ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Mobility on Demand: From Car to Fleet
Class of 2019

DUCKIETOWN

Adaptive Lane Following

Project Report

Arreghini Simone & Griffa Pietro

Zurich
December 2019

Contents

1 Preview

In this report the applicability of an Adaptive Controller to the Duckiebot platform is studied, with the intent of autonomously identifying a correct value for the trim without the necessity of the manual calibration. The project starts from the analysis of the theoretical basis of the problem, up to the practical effects of the insertion of this mechanism within the real system. Results show that is is possible to estimate a right value for the trim within some degree of precision, and the method gives good hope for the possibility of application to more than one kinematic parameter.
A preview of the final results can be appreciated from the videos in figures (1) and (2).
To properly play the videos the PDF should be opened using Adobe Acrobat. If it is impossible to do so, the same videos can be found in the GitHub repository of the project: duckietown-ethz/proj-lf-adaptive.

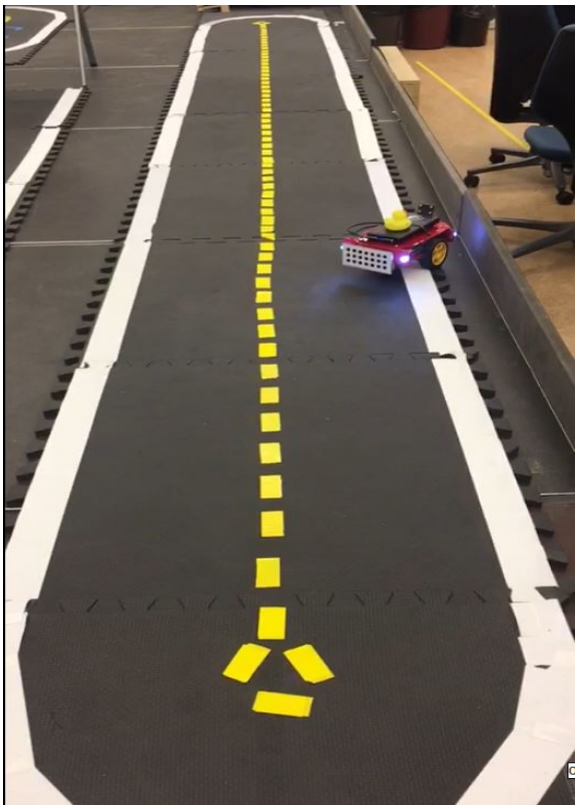

Figure 1: Before calibration using trim 0.0

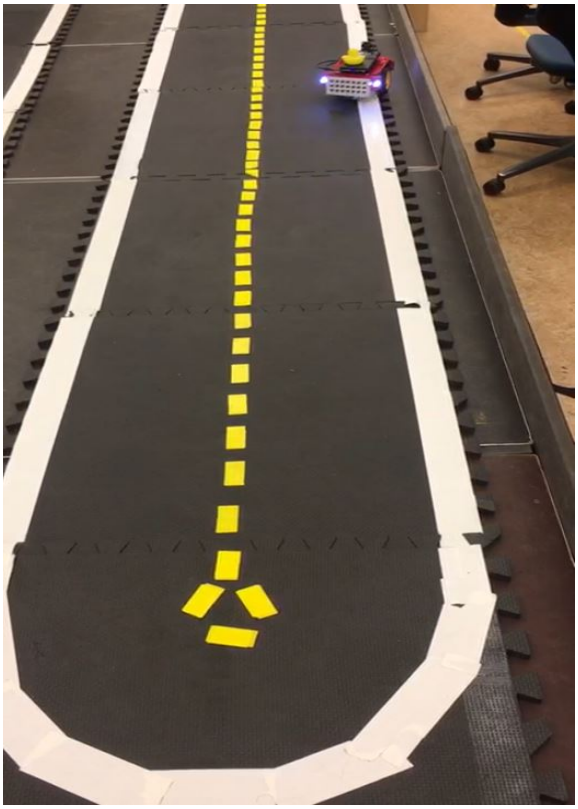

Figure 2: After calibration using trim 0.02

2 Individual contributions

This project was realized by the team *lf-adaptive*, as part of the course *Autonomous Mobility On Demand: From Car to Fleet*, offered in the fall semester 2019. The following two team members were involved and contributed equally:

- Arreghini Simone (18-949-370; arsimone@ethz.ch)
- Griffa Pietro (18-951-285; griffap@ethz.ch)

A special thanks goes also to the supervisor of the project, Jacopo Tani, and the two mentors, Rohit Suri and Aleksandar Petrov.

