Design and Implementation of a Pipeline for Fully Autonomous Driving Systems

Carlos Conejo, Yuejiang Liu, Mohammadhossein Bahari and Alexandre Alahi
(carlos.conejobarcelo, yuejiang.liu, mohammadhossein.bahari, alexandre.alahi)@epfl.ch

ABSTRACT

Mobile robots are already playing a pivotal role in our society. New innovative systems are emerging, while autonomous driving's software is evolving thanks to Deep Learning (DL). In our work, we propose a modular pipeline for autonomous systems, which can combine DL algorithms for Computer Vision (CV) with classic robust functions for Path Planning and Control tasks. We design our structure to be generic for diverse mobile robots, flexible to changes on its modules, and robust in different scenarios. As we use the Robot Operating System (ROS) and socket connections between the pipeline and our system, we create a wireless network that goes from sensors' outputs to actuators' inputs. Finally, we implement the designed environment in a Loomo Segway robot with real scenarios, testing the proposed method's robustness and flexibility.

1 INTRODUCTION

It is known that mobility has a clear tendency to remove the human factor [26, 18]. For this reason, we need to find the best possible replacement for our brain and our five senses: cutting-edge sensors, classic robotics, and Artificial Intelligence (AI) combination. The main reason for this change is to increase users' safety, reducing human-caused accidents. Other fundamental consequences include improving the vehicle's energy efficiency or reducing lost time in displacements.

We are about to face building a robust platform, which allows substituting a driver in real transportation scenarios. Consequently, it is essential first to understand how we are behaving in these situations. Considering that our system is the brain, we first require input data. We use our senses to know our surroundings; then, we process the perceived information inside our brain; and finally, following the previous steps, we actuate with our bodies.

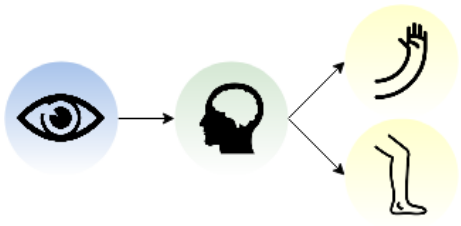

Figure 1: Human driving process.

While humans are processing the information as we show in Figure 1, robots also need to follow the same three steps, changing the senses, brain, and body by sensors, software, and actuators, respectively. In this work, we focus on the software part, presented in green in Figure 2, receiving the environment information as an input (blue) and sending the actuator commands as an output (yellow).

Figure 2: Robot driving process approach.

Pioneering work in autonomous vehicles software architecture design and implementation was done for racing by AMZ [11]. The ETH Formula Student team performed outstanding results in the competitions thanks to the robust and modular pipeline they created, suitable for modern Deep Learning algorithms. Despite this fact, their project was focused on a singular competition, so their software structure was specific. For this reason, they did not include a fundamental task for traffic scenarios: predicting how the environment will behave in the near future.

In contrast to the mentioned related paper, we aim to design and implement a pipeline adapted to different driving situations. Two central challenges exist for achieving the goal, the reliability and flexibility of the structure, and the integration in a real system.

Our main goal is to design and implement a real-time, robust and flexible pipeline, which needs to be easy to modify and suitable for diverse autonomous vehicles. We contribute with a tested open-source mobile robot's structure, improving a conventional method, based on five different pillars: Perception, Estimation, Prediction, Path Planning, and Control. In our contribution, each one has its own fixed inputs and outputs that give robustness and a variable main algorithm that allows flexibility. Moreover, every single pillar can be designed independently from the others, supporting Deep Learning functions. We design and implement the pipeline using ROS, testing it in a Loomo Segway robot, where socket enables wireless communication between the autonomous vehicle and the structure (external computer). Finally, we present and explain all details related to the code on GitHub[1].

2 RELATED WORK

To expand our knowledge and have an reasonable basis before designing the method, we present the most helpful bibliography we found for the project. We classify it depending on where they have contributed to our work.

Pipelines completely designed with AI. As AI has become a right solution for modelling complex processes in real-time, we considered related work [14, 27, 13, 2] to build the structure. In [27], a model is built with deep reinforcement learning and tested in a 3D simulation, while in [13, 2], a combination of Recursive Neural Networks (RNNs) for Prediction and Model Predictive Control (MPC) is shown.

The main advantages they give are the low sampling time of the entire pipeline (compared to separate nodes) and the accuracy and the robustness they have if they are adequately trained.

We decided not to focus on these methods due to the complexity of acquiring useful data for training and, principally, due to the lack of flexibility they present: We cannot split them in different modules. None of the mentioned related work was implemented in real systems due to its complexity and expensive computation, whereas they were only tested in simulations. As we explained before, we aim to implement all our design in a mobile robot; therefore, we consider real execution feasibility a decisive factor.

Modular Pipelines. Looking for an alternative to AI, modular pipelines for mobile robots have been studied. We analysed autonomous car [29, 4, 24, 31, 12, 25, 19, 20, 1] and Formula Student vehicle [11, 23] structures.

In [4, 24, 25, 19, 20], structures were designed and theoretically discussed, but not implemented in a real system. On the other hand, [29, 11, 23, 31, 12, 1] include modern machine learning algorithms for Perception and Prediction with an implementation section.

Prediction was not considered in the Formula Student work because it does not exist vehicle coexistence nor object motion in the competitions. Our project requires this module as it needs to be adaptable to all possible scenarios.

All other work that includes implementation is focused on a particular problem (autonomous cars). Therefore, they do not consider possible space problems in other mobile robots and, consequently, their network that connects the structure and the vehicle is wholly wired, via Ethernet and CAN. We purpose using socket to avoid these issues, connecting an external server with the robot via IP address. This wireless solution enables to test the pipeline in different mobile robots.

Algorithms. As we want to create a robust and flexible structure, we have been looking for algorithms that are only focused on a singular task. Consequently, if a user decides to change a specific module, the rest will remain invariant. At the same time, we look for algorithms that hold a low computation time, i.e. applicable to real-time scenarios.

Starting from Perception, fast human detection algorithms (PifPaf) [16] and simple object detectors [22] have been studied. They both use AI to build a model that enables a more accurate detection.

Continuing with State Estimation, Kalman Filters [30] are commonly used for robot pose estimation, when state observation is required without the presence of sensors that directly output the value. On the other hand, Simultaneous Localisation and Mapping (SLAM) [8] is usually useful to complement the state observation and to generate a map that stores all previous observations, associating data if required.

