s)	1.8 (0.55s)	83.9%
50	77 (0.013s)	2.6 (0.38s)	83.8%
10	77 (0.013s)	2.7 (0.36s)	74.3%

- Using **Resnet18** :

TABLE 3.2

Proposals	FPS (on GPU)	FPS (on CPU)	mAP
300	111 (0.009s)	5.55 (0.18s)	86.472%
50	142 (0.007s)	7.69 (0.13s)	86.462%

YOLOv5 has been tested on high and low-resolution images.

TABLE 3.3

Resolution	FPS (on GPU)	FPS (on CPU)	FPS (on Jetson Nano)	mAP
640x480	110 (0.009s)	9.6 (0.104s)	5 (0.200s)	71.14%
320x240	113 (0.009s)	19.5 (0.051s)	10 (0.100s)	68.56%

3.8. 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 opmanu-

al_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 '#opman-
ual_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 '#opman-
ual_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

Step 2: XXX

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
```

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:

```
1  import tensorflow as tf
2  import numpy as np
3  import cv2
4  import sys
5  import time
6
7  def run(input_image):
8      tf.reset_default_graph()
9      with tf.Session() as sess:
10         saver = tf.train.import_meta_graph('./output/inception-v3.meta')
11         saver.restore(sess, 'inception_v3.ckpt')
12
13         softmax_tensor = sess.graph.get_tensor_by_name('Softmax:0')
14         feed_dict = {'input:0': input_image}
15         classification = sess.run(softmax_tensor, {'input:0': input_image})    #first run fo warm-up
16
17         start_time = time.time()
18         classification = sess.run(softmax_tensor, {'input:0': input_image})
19         print 'predict label:', np.argmax(classification[0])
20         print 'predict time:', time.time() - start_time, 's'
21
22 if __name__=="__main__":
23     args = sys.argv
24     if len(args) != 2:
25         print 'Usage: python %s filename'%args[0]
26         quit()
27     image_data =tf.gfile.FastGFile(args[1], 'rb').read()
28     image = cv2.imread(args[1])
29     image = cv2.resize(image, (299,299))
30     image = np.array(image)/255.0
31     image = np.asarray(image).reshape((1, 299, 299, 3))
32     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 '#ducku-
mentation/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 '#ducku-
mentation/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 '#opman-
ual_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 '#opman-
ual_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

Step 2: XXX

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