3 Mission and scope

3.1 Motivation

The mission of this project is to design an Adaptive Controller that allows to estimate the trim of a Duckiebot while it roams around the city, possibly getting rid once and for all of the manual wheel calibration procedure. This would allow to maintain consistent performance of the system in the presence of uncertainty and variations in plant parameters (i.e. if the bot hit a bump or accidentally taking a hit while being carried around). The problem was decided to be approached through the design of an Adaptive Controller, since this allows to deal with complex systems that have unpredictable parameter deviations and uncertainties, such as a Duckiebot.
More specifically and to give a better understanding of the project, the trim parameter is a percentage difference in the gain multiplied to the voltage applied to the two wheels. This is a first attempt to take into account the unbalance in the Duckiebot model. A more mathematical view of the effect of this parameter can be appreciated in the following equations (1) and (2).

$$v_r = k \cdot (g + t) \cdot V_r \cdot R \tag{1}$$

$$v_l = k \cdot (g - t) \cdot V_l \cdot R \tag{2}$$

Where v_r and v_l are the speed of the right and left wheel respectively. A proper introduction of all the variables showed here is left to section 3.4. This is just an anticipation to give the reader an initial idea, a more complete and exhaustive explanation of the above-mentioned equations will be given further in this relation.

3.2 Existing solution

No proper method to estimate the parameters of the Duckiebot is currently implemented on the platform. All the tuning is nowadays done manually once after assembling the robot and ,after that, every time necessary.

3.3 Opportunity

Having a better knowledge of the parameters of the Duckiebot's model would be an extremely powerful tool. Having access to this knowledge would indeed allow to deploy much more precise strategies for numerous tasks.
As a basis for the project, the existing control pipeline currently used on the Duckiebots was used as starting block.

3.4 Preliminaries

The final solution is based on a powerful tool, namely the Model-Reference Adaptive Controller (MRAC). This term refers to one of the two major families which are usually used to categorize the adaptive controllers. In particular, to reach the desired goal, an indirect model-based controller was used.

Definition 1 *For plants with unknown parameters,* ***indirect adaptive methods*** *are defined as control schemes which use a feedback control law complemented with an adaptation law. The goal of the adaptation law here is to estimate online the true values of the unknown plant's parameters.*

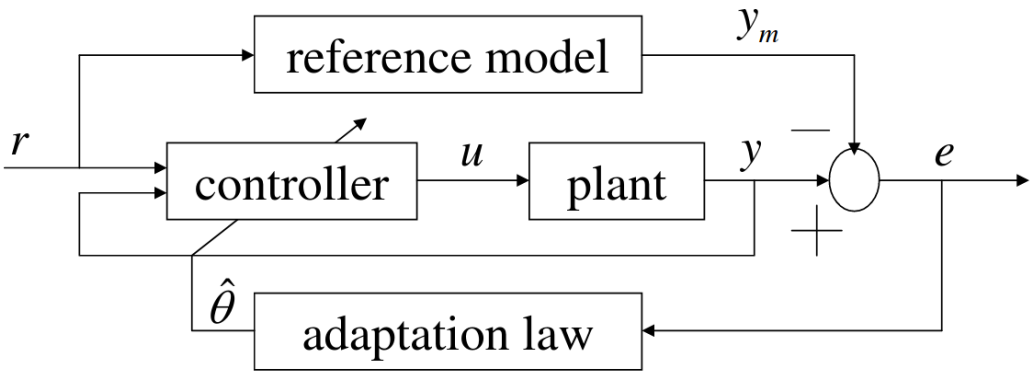

Figure 3: Model-Reference Adaptive Control

A standard Model-Reference Adaptive Control pipeline involves these elements:

- *Plant:* has a known structure but parameters are unknown;
- *Reference model:* specifies the ideal response y_m to the external command r;
- *Controller:* is parametrized and provides tracking;
- *Adaptation law:* is used to adjust parameters in the control law.