Concerning Prediction, algorithms implemented with neural networks for real-time problems have been considered, like human or vehicle predictors [15, 3].

Regarding Path Planning functions, obstacle avoidance has been prioritised in the research. Efficient methods like RRT and RRT* [17, 10] have been considered for our designed algorithm.

Finally, to build the Control pillar, research on robust classic control algorithms is made. Model Predictive Control [5] and other control techniques for non-linear systems [7] have been researched.

[1] GitHub: Autonomous Driving Pipeline in ROS [6]

3 METHOD

Before going deeper into the method, it is essential to remind that the project's primary goal is to create a robust and flexible pipeline, which can be changed partially without disturbing mobile robot's behaviour. To achieve it, we expose a ROS structure based on five different nodes, each containing fixed topics and variable algorithms.

As we show in Figure 3, the complete ROS pipeline represents what we process in our brain. It consists of five interconnected pillars: State Estimation, Perception, Prediction, Path Planning and Control.

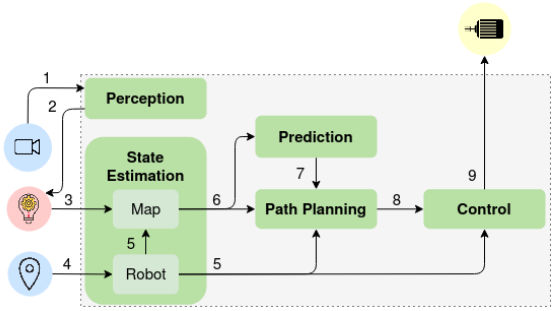

Figure 3: Purposed pipeline. We represent the main autonomous systems' components in circles, including sensors (blue), low-level software (red) and actuators (yellow). The ROS structure is drawn inside the central big gray rectangle, with its main pillars in green. Finally, numbers in connections are explained in detail in Table 1. We build all links that come in or out the ROS pipeline (1, 2, 3, 4, 9) with wireless socket.

It is important to note that we design the structure attaching importance to how it can be maximally invariant to different autonomous driving systems.

Intending to give flexibility to the pipeline, we present a group of parameters, which can be changed depending on user's needs. We decide all parameters to enable users a reasonable control over the algorithms. We provide:

- **Sample Time** (Δt): Iteration execution time of every node, equivalent to the inverse of the node rate. We include it as it gives control over the pipeline's total period. It is measured in seconds. We recommend to monitor ROS node rates and add a safety margin to ensure not to exceed never the set sample time.
- **Algorithm** ($Algo$): User can change a node algorithm by modifying the mentioned parameter. We decide to incorporate it to centralise all possible variations in a simple file.
- **Node activated**: We include this parameter to allow enabling or disabling optional nodes (Prediction and Mapping) depending on the problem requirements.

In addition to the previous parameters, we add specific ones for Prediction, Path Planning and Control, adequate to the used algorithms and taken directly from related work [5, 10, 15].

In addition to flexibility, we consider robustness as another pipeline's priority. Consequently, we present in Table 1, the principal connections between pillars, considered our structure's basis.

Topic	Description	N
Camera	Raw Image	1
Bboxes	Boxes with detections	2
P_{obj}	Detection's position	3
X_{IMU}	State given by IMU	4
X_V	Robot's estimated state	5
Map_{obj}	Map of all detections	6
T_{obj}	Predicted trajectories	7
Path	Robot's planned path	8
Control Cmds	Optimal Control commands	9

Table 1: Main pipeline's connections. Each one with a description and a label (N), represented in Figure 3.

3.1 PERCEPTION

As we use our senses to get an idea about the environment and interpret the information inside our brain, we present the Perception node. It pretends to substitute the process explained by an algorithm, which can be simple or complex depending on our necessities. We implement and explain both options to compare their advantages and disadvantages.

In this project, we use data sent by mobile robot's cameras, and we try to detect objects, animals or persons inside every frame, using neural networks. The expected inputs are raw images, which can be previously processed if needed, while expected outputs are bounding boxes that include the detection, as we expose in Table 2.

Inputs	Outputs
Camera	Bounding Boxes

Table 2: Perception's node inputs and outputs.

We implemented two different algorithms inside the ROS structure, both previously designed: a fast detector [22] and a more accurate but slower detector, Openpifpaf [16]. The fast one was trained to detect minions' images and is especially helpful for testing, thanks to its short detection time. However, the bounding box accuracy is low, and it may indirectly affect the rest of the pipeline.

On the other hand, we use Openpifpaf for human detection with high accuracy on the bounding box position. While detection time is higher, we need to consider that it causes a delay, directly affecting the data flow. Consequently, and explained afterwards with more detail, we must raise the control period as long as the autonomous vehicle's principal loop time increases. We show all parameters that allow Perception's node flexibility in Table 3.

Parameter	Description
Δt_{perc}	Perception sample time
$Algo_{perc}$	Perception algorithm

Table 3: Perception parameters.

The detection information goes back to the autonomous system, where we search the depth sensor outputs for singular ranges of the image. It returns the positions, calculated with Equation (1).

$$P_{obj} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} d \cdot cos(\psi) \\ d \cdot sin(\psi) \end{bmatrix} \tag{1}$$

Once we know the positions, we can send them to Mapping, Prediction or Path Planning, depending on our needs.

3.2 State Estimation

In order to locate ourselves, we need our position sense, known as proprioception, our motion sense, called kinaesthesia, and our memory [28]. Processing all this information, we estimate where or how fast we are.

Intending to substitute our proprioception and kinaesthesia, we use sensors such as an Inertial Measurement Unit (IMU) or wheel odometers data, which allow us to get the robot current states. We expect this sensor data as an input in our project, and we will give the estimated current position and speed as an output.

While we are also using our memory for location, we implement a Mapping algorithm that stores previous detections and access them when required. Consequently, another input is the current detection position and, storing it, we can generate a map with all past scenes. We present all inputs and outputs in Table 4.

Inputs	Outputs
X_{IMU}	X_V
P_{obj}	Map_{obj}

Table 4: State Estimation node inputs and outputs.

We also take into consideration one of our main project goals, flexibility. Therefore, we present the main parameters in Table 5, which users can set depending on their requirements.

Parameter	Description
Δt_{e_R}	Robot State sample time
Δt_{e_M}	Map State sample time
map_{active}	Map State activated
state map	State estimated by SLAM
$Algo_{Robot}$	Robot State algorithm
$Algo_{Map}$	Map State algorithm

Table 5: State Estimation parameters.

