






# Group 7

## Duck Squad

### Intersection Navigation

### Design Document

#### Team Information

			
Name	Yanni Kechriotis	Samuel Montorfani	Benjamin Dupont
	Ykechriotis@student.ethz.ch	smontorfani@student.ethz.ch	bdupont@student.ethz.ch
	point of contact		



# Part 1: Mission and scope

## Mission statement

### Your Mission

*Description of the goal of this project:*

With this work, we aim to enable multiple duckiebots to navigate intersections in duckietown without colliding

### Your Motto

*"The purpose is not the destination but the travel itself"*

## Project scope

### What is in scope

- Detect intersections in duckietown.
- Decide whether to stop or go or turn (depending on other agents, communicate using LEDs and their colors?).
- Be able to navigate the intersection by turning left/right or going straight on depending on the intersection options.
- Bonus idea: Stop before hitting a duckiebot in front of us in the same lane.
- Bonus idea: Navigate multiple duckiebots at a time arriving at an intersection
- Bonus idea: Be able to handle unplanned events (emergency vehicle, duckiebot with no lights, etc...)

Note: Some of these bonus ideas will try to be tackled if time permits, but are for now out of scope

### What is out of scope

- Handling different sized intersections (The intersection is assumed to be of the same "type", i.e red stop line and same dimensions for the turning radii, etc...)
- Doesn't need to be robust to sensor failure (No time for that)
- Navigating around malfunctioning duckiebot
- Lane following code (This is given from the current duckiebot software stack and will not be implemented).



## Part 2: definition of the problem

### Problem statement

*Write down the problem as you understand it. Describe the limitations of current solutions:*

**Currently duckiebots cannot navigate intersections reliably. If two duckiebots approach an intersection, they do not communicate their intentions to each other in any way. We want to be able to consistently turn into a desired lane and do so without colliding with any other agent. We want to communicate an agent's intention to any other agents at the intersection and make decisions based on the situation.**

### Assumptions

*What assumptions are you making about the problem/scenario?*

- *All sensors work*
- *Intersections are standard size and have standard markings and are clearly visible (no buildings or anything blocking view)*
- *Duckiebots are standard size and shape*

### Assessment

*How can we measure the performance of the future solution?*

*What would the "demo" be?*

- *Create small city with a few intersections, then let a couple duckiebots loose, and watch them navigate the intersections*
- *Show a couple increasingly complex scenarios*
  - *Single duckiebot approaching an intersection and turning*
  - *Two duckiebots approach an intersection at the same time*
  - *Three, more*
  - *Bonus: traffic lights, emergency vehicle, duckiebot not communicating*
- *Measurement of success: number of collisions, quality of navigation*



## Part 3: Approach

### This is the plan

*Describe in some detail how to solve the problem, down to the level of modules/algorithms.*

...

1. **Detect intersection (Sam)**
  - a. **Stop at red line, orient correctly**
2. **Randomly decide to go left, right, or straight (Ben)**
3. **Communicate decision to other duckiebots using LED's (Ben)**
4. **Detect other duckiebot LED's to learn their decisions (Ben)**
5. **Apply a decision making priority (Ben)**
6. **Decide to execute action or wait further**
7. **Execute action (Yanni)**

### Risk analysis

#### Possible Risks

*What can make this project fail?*

- 4 duckiebots at an intersection / symmetry
- Delayed decision making
- Duckiebot field of view
- LED detection issues
- Random component failures

#### Risk mitigation

*How can you reduce the risk?*

- Create robust priority system
- For field of view issue:
  - Detect duckiebots as you approach the intersection
  - Turn in place to see more of intersection
  - Always assume someone is on the left
- Make a random decision after some time to prevent four duckiebot roadblock
  - Maybe after a timer



# FINAL REPORT

## Intersection Navigation

Group 7 - Duck Squad

Yanni Kechriotis  
Samuel Montorfani  
Benjamin Dupont

### Part 1: Describe what you did

*Note: this is supposed to be the longest section of your document (at least 1-2 pages long)*

*Detailed description of your solution (software models, interfaces, diagrams...)*

*What did you develop on your own?*

*What external components did you use?*

Our duckietown project is all about intersection navigation. The purpose of our project is to enable navigation through intersections in duckietown. The starting point is a lane following code, which allows navigating through roads but does not handle intersection crossings.