Since the main target of this application is the trim, it turns out working with just the kinematics model of the bot is enough.
In this sense, the Duckiebot is a perfect example of differential drive robot, whose kinematics is defined by the simple equations:

$$v = \frac{\dot{\psi}_r + \dot{\psi}_l}{2} \cdot R \tag{3}$$

$$\omega = \frac{\dot{\psi}_r - \dot{\psi}_l}{2L} \cdot R \tag{4}$$

Where v is the linear speed, ω is the yaw rate, $\dot{\psi}_r$ and $\dot{\psi}_l$ are the rotational speed of the right and left wheels respectively, R is the wheel radius and $2L$ is the baseline (distance wheel-to-wheel).

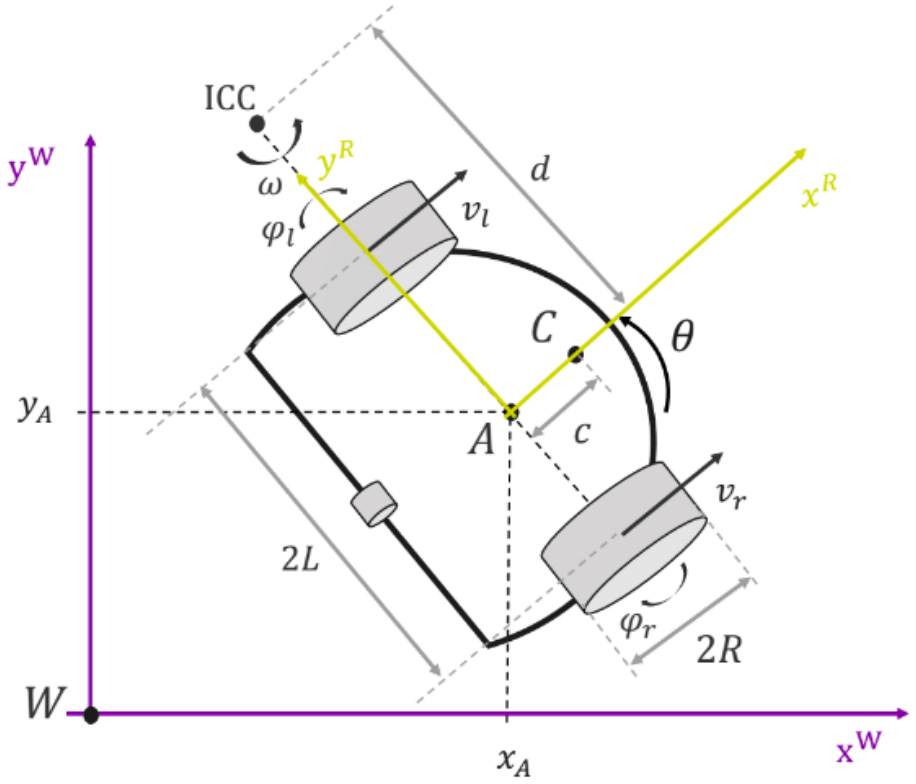

Figure 4: Model of the Duckiebot as differential drive robot

Introducing also the model of the motors, equations (3) and (4) become:

$$v = \frac{k \cdot (g + t) \cdot V_r + k \cdot (g - t) \cdot V_l}{2} \cdot R \tag{5}$$

$$\omega = \frac{k \cdot (g + t) \cdot V_r - k \cdot (g - t) \cdot V_l}{2L} \cdot R \tag{6}$$

Where now k is the nominal motor constant, g is the gain, V_r and V_l are the voltages sent as inputs to the right and left motors respectively, and t is the trim, which accounts for the asymmetry between the two motors.

4 Definition of the problem

4.1 Logic architecture

The main goal of this project would be to be able to start the lane following on an uncalibrated Duckiebot, leave it roam around the city, and find it calibrated after a short time.