3.2.1 Robot State Estimation

The first and usually the most straightforward way of State Estimation consists of combining sensors information, considering that they may not be accurate. If we compare it to the human driving process, proprioception and kinaesthesia are considered, while memory is not necessarily used. We recommend Extended Kalman Filters (EKF) [30] for generating immeasurable variables from a non-linear model.

In mobile robots, we design the model depending on the speeds or accelerations we are expecting. For low values ($v < 3m/s$), and due to its simplicity, we usually build a kinematic model. We suggest using a dynamic model for larger speed values, which is more complex and more accurate than the kinematic.

In the project, we decide not to include EKF inside our default structure, despite enabling an option to add it. We dispose of states' information directly from sensors, which already use EKF to correct their noise. We use a state formed by three components: 2-D position and heading (x, y, θ), where x belongs to the longitudinal axis and y to the lateral. We show the coordinate system used in Figure 5 and the state structure in Equation (2).

$$X_{IMU} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{IMU} \tag{2}$$

Intending to calibrate current's state every time we turn on our mobile robot, we implement inside the State Estimation pillar an initialisation algorithm. It consists of storing the first state acquired, considering it a reference value for all iterations, as we present in Equation (3).

$$X_0 = X_{IMU}(0) = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix} \tag{3}$$

Afterwards, applying rotation and translation from the current IMU state (X_{IMU}) to the reference state (X_0), we obtain the current position and orientation respect the initial robot's situation, as shown in Equations (4) and (5).

$$R = \begin{bmatrix} cos(\theta_0) & -sin(\theta_0) & 0 \\ sin(\theta_0) & cos(\theta_0) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$X_V = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_V = R^{-1} \cdot \left[X_{IMU} - X_0 \right] \tag{5}$$

3.2.2 Map State Estimation

The second way of estimating a mobile robot's state is to consider the previous inputs, allowing an overview of the surrounding (memory). One of the most used and recommended methods is the Simultaneous Localisation and Mapping (SLAM) [8]. The algorithm is used to estimate the state depending on the position related to the environment and, besides, it allows to generate a map containing all previous observations.

As we explained in Robot State Estimation, we dispose of an Inertial Measurement Unit, which provides the information we need for estimating our current state. Consequently, in our particular problem, we only use the second part of the SLAM, Mapping.

To add memory to the mobile robot, we decide to implement a Simple Mapping algorithm that helps us better understand the environment. It consists of associating or adding perceived objects or persons to a map depending on distances related to the robot and previous detections.

We consider that *n* observations in *m* periods come from the same object if the relative distances between them during the Perception time sample has not reached a specific value ($motion_{max}$). On the other hand, we only consider all perceived objects located inside a range ($Range_{max}$). For us, the rest are noisy and not reliable. We explain in more detail how the Simple Mapping algorithm works.

Simple Mapping

1. Check if detection is inside the sensor's distance range. If the condition is True ⇒ Continue.
2. Transformation from local to global coordinates.
3. Check if detection is already on the current map.
   (a) If it had been detected previously ⇒ Update from previous to the current position.
   (b) Else ⇒ Add new detection to the map.

3.3 Prediction

In order to predict trajectories from persons, animals or objects moved by an external force, humans use prior knowledge and observations [9]. As we show in Table 6 and to substitute the complex intuition process, we need detections as inputs (P_{obj}). As outputs, we calculate future positions of the mentioned observations (T_{obj}) depending on their previous behaviour.

Inputs	Outputs
P_{obj}	T_{obj}

Table 6: Prediction node inputs and outputs.

Intending to add flexibility to the Prediction pillar, we decided to set a group of parameters, which can be modified by users depending on their constraints or needs. We present them in Table 7.

Parameter	Description
Δt_{pred}	Prediction sample time
$T_{h_{pred}}$	Prediction time horizon
N_{pred}	Number of predictions
$N_{past_{pred}}$	Prediction past observations
$Algo_{pred}$	Prediction algorithm

Table 7: Prediction parameters.

We need to find an approach for intuition, and one possible solution is using Artificial Intelligence. The main advantage is the accuracy, while we predict in a high sample time. Another more straightforward option comes from approximating the motion with a kinematic or a dynamic model. Combining both presented methods, AI and modelling represented with constraints, we achieve a better approach in terms of accuracy but still with a high sample time.

As we need real-time responses in this project, we prioritise achieving a low sampling time before getting high Prediction accuracy.

Consequently, we present a simple model based on linear kinematics, where the input position P_k is the current observation with an identifier k. The output trajectory T_k contains the current and the future positions from t to $t + T_{h_{pred}}$, being k detection's identifier-we present variables structure in Equations (6) and (7).

$$P_k = \begin{bmatrix} x_{obj} \\ y_{obj} \end{bmatrix}_k \tag{6}$$

$$T_k = \left[\begin{bmatrix} x_{obj} \\ y_{obj} \end{bmatrix}_t, ..., \begin{bmatrix} x_{obj} \\ y_{obj} \end{bmatrix}_{t+T_{h_{pred}}} \right]_k \tag{7}$$

We use the difference of positions between diverse consecutive detections to estimate the observation's present speed. We divide this velocity into two components, longitudinal and lateral, as we show in Equation (8). We decide not to consider accelerations in our model because small position errors caused high-speed variations and, consequently, the trajectory predictor estimated large moves when, in reality, the object was not moving.

$$v_k = \begin{bmatrix} v_{lon} \\ v_{lat} \end{bmatrix}_k = \frac{1}{\Delta t_{perc}} \cdot (P_k - P_{k-1}) \tag{8}$$

Using Equation (9), we replace our intuition process by a kinematic Prediction model.

$$T_k = P_k + v_k \cdot t \tag{9}$$

As we mentioned before, AI is commonly used for Prediction. For this reason, we decide to implement TrajNet++ [15], a network used for human trajectory forecasting, based on an LSTM architecture. We give the possibility of combining the Prediction algorithm with the previously presented Openpifpaf human detector. However, we want to warn users about the consequences that could have the fact of increasing the sampling time: lower Control frequency means lower admissible speed in the robot.

3.4 PATH PLANNING

While we use our prior knowledge and predicted trajectories of our surrounding to calculate the optimal path to follow, robots need a replacement for this calculation that we do unconsciously, by generating math equations and implementing search algorithms.