We wanted to create a state machine component that allows duckiebots to successfully navigate through a town with intersections by progressing through different behaviors. Due to difficulties understanding the component framework, we instead chose to create our own classes and functions. The high level overview is described as follows:

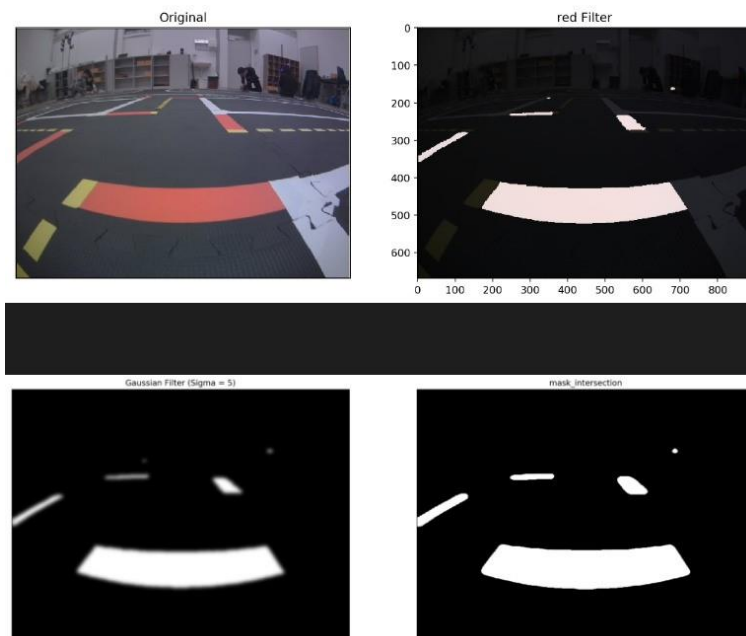
The duckiebot begins by following the lane. It uses its camera to detect if it is at an intersection by looking for the red markers, and stops in front of them. It will then choose a possible action at random (straight, left, right) depending on the type of intersection. Next it will light its LED's based on the chosen action to show the others where it plans on going. It will use the camera to detect if there are any other duckiebots at the intersection, and if they have an action with a higher priority than us. If so, it will wait, and if not, it will proceed with its action to cross the intersection, and then return to the lane following behavior. The project is thus split into several parts: detecting and classifying intersections, choosing a direction and displaying it with leds, detecting other duckiebots LED's, interpreting the signal, deciding if it should move or wait (priority), and finally executing the action (turning or going straight ahead).



For the first part of the project, detecting the intersections, we created the “intersection\_detection” function, which takes as input the camera image and outputs the image of the mask used for detection, a boolean to stop (or not) the motors, and a list of boolean for the possible actions to take (between left, straight and right).

The detection of an intersection is achieved through a series of masks and filters applied to the original image received from the Duckiebot's camera. A first mask excludes the horizon, considered as all pixels above a certain threshold. Then, a mask is applied within a specific range of red color, intensity, and illumination. A Gaussian filter is applied to the resulting image to prevent the same object from being considered as multiple close objects. Finally, another mask is applied, retaining only the pixels in the current image above a certain intensity threshold. The final result is a binary image where distinct objects appear within a specific range of colors.

To determine the type of intersection, the resulting binary image is analyzed to determine the number and size of the relevant objects. Only objects with an area greater than a certain threshold are considered as stop lines belonging to the intersection. This is because stop lines belonging to more distant intersections can also be detected in the filtered image. The stop lines are then classified based on their position in the image and sent as the output of this function, along with the filtered image and a boolean variable 'stop' signaling the imminent proximity of the stop line where the Duckiebot must stop. Once a stop line considered close enough is detected, the motors are stopped. Additionally, the function outputs the possible actions based on the type of intersection. A “+” type intersection will have four markings, and all actions are possible, while a “T” type intersection only has three markings, and there are only two possible actions.



*Detection of a “+” type intersection*

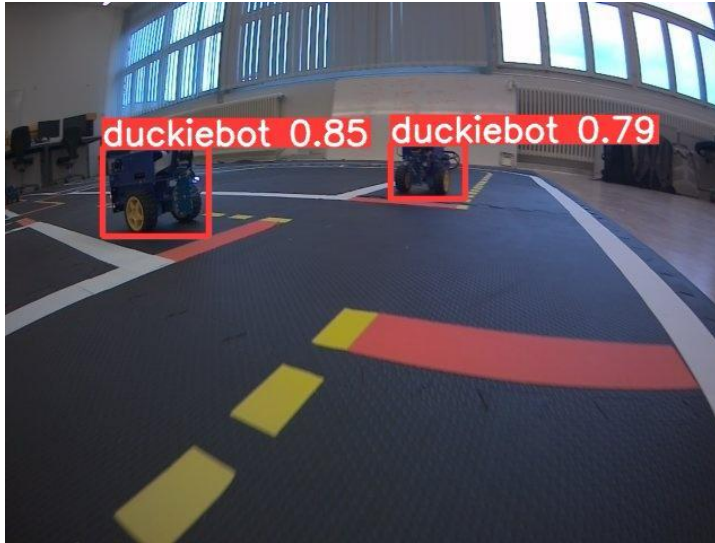


A brief “adjustment” state is used to move the duckiebot closer to the intersection marking, and adjust its heading so that it directly faces the intersection. This is done by determining the angle of the bottom rectangle in the masked image, and slowly turning until the angle is near zero.