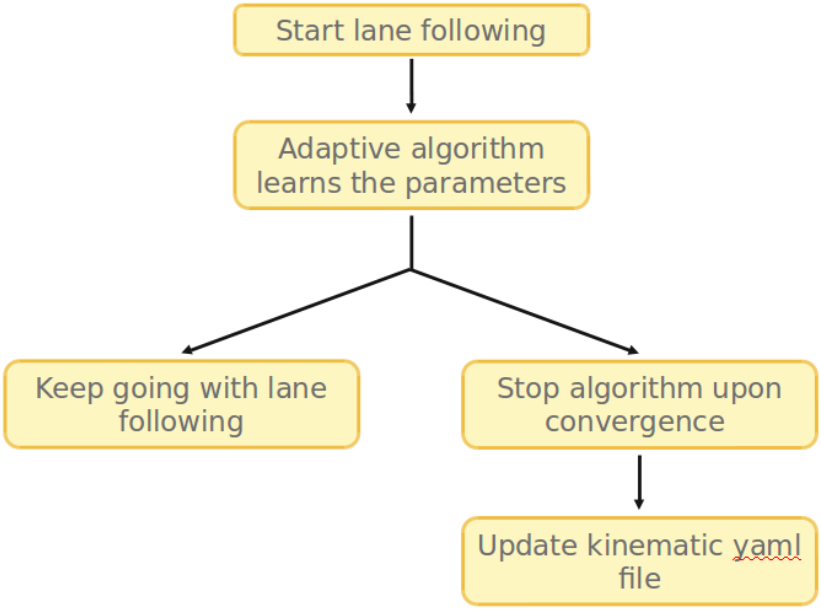

Figure 5: Logic architecture

To achieve this result a Model-Reference Adaptive Controller, such as it was introduced in section 3.4, was used. The core of the design process was finding a correct adaptation law and a reference for our application. As reference r it was decided to take the angular velocity computed by the existing PI controller implemented on the Duckiebots. This choice was done based on the fact that this input should maintain the bot reasonably in the lane, and moreover this also provide a signal that is *"as rich as possible"* given the structure of the road in which the bot is moving. The richness of the reference signal is essential to guarantee the convergence of the unknown parameters to their correct value, and as such this was a critical point to take into consideration in design process.
To come up with a proper adaptation law (7) it was first tested the common solution to consider the dynamic of the adaptation as simply proportional to the error e, defined as the difference between expected and real response of the system. This solution turned out

to work well in our application, if used together with an "additive" parameterization of the control law (8).

$$\dot{\theta} = -\gamma \cdot e \tag{7}$$

$$u = r + \theta \tag{8}$$

The form used to correct the yaw rate to take into account the uncertainty on the parameter, accounts for the effect that the trim naturally have on the system (it represents the asymmetry of the left and right motor).

To compute the expected pose at every time step, the kinematics of the system was integrated using a second order method (Runge Kutta), using as Δt the difference of the timestamp between previous and actual pose estimate:

$$\phi_{k+1} = \phi_k + \omega \cdot \Delta t \tag{9}$$

$$d_{k+1} = d + v \cdot \Delta t \cdot \sin\left(\phi_k + 0.5 \cdot \omega \cdot \Delta t\right) \tag{10}$$

By comparing this expectation with the actual pose coming from the video feedback, it is possible to compute the error as:

$$e = y_p - y_m = \begin{bmatrix} e_\phi \\ e_d \end{bmatrix} \tag{11}$$

Having e it is then possible to update the internal state of the Adaptive Controller just by forward integrating its dynamics (7), and use this to correct the yaw rate as expressed in (8):

$$\theta_{k+1} = \theta_k + \dot{\theta} \cdot \Delta t \tag{12}$$

This procedure is then reiterated until θ reaches convergence, which can be considered achieved when the difference of the maximum and minimum value of a buffer of its value at previous time steps is below a certain threshold.

Finally, upon convergence it is possible to compute a correction factor for the trim:

$$t_{correction} = \frac{-\theta \cdot baseline \cdot gain}{2 \cdot v_{bar}} \tag{13}$$

$$t_{corrected} = t_{original} - t_{correction} \tag{14}$$

4.2 Assumptions

For the system to work as intended, two assumptions are necessary:

Assumption 1 *The feedback about the lane pose should be reasonably accurate at every time step, meaning that it is required that the camera is correctly calibrated beforehand. This is a standard assumption for most of the Duckiebots' applications, since the camera is the only sensor mounted on the bot.*

Assumption 2 *The starting guess of the trim value should be "good enough". This is necessary to let the Duckiebot successfully navigate through the city even the first moments when the lane following is launched: if the trim is initially set to a completely unreasonable value the bot will most probably go outside the lane before the Adaptive Controller can even reach a proper value for the correction.*

Regarding *Assumption* 2, it must be noted that a correct value for the trim is typically a small number around zero. It was indeed observed through different runs that a starting guess of 0 for the trim value works fine. Alternatively, if one has already some knowledge of the Duckiebot's behavior (i.e. it is clearly biased on the right/left), it is possible to use a smarter value as initial guess, such as -0.05 or +0.05, to allow faster convergence.