As we show in Table 8, Path Planning requires all previously obtained information as an input, including the vehicle's current estimated state (X_V), the map generated with all previous observations (Map_{obj}) and the predicted trajectories (T_{obj}).

On the other hand, algorithm's output consists of a set of states where we want the autonomous vehicle to be in the near future ($Path$), ensuring that it does not collide with an obstacle during its way.

Inputs	Outputs
T_{obj}	$Path$
Map_{obj}	
X_V	

Table 8: Path Planning node inputs and outputs.

Like in the rest of the pillars, we try to allow the pipeline to remain robust to user changes. Depending on user requirements, all parameters, presented in Table 9, can be changed depending on the singular mobile robot problem.

Parameter	Description
Δt_{path}	Path Planning sample time
$speed$	Mobile robot speed
$goal$	Goal coordinates (x,y)
$T_{h_{path}}$	Path Planning time horizon
$Algo_{path}$	Path Planning algorithm

Table 9: Path Planning parameters.

We consider that the understanding of the parameters is crucial, and, for this reason, we give a brief explanation. Firstly, we need to ensure that the Path Planning sample time is higher than the search algorithm maximum computation time. Secondly, the mobile robot's speed should depend on the Control's sampling time and the free space we have around the autonomous vehicle. We strongly recommend first to test the robot with lower speeds to avoid severe collisions. In addition to the last recommendation, we suggest checking if Control is driving the robot over the path because we observed that MPC could decrease its accuracy when increasing the speed. Finally, we find it essential to remind that the Path Planning time horizon must be higher than Control's time horizon while using a Model Predictive Control.

For us, Path Planning aims to avoid obstacle collision going from a start to an endpoint. We know several algorithms, such as A* (search-based algorithm), RRT, RRT* (sample-based algorithms).

A* is a search-based method that adds nodes in an ordered way and considers the distance between nodes and goal to get the shortest path. The main problem of this method is that it becomes computationally expensive when we require a long path. For this reason, we discard using search-based algorithms, and we focus on comparing sample-based ones.

On the other hand, Rapidly Exploring Random Trees methods, find the path selecting nodes randomly over the working area, with a maximum distance between nodes set by the user. It also checks obstacle collision in every connection. The main difference between RRT and RRT* is that the first one is not necessarily close to the optimal path (zig-zag pattern). In contrast, the second one is smoother and considered optimal for a large number of iterations, thanks to the trajectory restructure (rewiring). Moreover, this rewiring is the main reason why we choose RRT* algorithm for our project. We take into account that, as we have a real-time problem, we cannot iterate until finding the optimal solution. However, the path is close to the optimal and, applying external smoothness, we can obtain a good approximation of RRT* large number of iterations algorithm.

Depending on the objects, persons or animals we want to detect, we estimate a safety radius to ensure that robot and observation do not collide. We show how the Prediction RRT* (PRRT*) algorithm is working in Figure 4.

Prediction RRT* (PRRT*)

1. Get a random node, inside a delimited area and meeting the established constraints between nodes: d_{max} [m], ψ_{max} [rad] (max. node distance and max. node angle, respectively).
2. Calculate the total cost to get to the node from the start point, where $cost = t = d/v$ $[s]$.
3. Check possible collisions between the generated path and the obstacles in $t = cost$ to add Prediction:
   (a) If collision ⇒ Go back to 1.
   (b) Else ⇒ Continue.
4. Find a previously calculated neighbour node that minimises the total path cost to get smoothness.
5. Append node to the path node list.
6. Node rewiring process inside a decided radius.
   (a) If reached max. iterations ⇒ Continue.
   (b) Else ⇒ Go back to 1.
7. Calculate the angle between the current and the first desired positions and rotate all the path.
8. Improve smoothness in the sharpest nodes with a spline function and rotate back.

As we only need to avoid an obstacle if it is located in the middle of our way, we decide to check if the shortest feasible path (straight line between the goal and the start) collides with an observation or a Prediction (T_{obj} and Map_{obj}). If no collision exists, we decide to take the shortest path. Otherwise, we need to recalculate it using

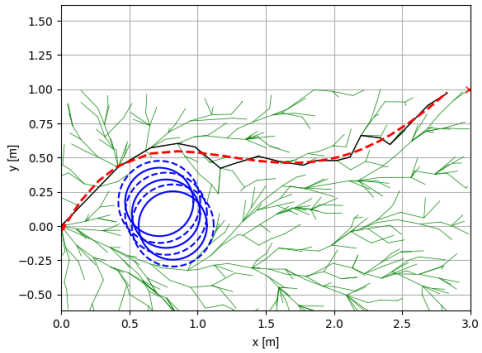

Figure 4: Robot's Prediction RRT* for Path Planning. Where obstacles are represented with blue continuous circles (their safety margin in blue discontinuous), all trees connecting generated nodes are drawn in green. The PRRT* best solution is represented in black, while the smooth PRRT* (and final desired trajectory) is drawn in discontinuous red.

the algorithm presented before. Finally, as the controller tries to follow the generated path, we need to ensure that the mentioned path is feasible and meets all vehicle restrictions and kinematic constraints (such as maximum speed, angular speed, acceleration or brake).

Smooth MPC constrained Path Planning

1. Create a straight line between the starting point and the local coordinates' goal (shortest path): $y = m \cdot x$, where $m = y_{goal}/x_{goal}$.
2. Check possible collisions between the shortest path and the obstacles (T_{obj}) in $t = cost$ to add Prediction:
   (a) If collision ⇒ Continue.
   (b) Else ⇒ Shortest feasible path. Directly go to step 4.
3. RRT* algorithm including Prediction, avoiding obstacle collision.
4. Add kinematic constraints and split it into points separated in Δt_c to get the smooth feasible MPC path.

We send the smooth feasible path to Control, ensuring that we obtain a valid solution from the controller's solver.

3.5 CONTROL

We are about to explain in details the last pillar before actuating over the system, Control. By definition, a controller's primary goal in mobile robot scenarios is to follow the desired path providing commands to a plant that reacts changing (or not) its state. In order to design a controller and get the optimal Control commands (u_i), we require the current autonomous vehicle's state ($X_V = X_i$) and the planned path ($Path_i$), as we present in Table 10.

Inputs	Outputs
Path X_V	Control Commands

Table 10: Control node inputs and outputs.

As we need flexibility in our robust pipeline, we present in Table 11, all parameters that the user could change if required.