At this point, the state machine moves to the “LED detection” state. In this state, the duckiebot will light up all 4 LEDs (although front right never works for some reason) with a color based on its randomly chosen action. BLUE for straight, GREEN for right, or RED for left. With the motors still stopped, it will read the camera image and look for other duckiebots at the intersection. It does this using a custom trained yolov5 object detection model. To achieve this, we gathered data using the duckiebot’s camera in various lightings throughout the day, making sure to capture various backgrounds and various parts (especially intersections) of the duckietown to get a very diverse dataset to train on. We then labelled roughly 500 images and used a 80% split for training and validation. The reason behind needing to do this is the non-reliable LED detection we faced, which appeared to be hard to tune to avoid false positive detection, which ultimately is very bad for the priority decision stack. Using yolov5 object detection resulted in very reliable detection (No false positive nor false negative occurred in our testing phases), primarily thanks to the size of the duckiebot and its distinct features that are quite easily identifiable for a learning based model. Alternatively it would take way longer to use and especially tune so called “traditional computer vision” approaches and get a similar robustness in the detection, even considering the labelling time. Additionally with this robust detection we are able to extract bounding boxes for where the duckiebots are and use these reduced image parts to do LED color detection on smaller areas which is of course much simpler and robust. The information of the other Duckiebots' position (on the right, in front, and sometimes left) from the object detection model and intention from its LED color is therefore sent as input for the decision making task.

The priority system then follows the normal rules of the road, with a key difference that only one duckiebot at a time can cross the intersection. When a Duckiebot enters the intersection, it signals this with a change in the color of its LEDs to purple. In this situation, all other Duckiebots must wait. If the intersection is clear, right-of-way is given to the vehicle approaching from the right. If two Duckiebots are facing each other, the priority depends on their intention: going straight or turning right takes precedence over a left turn, following standard traffic rules. In case of a conflict (such as 4 Duckiebots present at an intersection), all Duckiebots will wait for a certain time. The first one that arrived at the intersection will be the first to break the symmetry and move into the intersection, signaling with the color purple] of movement. The other Duckiebots must then wait, and the priority check begins again.

Unfortunately due to time constraints and difficulties with hardware, we were not able to fully implement a robust priority system. Currently duckiebots only detect other duckiebots, as the LED detection is not fully robust enough, so only the “right of way” priority works well.



*Duckiebot detection using yolov5*

Finally, once the duckiebot knows that it can safely cross through the intersection, it must be able to execute its chosen action consistently without relying on lane markings. Instead of relying on the camera like it does for lane following, the duckiebot will now rely on the motor encoders to navigate through the intersection. Based on the action, a reference ratio will be assigned between the left and right encoder values. A right turn has a  $LR\_ratio > 1$ , left turn  $LR\_ratio < 1$ , and straight  $LR\_ratio = 1$ . The value of the  $LR\_ratio$  determines the turning circle radius. Adequate values are found experimentally. PID control is used to apply motor PWM values to follow the desired  $LR\_ratio$ , and odometry is used to stop turning after reaching a desired theta value.

By combining these three behaviors, the duckiebot should now be able to detect intersections, stop before them, scan for other agents, and successfully navigate the intersection. Our demonstration will show this working at increasing scales, starting with a single duckiebot navigating through intersections, and then adding other agents to demonstrate the priority system.

## Part 2: Evaluation

### Systematic evaluation Method

What is your evaluation procedure(s)?

- Turn completion rate
- Duckiebot detection accuracy
- Intersection detection accuracy
- Number of crashes
- Number of off road occurrences





## Systematic evaluation Results

What are your quantitative results?

- Intersection detection accuracy ~90%
- Turn completion rate ~85%
- Duckiebot detection accuracy ~95%
- Number of crashes ~10% of the time
- Number of off road occurrences ~40% of the time (much of the time due to camera delay, weird motor behaviors, or other hardware issues)

## Demo instructions [in the repo]

Write detail demo instructions not here, but in the repo.

Include setup of the environment, things to run, expected results. Make a video showing operations.

Written in the repo

## Part 3: next steps

### Further work

How would you extend this work?

What additional skills would be unlocked on the duckiebots?

How much effort would be needed?

- More robust tie breaker to solve four way tie
- Robust to duckiebot not following the rules, or emergency duckiebots
- Traffic light intersections
- Make multiple duckiebots able to drive through intersection simultaneously with little to no constraints on the traffic density
- Integrate code better into component framework

For the successful completion of these actions, however, a higher reliability of hardware executions would be needed.