4.3 Performance metrics

To evaluate the performance of the final system, two aspects had to be taken into consideration:

- *Adaptive part:* since the main goal is to identify the unknown parameter on the fly, it is of the outmost importance to verify that the algorithm converges as best as possible to a nearly optimal value. Through trial and error it is possible to manually tune the trim parameter to find a good value for the trim, such that the Duckiebot goes reliably straight when manually inputted to go straight. This manual wheel calibration, which is at the moment the only way to properly calibrate the robot after the initial setup, allows to find a reference value for the trim with which the final result of the designed algorithm can be compared against to valuate performance of the identification part.
- *Controller part:* it is also important that the developed system also works properly as a controller. The Adaptive Controller should refine the car command computed by the PI controller and allow the Duckiebot to navigate through the city also when it is not properly calibrated. It is then necessary to verify that a bot equipped with this system can roam around for a good time without crushing outside the lane.

5 Added functionalities

What was previously discussed in theory, practically translated in the implementation of a new node, which will be called *adaptive_controller_node*, designed to be integrated into dt-core. To make the new controller work properly, some changes to the existing code were also necessary, as it will be explained in the next section. For prototyping purposes a simulator was implemented too, on which the new controller was tested before trying on a real Duckiebot.

All the work done is collected in the apposite Github repository duckietown-ethz/proj-lf-adaptive. All the code is described through comments in more detail in the respective files. Furthermore, the README gives instructions on how to reproduce some of the results and on how to run the demo.

5.1 Simulator

To test the initial various possible control architecture a simulator was also built. This implements the control pipeline normally running on a Duckiebot (Proportional Integral architecture) without taking into account the visual feedback component, which is replaced by a simulated noisy knowledge of the pose along the track together with an imposed latency. The simulator was coded in *MATLAB*, for the sake of easiness of implementation of new functionality and representation of the final results. All the relative code is collected in the *matlab_simulator* in the above-mentioned *GitHub* repository.

5.2 Preliminary work

Before diving into the Adaptive Controller itself, it was necessary to introduce some minor modifications to three packages already present in the *daffy* release of dt-core. In particular the packages that were modified were *lane_controller*, *lane_filter* and *line_detector*.

On the line of what was done for the Classical Robotics Architectures (CRA) second exercise and the results observed, it was decide to introduce two more tunable parameters in the original code: *matrix_mesh_size* in the *lane_filter_node*, and *segments_max_threshold* in the *line_detector_node*. Being the aim in this case to maximize the accuracy of the lane pose feedback while maintaining the frequency of the system high enough to guarantee good response from the system, these variables were finally set to a default value of 0.9 and 15 respectively. A big positive impact on the controller performance was also due to the introduction of the Bayesian prediction step in the lane filter, which was previously not properly working.

The *lane_controller_node* parameters were also subject to some fine tuning to guarantee the performances needed for the application. In particular, due to the objective of this project, it was important to make so that the Duckiebot is able, to some extent, to follow the lane even when starting with a trim value far from its optimal value. To obtain the best robustness from the controller, it was identified a value of -2.8 for the proportional gain of the displacement k_d (compared to the original -3.5). The main contribution in the *lane_controller_node*, though, was the introduction of the dynamically editable parameter *lane_controller_node/ac_on*, that works as a switch for the Adaptive Controller:

- *ac_on = false:* the system acts like the Adaptive Controller didn't exist, and the *lane_controller_node* normally publish on the topic *car_cmd*;
- *ac_on = true:* the controller publish the yaw rate it computed on a new topic *ac_rif*, which is read by the *adaptive_controller_node*.