Parameter	Description
$\Delta t_{control}$	Control sample time
n_{states}	Number of states
$T_{h_{control}}$	Control time horizon
$Algo_{control}$	Control algorithm

Table 11: Control parameters.

We considered different control methods and, for us, the main factors that help us decide for one are maximum robustness, maximum flexibility and minimum sample time. For this reason, we implement one of the most robust controls for mobile robots, which takes into consideration the kinematic or dynamic model of the vehicle (good flexibility), and that can be used in real-time. It is known as Model Predictive Control (MPC).

One of the distinctive characteristics of MPC is that it predicts the vehicle's behaviour during $T_{h_{control}}$ (future predicted states) depending on the equations designed (model). This project determines that using a kinematic model instead of a dynamic one is better because it is simpler to design and faster to compute. It is known that with low accelerations and speeds, both models behaviour is considered the same. We present our built model in Equation (10), where i is the iteration number. It is important to remark that we use the local coordinate system inside our MPC Control and that these equations with three states are only valid for a two-wheel robot.

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ \theta_{i+1} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix} + \begin{bmatrix} v_i \cdot cos(\theta_i + w_i \Delta t_c) \\ v_i \cdot sin(\theta_i + w_i \Delta t_c) \\ w_i \end{bmatrix} \cdot \Delta t_c \qquad (10)$$

Depending on the error between future predicted states and desired path (e_i) and the output variation (Δu_i), we consider that a set of control commands is good or bad. Changing these outputs, we try to minimise both variables with a solver and, in parallel, we give more importance to one than other adding weights (Q and dR) to the optimisation function (J), as we show in Equations (11) and (12).

$$Q = \begin{bmatrix} Q_x \\ Q_y \\ Q_\theta \end{bmatrix}; dR = \begin{bmatrix} dR_v \\ dR_w \end{bmatrix} \qquad (11)$$

$$J = \sum_{k=1}^{k=N} [e_i^T \, w_Q Q e_i + \Delta u_i^T \, w_{dR} dR \Delta u_i] \qquad (12)$$

MPC Solver

1. Start solver's first iteration with the N previous predicted Control outputs.
2. Get next state from the current state and Control output $X_{i+1} = f(X_i, u_i)$ for N times.
3. Get state error for every N Prediction, calculated as the difference between the predicted state and the reference state: $e_i = X_i - Path_i$.
4. Calculate the output difference between every two consecutive predictions $\Delta u_i = u_{i+1} - u_i$.
5. Store objective function value (J) to compare it with other solver iteration's result to minimise it. Output the predicted Control commands for the minimum J.

It is important to note that Q and dR parameters need to sum one and one, respectively. In the same way, weights w_Q and w_{dR} also need to sum the unit. The first pair of parameters refer to the importance of giving to every state or output rate, while the second means if we give more importance to states' error or output rate in general. For example, giving a value of $w_{dR} = 0$ would take us to have no error between the predicted states and the desired path but, at the same time, it could cause damage to the motors due to high peaks on actuator commands. On the other hand, setting $w_Q = 0$ would completely minimise the output rate, but not necessarily following the path designed beforehand.

All mentioned weights have been experimentally set and could be changed by the user if needed. We also normalise the weights dividing them by the maximum error admissible (for states and output rate) squared. This method allows us to have total control over prioritisation inside optimisation.

4 EXPERIMENTS

Unlike all previously explained parts of the method, this section is a singular detailed exposition of our implementation in a real mobile robot.

We first expose the autonomous system we use for testing, adding relevant information about complementary software that connects the robot and our method. Then, we make experiments to test how the structure behaves, making changes in some pillars.

4.1 AUTONOMOUS VEHICLE

As we informed before, we use a Loomo (Segway) robot to implement and verify that the pipeline is meeting all our goals. We choose Loomo because it allows us to focus on the pipeline's software part, avoiding the design and hardware implementation (such as sensors or PCBs). We also took advantage of the robot's previous software work [21], which made it easier to develop and test the structure in a shorter time.

Figure 5 shows the coordinate system we use in all the ROS structure, where G components are global, and L coordinates are local.

Figure 5: Loomo coordinate system. We use it in all the structure pillars. The global axis is represented in red, while the local one in blue. The heading angle (θ_G) goes from global x-axis to local x-axis.

In addition to the ROS structure, we modified an already implemented algorithm [21], which runs inside the robot (system) in real-time. The system inputs are the bounding boxes, from the previously explained detector, and Control commands, sent by the controller. On the other hand, system outputs are raw images (frames) sent to Perception, observations' positions provided to Mapping and Prediction, and IMU data to estimate vehicle's state in real-time, as we show in Table 12.

Inputs	Outputs
Bounding Boxes	Camera
Control Commands	P_{obj}
	X_{IMU}

Table 12: Autonomous Vehicle inputs and outputs.

As we explained before, we only use Loomo already integrated sensors required for topics in Table 12. 6-axis IMU, wheel odometry, RealSense RGB camera and ultrasonic sensors are used by State Estimation, Perception, Prediction and Mapping to receive information about the surroundings. At the same time, wheel motors receive optimal Control commands.

The mentioned algorithm consists of a closed-loop that receives and forwards information from Loomo sensors and actuates over the robot using some optimal commands calculated beforehand by Control. Between these two steps, it estimates the detection's positions from the detector's and depth sensor's information. We establish the connection between the main pipeline and the Loomo via Socket, due to its easy implementation and fast data transfer. We explain below the algorithm in more details.

Loomo Algorithm

1. Get an image from the camera and send it to Perception.
2. Receive bounding boxes from Perception, get depth sensor outputs in this range, transform it to $P_{obj} = (x, y)$ and send it back to State Estimation.
3. Get $X_{IMU} = (X, Y, \theta, v, w)$ from the inertial sensor and drive it to State Estimation.
4. Get optimal commands from Control.

Like all mobile robots and due to physical constraints, Loomo has its limitations, shown in Table 13.

Loomo Segway Robot	
Wheelbase	0.57 m
v_{max}	2.0 m/s

Table 13: Loomo kinematic properties.

4.2 POSSIBLE SCENARIOS

We analyse three different real scenarios and how the autonomous vehicle behaves in each one depending on a set of parameters. All experiments are located in the same place. Consequently, we always set the same planner's working area to $x \in [0, 3]$ m and $y \in [-1, 1]$ m.