## Considerations

If you had to work on this project from scratch, what would have you changed from the initial plan?

Given we only had a short time to complete this project, if we were to restart we would probably want to lessen our project scope, perhaps by only focusing on one or two of the parts rather than all three. We feel that we successfully implemented each of the individual behaviors, but combining them together was very challenging.

Additionally, it would have been easier to use ROS directly, where the interface between actuators and sensors would be already built and we would just have to subscribe and publish to these topics. We faced a lot of issues in building the state machine since it wasn't trivial to stop or start components as we would like to.

## Part 4: Retrospective

### Project scope

*Did the project change scope? Why?*

The purpose of our project has remained the same since the initial idea, but with changes due to obstacles and problems that we had to face.

Yes the project changed scope because we only had a short time to complete it and ran into many issues with the hardware. The lane following algorithm did not work well originally, and even after some tentative fixes, it frequently fails due to the camera or other components becoming very delayed. It also only works on "outside" lanes, where white markings are on the right side and yellow on the left side. The camera delay also affected our intersection and LED detection algorithms. Much time was spent trying to remedy these issues, which left us with even less time to actually build and test our project. Much time was also spent trying to figure out how exactly the component framework worked. It would have been nice to have more documentation or a longer tutorial video for these components, or if they could have been integrated into the weekly assignments throughout the course so that we could gain experience using them. The notebooks indeed show how to do basic component manipulation but whenever we wanted to apply this to our usecase it resulted in just more issues and was easier to just find a way to use them as little as possible to avoid delays, unexplainable motors spinning or code crashes.

### Approach

*Did your approach change? Why?*

Our original approach was to use the component framework and define components for each of our behaviors. We ran into issues with starting and stopping components, and other



problems, so we moved to defining our own functions and classes that made use of the components only when needed.

We also wanted to use the lane following behavior for the duckiebot to approach the intersections and continue after crossing them, but due to it not working well we considered changing to simply commanding the duckiebot to drive straight.

For the LED detection we first anticipated being able to detect duckiebots' positions and intended direction to follow, but it was very difficult to get a robust LED detection working especially when it should expect various colours as input (even whilst selecting the colors in a smart way). That is why we decided to go with a learning based approach and use yolov5 as the results of a "proof of concept" with 50 images were very promising.

## Results

Did you expect such results? If not, why? What happened?

Is this project to be considered successful according to your initial commitment?

We did not expect to have so many issues with the hardware and lane following code, which was supposed to work out of the box. Along with the camera issues, the motors would behave randomly at times, seemingly speeding up or slowing down on their own. Additionally, changing the color of the duckiebot LED's sometimes caused one motor to activate for a couple seconds, causing the duckiebot to spin and ruin our tests.

In the end we were able to successfully complete each individual part of the project (intersection detection, LED detection, and intersection navigations), but combining them all together to form a consistent behavior was very difficult due to the aforementioned issues.

We are satisfied with the result of our project, especially considering the success of the various challenges that this project required. The results show that the Duckieboats perform their actions correctly, with a precision that can be variable and not always satisfactory.

The detection and classification of intersections and LEDs of other Duckiebots are successfully carried out. Signaling with LEDs and respecting priorities are performed correctly. Finally, the actions taken within the intersection are carried out with a good level of precision. However, the combined execution of actions can lead to a less satisfactory result.

The main reason seems to be due to more or less pronounced delays in the execution of actions or the transmission of information. This has significantly reduced the quality of the final result. The reason for these delays is not clear and appears more or less pronounced seemingly randomly. Apparently, it is a hardware-related issue, for which we have been unable to make improvements.

Overall, we are still satisfied with our project and its results

## Considerations

What did you like about this project?



*What would have you changed about it?*

*What could have done better?*

We enjoyed how we were able to define our own project scope and many of the details of what we would be working on ourselves. This freedom allowed us to adapt along the run and take decisions on the next steps that should be worked on based on the status of our progress.

We would like to have been given more time to complete the project. It would have been nice if the component framework was used in previous assignments, so we could build understanding of it throughout the class. We may also have preferred if we were able to use ROS ourselves directly, which would probably have been easier since there would have been more documentation available online, but also certainly a big benefit for the future projects in the robotics industry, provided that we have more time for the project.

Understanding the seemingly random delays in the execution of the code could lead to more precise and satisfying results in a future. Again the component framework made it hard to monitor the rate of the components or the queue sizes, and understand if the sensor was responsible for the delay or if the code was. This was not always obvious since sometimes even running the base component notebooks provided resulted in huge delays which could not be explained.