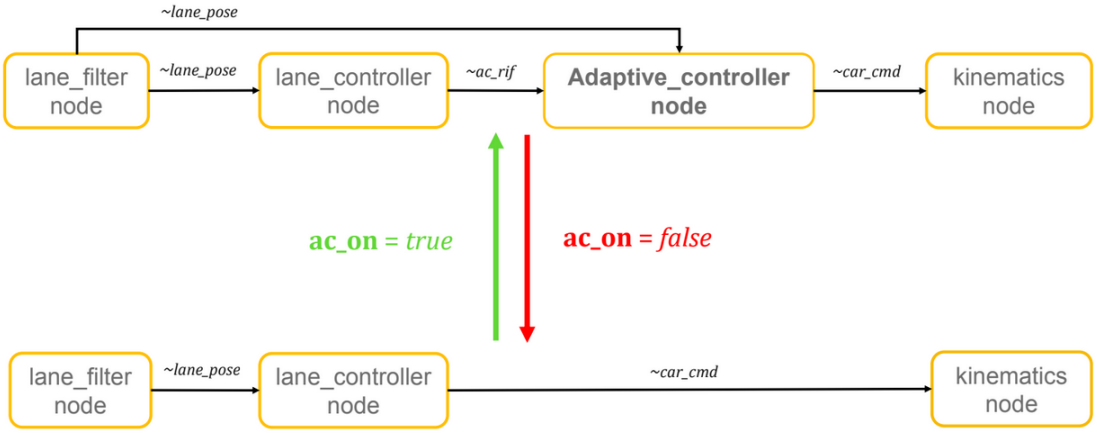

Figure 6: Switch for the Adaptive Controller

5.3 Adaptive Controller

The *adaptive_controller_node* was designed such that it can fit easily in the existing control pipeline, while introducing the least delay possible. When the node is active, it subscribes to two topics:

- *ac_rif* (msg type: Twist2DStamped), from which it reads the yaw rate computed by the PI controller, that is used as reference signal r for the Adaptive Controller;
- *lane_pose* (msg type: LanePose), from which it reads the estimate of the actual pose y_p (vector of two elements: d and ϕ) coming from the lane filter.

The the application considered it is of the out-most importance that the messages coming from the two topics are synchronized: to guarantee this synchronization, the function *TimeSynchronization* provided by the *rospy* library was used.

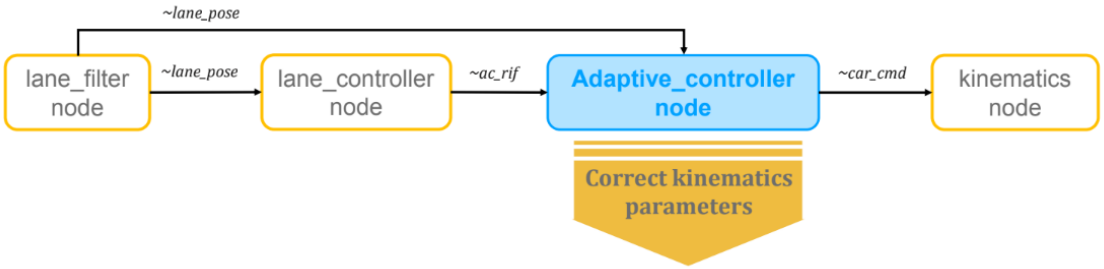

Figure 7: Synthetic rqt graph of the node

Knowing the lane pose and the twist command it would be possible to compute the expected next pose, through equations (9) and (10). But since also the time for which the yaw rate was effectively applied is necessary for this, the content of the two message is saved at every iteration and used in the following time step, when the Δt can be obtained as the difference between the time stamp in the headers of current and previous lane pose messages.
This expectation can be then compared to the current lane pose to compute the error vector, as showed in (11).
This information allows subsequently to update the internal state of the controller using equations (7) and (12), which in turn allow to correct the yaw rate to take into account the uncertainty on the trim, as shown in (8).
To check for convergence of the internal state of the controller θ, a buffer of its value in the previous time steps is kept in memory: when the difference between its maximum and minimum value is less then a certain threshold the node is shutdown.

It is important to highlight how the pose estimate coming from the lane filter proved itself to be particularly noisy along the curves: for this reason it was decided in the final design to use the feedback only on straight segments of the road, filtering these samples by imposing a threshold on the maximum yaw rate for which update the internal state θ.
This condition actually works fine to also filter out the ouliers coming from the lane filter. In fact it is fairly common than a sudden erroneous pose estimate causes the Duckiebot to abruptly oscillate for a short time: in this case too it is important that the adaptation law doesn't follows the wrong behavior.

To avoid that the Adaptive Controller runs all the time, but only when the Duckiebot requires calibration, a custom *OnShutdown* procedure was implemented. This is called

whenever the algorithm reaches convergence, and allow to save the updated trim, corrected with the method showed in equations (13) and (14), into the yaml file containing the kinematics parameters (*lane_controller / config / lane_controller_node / default.yaml*) before actually shutting down the node and giving back control to the keyboard controller. From here it is possible, if wanted, to start again the standard lane following, now on a calibrated bot.