4.2.1 NO DETECTIONS' PRESENCE

The first case of study is the easiest one, considering no obstacles or persons in the scenario. It is important to note that PRRT* is only activated when the mobile robot finds an obstacle on its way, so this section's results can be extrapolated to all other scenes. Consequently, as we show with a discontinuous red line in Figure 6, we take the shortest feasible path between start and goal: A straight line that includes Loomo's kinematic constraints.

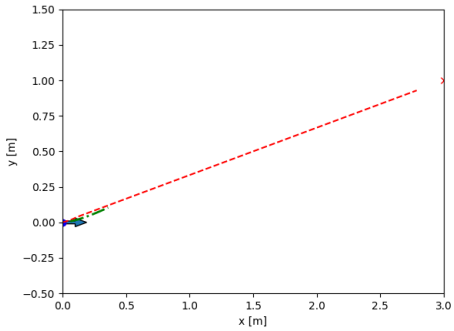

Figure 6: Robot clean environment's analysis. We represent in discontinuous red the desired path for the Loomo, while in discontinuous blue, we plot the predicted robot's positions if we apply the MPC optimal Control commands calculated. The blue arrow represents Loomo's position and orientation.

As the scenario is the simplest we can find, we use this experiment to get the best set of Control parameters for the Loomo and see how it reacts to goals that require a variation in the heading. We ensure that there is no presence of obstacles in the path; therefore, we can disable Mapping and Prediction nodes, detecting with the simplest algorithm available (Minion). We present the Loomo fixed parameters for this scenario in Table 14.

Parameter	Value
Prediction activated	False
Mapping activated	False
Detector	Minion

Table 14: Loomo fixed parameters for no detections' experiment.

In our project, we considered local state equations in the MPC to facilitate our parameter calibration, ensuring that x and y are strictly following the local coordinate system presented in Figure 5.

Consequently, if Δerror is low, x error is associated with the speed error (the difference between speed parameter and current speed) and y error is related to the horizontal error, which is crucial to avoid object collision. Thus, we decide to give a higher dR weight value, verifying that the robot's orientation is still similar to the desired.

We take care of actuator input rates, as we know that high variations could cause an actuator failure. Thanks to dR weights, we can avoid the presence of sharp signals inside the motors.

We modify the speed to evaluate how the mean squared error (MSE) between the desired states (Path Planning) and the estimated states (robot states) is varying. Depending on these three values, we decide the best set of weights Q and dR for the MPC Control. We present the best results of the experiments in Tables 15 and 16.

MSE	v=0.25 m/s	v=0.5 m/s
x [m]	0.026	0.049
y [m]	0.018	0.054
θ [rad]	0.084	0.137

Table 15: Loomo states mean squared errors.

The mean squared error between the desired path and the estimated state is always smaller than 6 centimetres for x and y-axis. In the heading (θ), the MSE is lower than 8° when the robot's speed is 0.5 m/s.

Δu	v=0.25 m/s	v=0.5 m/s
v [m/s]	0.055	0.135
w [rad/s]	0.004	0.004

Table 16: Loomo rate of Control outputs.

We display the result of this set of weights in Figure 7. The error is the difference between the red trace (already followed by the vehicle) and the straight line that connects the start (0, 0) and the goal, represented with a red cross.

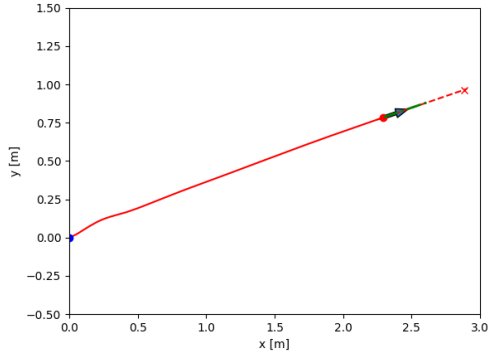

Figure 7: Robot's trajectory without detections' presence. We represent the trace followed by the robot in a continuous red line. Other elements plotted were previously explained in Figure 6.

It is important to highlight that the error is concentrated in the first part of the trajectory, when the robot is correcting its heading.

4.2.2 PRESENCE OF STATIC OBJECTS

The second scenario we are about to face consists of detecting objects, located in the middle of the shortest path between the start and the goal. We aim to find if the algorithms designed are appropriate for obstacle collision avoidance.

The scene consists of 3 static obstacles, represented by minion images and located in a singular position relative to the Loomo's initial starting point. This particular case requires RRT* Path Planning to avoid a collision, as we present in Figure 8.

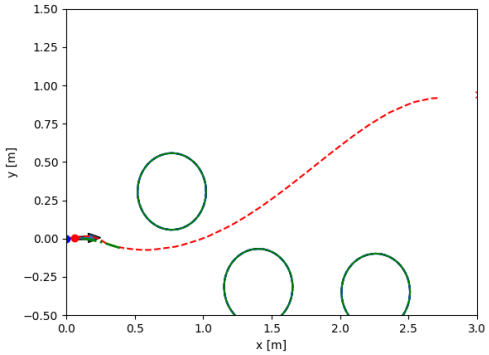

Figure 8: Robot's RRT* Path Planning avoiding static obstacle's collision. Observations are represented with green circles, while other plot elements are explained in Figure 6 footnote.

On the other hand, we do not need the Prediction node, as we consider observations completely static. We use the default detector, where minion images replace the objects. It is important to note that the mentioned detector effectiveness depends on the light and, for this reason, we decide to enable Mapping and store all previously seen objects. In Table 17, we show all fixed parameters required for the experiment.

Parameter	Value
Prediction activated	False
Mapping activated	True
Detector	Minion

Table 17: Loomo fixed parameters for observations' experiment.

Once the robot goes through the obstacles zone, the desired trajectory corresponds to the shortest one if the path is clear until the goal. Following the example of Figure 8, we see how the Loomo reacts to the commented scenario in Figure 9.

After we check that the robot is in the set goal position, we repeat the experiment 20 times with the exact object positions to verify that Loomo algorithms are robust.

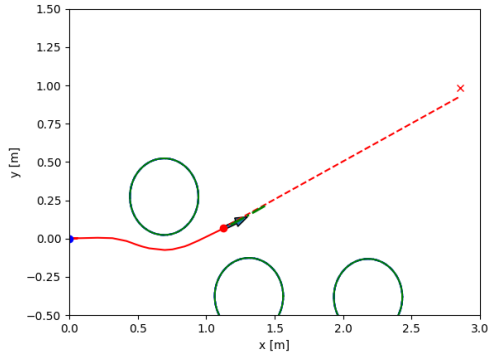

Figure 9: Loomo's trajectory once obstacle zone is crossed.

We show the repeatability test results in Table 18, where objects 1, 2 and 3 are the observations ordered in the x-axis. We also calculate in Table 19 the distances of the different calculated paths for every experiment to see if variability exists between them, adding in observations if the robot goes to the right or the left relative to object 1.

Object	1	2	3
Collisions	2	0	0

Table 18: Repeatability test results for static detections for collision avoidance. We acquire data from 20 runs.

RRT*	Minimum	Maximum	Average
Length	3.36 m	3.88 m	3.52 m
Side	Right	Left	Right

Table 19: Repeatability test results: RRT* planning. Where length is the total distance travelled by the mobile robot and the side refers to what direction took the robot (right or left relative to the person).

4.2.3 PRESENCE OF PERSONS

The last experiment contemplates the most challenging scenario for a mobile robot: human interaction. The scenario can be extrapolated to animals or other dynamic objects, such as vehicles. Our goal is to verify robustness and flexibility in the pipeline, modifying the algorithms, enabling and disabling nodes, and checking that robot's behaviour is the expected.

In this case, we need a more complex detector to locate persons in every frame. As we could have more than one person in the scene, we require Mapping to associate two different frames' observations. We also need to add Prediction to foretell the trajectory of every person in the scenario. We present the fixed parameters for the experiment with persons in Table 20.

Parameter	Value
Prediction activated	True
Mapping activated	True
Detector	OpenPifPaf

Table 20: Loomo fixed parameters for person detection with Prediction.

The experiment consists of 1 person, going from a singular starting point, (1.0, 0.2) relative to the initial robots' position, to a goal (5.0, 1.0) with a constant speed of 0.25 m/s. This person does not stop until the robot has reached its goal's coordinates.

Furthermore, Path Planning needs to apply the Prediction knowledge to avoid possible future person collision and, consequently, we apply PRRT*, as we show in Figure 10.

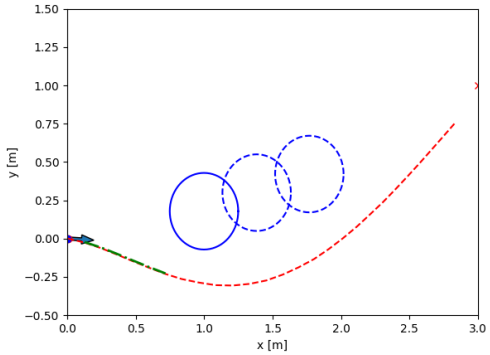

Figure 10: Robot's PRRT* with persons' presence. We plot the current detection with a continuous blue circle, while predictions are represented with a discontinuous blue circle. It is important to note that we decide how many predictions and what value of horizon time we set. In this experiment, the number of predictions is two, and the Prediction horizon time is 2 s.

Deeply analysing Figure 10, we could think that the robot did not take the shortest path. First, we need to consider the safety margin we added to objects and predictions. Moreover, it tends to go to the right direction relative to persons, due to the working area we set (according to the experiment's space).

Like in the last scenario, we create a repeatability test to see how the robot behaves while driving autonomously twenty times inside the same scene.

We check the robot-human collision number and the desired path's length to compare values with the object's presence ones. We give all results in Tables 21, 22.

As we aim to find robustness and flexibility, we repeat the same experiment without the Prediction node. We summarize new fixed parameters in Table 23.

Person	1
Collisions	2

Table 21: Repeatability test results for human collision avoidance using Prediction. We receive data from 20 runs.

PRRT*	Minimum	Maximum	Average
Length	3.48 m	4.27 m	3.9 m
Side	Left	Right	Right

Table 22: Repeatability test results in human detection: PRRT* planning. Where length corresponds to the total distance driven by the mobile robot and the side refers to what direction took the robot (right or left relative to the person).

Parameter	Value
Prediction activated	False
Mapping activated	True
Detector	OpenPifPaf

Table 23: Loomo fixed parameters for person detection without Prediction.

We present the collision and RRT* length results in Tables 24 and 25.

Person	1
Collisions	4

Table 24: Repeatability test results for human collision avoidance without Prediction. We receive data from 20 runs.

RRT*	Minimum	Maximum	Average
Length	3.57 m	4.16 m	3.86 m
Side	Left	Right	Right

Table 25: Repeatability test results in human detection: RRT* planning without Prediction.

In order to compare all results easily for robustness and flexibility, we show all of them in Table 26.

Evaluation	RRT* (Object)	PRRT*	RRT* (Person)
μ Length [m]	3.57	3.9	3.86
σ Length [m]	0.15	0.23	0.18
N Collisions	2	2	4

Table 26: Test results comparison: RRT* (Object) represents the object collision avoidance scenario, while PRRT* and RRT* (Person) refer to person collision avoidance.

First of all, we compare the means between the three mentioned scenarios. As we include motion in the robot goal's direction for the human experiments, the followed path is larger than in static obstacles. Applying a one-way ANOVA test, we can accept that length means obtained in person detection scenes are the same, while in object observations, they are different. Studying the standard deviation in all scenarios, we can observe how RRT* variance values are similar and lower than Prediction RRT* one.

The number of collisions does not change when we switch from object to human detection with Prediction. When we do not consider the human motion in our Path Planning, performance results are reduced. We can confirm that our pipeline is flexible and robust simultaneously, as we can switch detection algorithms without affecting the mobile robot's performance.

5 Conclusion and Future Work

We have presented the design and implementation of a ROS structure for autonomous driving systems. Thanks to our specifically designed PRRT* and MPC Control, our proposed method enables the structure's use in multiple mobile robots and diverse scenarios without disturbing the performance robustness. We have built a wireless socket connection between pipeline and platform and, consequently, we facilitate the implementation in a real system, avoiding space problems caused by wired networks. We have also designed a modular structure that allows an easy algorithms' switch in every single pillar, where algorithms could contain classic or modern Deep Learning functions if required. We have tested our method in a Loomo Segway robot, achieving outperforming flexibility and robustness results for three different real scenarios.

Future work will improve our pipeline's applications if other Deep Learning detectors and predictors are acquired. Classifying by types of objects instead of using data association would give better results in the robot's performance. SLAM implementation for State Estimation would also improve the system's behaviour, while abruptly accelerating or decelerating.