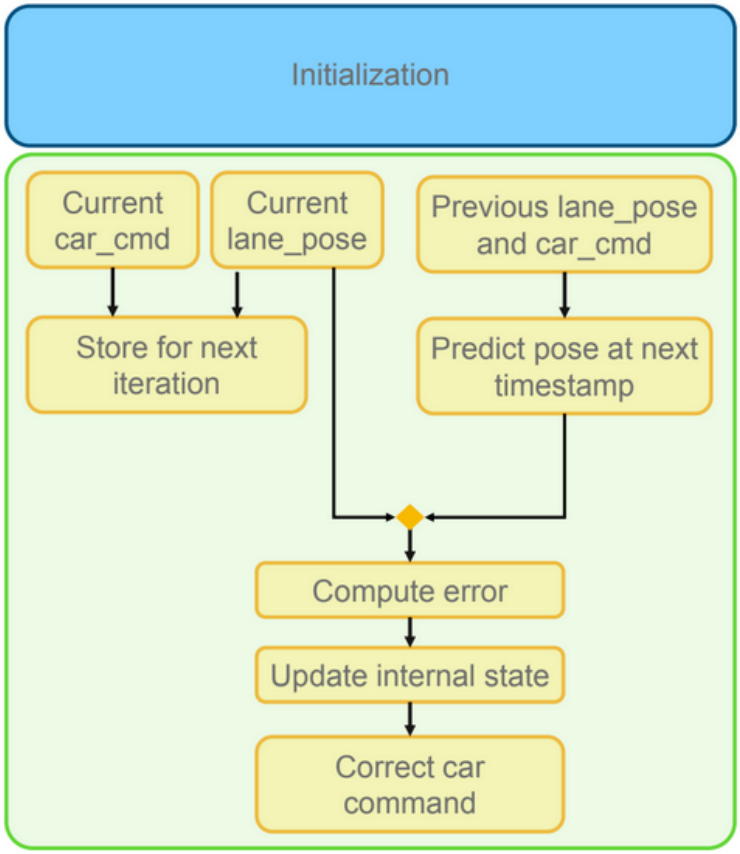

Figure 8: Compact scheme of the node

The node finally publish on two topics:

- on *car_cmd* (msg type: Twist2DStamped) it publishes the corrected car command, which is subsequently converted in inputs to the motors by the *kinematics_node*;
- on *joy* (msg type: Joy) it publish the stop command during the shutdown procedure, to give control back to the keyboard controller.

6 Performance evaluation

6.1 Expectation

Based on the results obtained by the runs on he simulator, the algorithm devised proved itself to be quiet robust. In particular, testing on map #2, the most complete one, the adaptive controller showed the following performance in the simplest setting:

real trim	-0.1
initial guess	0.0
sampling time	0.1 s
variance noise on d	0.0 m
variance noise on ϕ	0.0 rad
estimated trim	-0.0983
time to convergence	17.2 s

Table 1: Simulation 1

With an added noise on the pose measurement (that simulates the non exact feedback provided by the vision pipeline on the real system), the algorithm still converged with good performances:

real trim	-0.1
initial guess	0.0
sampling time	0.1 s
variance noise on d	0.0005 m
variance noise on ϕ	0.03 rad
estimated trim	-0.1
time to convergence	33.8 s

Table 2: Simulation 2

From other tries it was also showed that the Adaptive Controller is able to adapt itself in case of sudden change of the nominal value of some of the parameters, quickly converging to a new correct value.
To ensure robustness before moving the algorithm on the real system, the simulation was also run at pessimistic frequencies: the robot could successfully follow the track while learning a correct value for the trim until a frequency of around 6.667Hz (sampling time of 0.15s),which is significantly slower than the real system, which is usually running between 10 and 13 Hz. Using the same data from simulation 2, the simulation with $Ts = 1.5s$ converged to the correct trim value of -0.1, even though slower than the previous example: it took slightly less than 200s.