Finally, we want to highlight the limitations of the pipeline. Deep Learning algorithms can be used, but always being careful with their execution times. If sample time is too high, the control does not have enough environment's information update to perform successfully. We also find it essential to warn users that all kinematic or dynamic models need to be readapted if the autonomous system changes.

6 Acknowledgements

The Visual Intelligence for Transportation (VITA) Laboratory supported the research reported in this project. We could acquire knowledge and material related to software and hardware in our design and implementation from the mentioned VITA Laboratory, located in the EPFL.

References

[1] Michael Aeberhard, Thomas Kuehbeck and Bernhard Seidl. "Automated Driving with ROS at BMW". In: Sept. 2015. DOI: 10.36288/ROSCon2015-900192.

[2] Sangjae Bae et al. *Cooperation-Aware Lane Change Maneuver in Dense Traffic based on Model Predictive Control with Recurrent Neural Network*. 2019. arXiv: 1909.05665 [cs.RO].

[3] Mohammadhossein Bahari and Alexandre Alahi. "Feed-forwards meet recurrent networks in vehicle trajectory prediction". In: (May 2019).

[4] Andrew Best et al. *AutonoVi: Autonomous Vehicle Planning with Dynamic Maneuvers and Traffic Constraints*. 2017. arXiv: 1703.08561 [cs.RO].

[5] E.F. Camacho, C. Bordons and C.B. Alba. *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2004. ISBN: 9781852336943. URL: https://books.google.ch/books?id=Sc1H3f3E8CQC.

[6] Carlos Conejo. *Github: Autonomous Driving Pipeline in ROS*. Jan. 2021. URL: https://github.com/cconejob/Autonomous_driving_pipeline.

[7] Warren Dixon et al. "Nonlinear Control of Wheeled Mobile Robots". In: *Lecture Notes in Control and Information Sciences* 262 (Jan. 2000).

[8] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022.

[9] Thomas Griffiths and Joshua Tenenbaum. "Predicting the Future as Bayesian Inference: People Combine Prior Knowledge With Observations When Estimating Duration and Extent". In: *Journal of experimental psychology. General* 140 (Aug. 2011), pp. 725–43. DOI: 10.1037/a0024899.

[10] Fahad Islam et al. "RRT-Smart: Rapid convergence implementation of RRT towards optimal solution". In: Aug. 2012, pp. 1651–1656. DOI: 10.1109/ICMA.2012.6284384.

[11] Juraj Kabzan et al. *AMZ Driverless: The Full Autonomous Racing System*. 2019. arXiv: 1905.05150 [cs.RO].

[12] S. Kato et al. "An Open Approach to Autonomous Vehicles". In: *IEEE Micro* 35.6 (2015), pp. 60–68. DOI: 10.1109/MM.2015.133.

[13] Hadi Kazemi et al. *A Learning-based Stochastic MPC Design for Cooperative Adaptive Cruise Control to Handle Interfering Vehicles*. 2018. arXiv: 1802.09356 [cs.SY].

[14] Reinhard Klette. "Computer Vision in Vehicles". In: *Computer Vision in Vehicle Technology*. John Wiley Sons, Ltd, 2017. Chap. 1, pp. 1–23. ISBN: 9781118868065. DOI: https://doi.org/10.1002/9781118868065.ch1.

[15] Parth Kothari, Sven Kreiss and Alexandre Alahi. *Human Trajectory Forecasting in Crowds: A Deep Learning Perspective*. 2020. arXiv: 2007.03639 [cs.CV].

[16] Sven Kreiss, Lorenzo Bertoni and Alexandre Alahi. "PifPaf: Composite Fields for Human Pose Estimation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[17] Steven M. Lavalle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. 1998.

[18] S. Liu, J. Peng and J. Gaudiot. "Computer, Drive My Car!" In: *Computer* 50.01 (Jan. 2017), pp. 8–8. ISSN: 1558-0814. DOI: 10.1109/MC.2017.2.

[19] S. Liu et al. "Computer Architectures for Autonomous Driving". In: *Computer* 50.8 (2017), pp. 18–25. DOI: 10.1109/MC.2017.3001256.

[20] S. Liu et al. "Edge Computing for Autonomous Driving: Opportunities and Challenges". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1697–1716. DOI: 10.1109/JPROC.2019.2915983.

[21] Yuejiang Liu. *Github: loomo-algodev*. Aug. 2018. URL: https://github.com/segway-robotics/loomo-algodev.

[22] Yuejiang Liu and Parth Kothari. *Github: socket-loomo*. Apr. 2019. URL: https://github.com/vita-epfl/socket-loomo.

[23] Felix Marti Valverde. "Development of Control Algorithms for a Driverless Vehicle Using ROS". PhD thesis. UPC, Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial, July 2020. URL: http://hdl.handle.net/2117/333704.

[24] Brian Paden et al. *A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles*. 2016. arXiv: 1604.07446 [cs.RO].

[25] Scott Pendleton et al. "Perception, Planning, Control, and Coordination for Autonomous Vehicles". In: *Machines* 5 (Feb. 2017), p. 6. DOI: 10.3390/machines5010006.

[26] P. E. Ross. "Robot, you can drive my car". In: *IEEE Spectrum* 51.6 (2014), pp. 60–90. DOI: 10.1109/MSPEC.2014.6821623.

[27] AhmadEL Sallab et al. "Deep Reinforcement Learning framework for Autonomous Driving". In: *Electronic Imaging* 2017.19 (Jan. 2017), pp. 70–76. ISSN: 2470-1173. DOI: 10.2352/issn.2470-1173.2017.19.avm-023.

[28] Barry Stillman. "Making Sense of Proprioception: The meaning of proprioception, kinaesthesia and related terms". In: *Physiotherapy* 88 (Nov. 2002), pp. 667–676. DOI: 10.1016/S0031-9406(05)60109-5.

[29] J. Wei et al. "Towards a viable autonomous driving research platform". In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. 2013, pp. 763–770. DOI: 10.1109/IVS.2013.6629559.

[30] Greg Welch and Gary Bishop. "An Introduction to the Kalman Filter". In: *Proc. Siggraph Course* 8 (Jan. 2006).

[31] J. Ziegler et al. "Making Bertha Drive—An Autonomous Journey on a Historic Route". In: *IEEE Intelligent Transportation Systems Magazine* 6.2 (2014), pp. 8–20. DOI: 10.1109/MITS.2014.2306552.