6.2 Results

To test the algorithm on the real system, two different Duckiebots (with different nominal trim) were started in lane following mode, with the Adaptive Controller switched on, different times, every time with a different starting guess for the value of trim. As testing bench it was used a custom-made track consisting in 4 straight road tiles and two U-curve tiles: this was done with the intent of reaching as fast as possible convergence of our algorithm, in a very reproducible setup. The choice of a long straight segment is due to the fact that the designed algorithm can't reliably update the internal state along curves, and as such convergence is reached only after travelling for a certain length on straight segments of road.
For completeness, test were conducted both using the error on the displacement d or on the yawn angle ϕ. On this regard it is important to note that from a previous study on the repeteability and accuracy on the pose estimate coming from the lane filter node it has been proved that generally the displacement measurement is less prone to noise. It is in fact been observed that the Adaptive Controller here implemented reached convergence using both errors, but using the error on ϕ it took longer to satisfy the convergence condition due to oscillations in the visual feedback.

The used Duckiebots had the following characteristics of interest (where the optimal trim refers to the best value found through manual calibration):

name	**optimal trim**
paperinik313	0.015
andrew	-0.115

Table 3: Duckiebots used

For the other kinematics parameters, the default parameters were used for both the robots, namely:

baseline	$0.103\,\mathrm{m}$
wheel radius	$0.0318\,\mathrm{m}$
motor constant	$27\,\mathrm{m\,s^{-1}\,V^{-1}}$
gain	1

Table 4: Kinematics parameters

Finally, following the standard assumption used of lane following, the the linear speed was kept constant to a value of $v_{bar} = 0.23\,\mathrm{m\,s^{-1}}$ through all the tests.

Running the system different times until it automatically stopped, the following results were observed:

Andrew			
before calibration		**after calibration**	
trim	**displacement**	**trim**	**displacement**
0.0	can't go straight for one tile	-0.1236	23cm in 4 tiles
-0.2	//	-0.1151	20cm in 4 tiles
0.0	//	-0.0765	38cm in 4 tiles
-0.2	//	-0.1496	32cm in 4 tiles

Paperinik313			
before calibration		**after calibration**	
trim	**displacement**	**trim**	**displacement**
0.0	22cm in 2 tiles	0.0217	22cm in 6 tiles
0.0	//	0.004	20cm in 5 tiles
0.0	//	0.0074	26cm in 5.5 tiles
0.0	//	0.0043	31cm in 5 tiles

Due to the different nominal value of optimal trim, the two Duckiebots usually took different times to reach convergence: for the tests listed above, Andrew typically stopped the Adaptive Controller after 3 laps of the test tracks, while Paperinik313 took around 1,5 laps.

Even though the final value computed for the trim can vary from run to run, the test showed that there was always an approach to the ideal value, granting better performance in lane following once the trim was updated in the Parameter server.

6.3 Conclusions

In conclusion, the algorithm proved itself to work fairly well, not only in theory but also in practice, to identify a good value for the Duckiebot's trim while making the bot perform lane following to roam around the roads of Duckietown.
Due to many unpredictable characteristics of the road, such as small bumps due to holes or misalignment between tiles, and aspects that strongly influence the precision of the lane pose, such as lighting conditions, the estimated value is not as perfect as the one it can be found manually through trial and error. The value found with the proposed algorithm is still very close to the optimal one upon convergence, and in all tests ran, it guaranteed much better performances if compared to the totally uncalibrated Duckiebot.

7 Future avenues and development

What was addressed in this work is the automatic calibration of only the trim, but it also possible to think of the possibility of calibrating not only that but all of the kinematics

parameters of the Duckiebot.
This would be extremely useful to gain a better predictability of the behavior of the bots in a huge number of applications. As for now, all the methods for control or planning that involve knowledge of the model of the system are precluded, exactly because there is too much uncertainty on the parameters.
This problem presents several difficulties though. First, the main problem in trying to identify more than one of the kinematics parameters is the superimposition of effects. It is indeed easy to see from equations (5) and (6), that many of the parameters affects similarly the behavior of the Duckiebot. For this very reason, it is hard to identify reliably more parameters, at least without any additional measurement (i.e. on the linear velocity), or without having to force certain trajectory on the bot. In particular the idea to input to the system specific patterns of inputs to study the response, which is a classical approach in system identification, was in this project avoided as much as possible since the main goal was to calibrate autonomously while driving around the city, but for more parameters it is most probably unavoidable using this method.

8 References

[1] I.Fantoni, R. Lozano. Non-linear Control for Underactuated Mechanical Systems. 2002
[2] Master Thesis of Selcuk Ercan. 2019