École Polytechnique Fédérale de Lausanne

# Design and Implementation of a Pipeline for Fully Autonomous Driving Systems

Master's Thesis

Carlos Conejo Barceló

10th January 2021

# Design and Implementation of a Pipeline for Fully Autonomous Driving Systems

Carlos Conejo, Yuejiang Liu, Mohammadhossein Bahari and Alexandre Alahi *

**Abstract**

TO-DO

## 1 Introduction

It is known that mobility has a clear tendency to remove the human factor [26, 17]. 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 important 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 important first to understand how we are behaving in this kind of situation. Considering that our system is the brain, we first require input data, where we use our senses to get the knowledge of our surroundings; then, we process the perceived information inside our brain; and finally, following the previous steps, we actuate with our bodies.

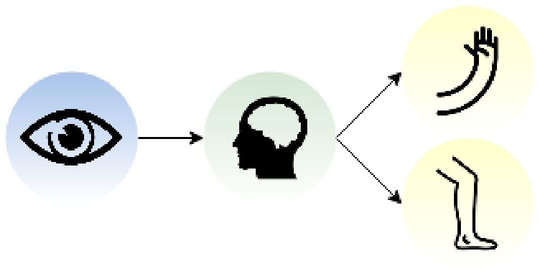
**Figure 1:** Human driving process.

While humans are processing the information as we explained above, 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, receiving the environment information as input and sending the actuator commands as an output.

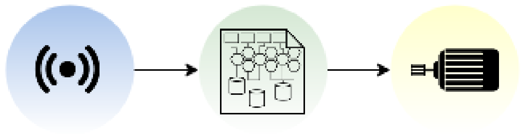
**Figure 2:** Robot driving process approach.

Pioneering work in autonomous vehicles software architecture design and implementation was done for racing by AMZ [10], where the ETH Formula Student team performed outstanding results in the competitions thanks to the robust pipeline they created.

In contrast to the mentioned related paper, we aim to design and implement a pipeline adapted to different driving situations. Two main 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. We design and implement the pipeline using the Robot Operating System (ROS) [25], testing it in a Loomo Segway robot, where socket enables the communication between the autonomous vehicle and the structure (external computer).

## 2 Related Work

To expand our knowledge and have an appropriate basis before designing the method, we present below the most

*VITA Laboratory, EPFL. Emails: carlos.conejobarcelo@epfl.ch, yuejiang.liu@epfl.ch, mohammadhossein.bahari@epfl.ch, alexandre.alahi@epfl.ch

helpful bibliography we found for the project. We classify all documents depending on where they have contributed to our work.

**Pipelines using AI**. As AI has become a good solution for modelling complex processes in real-time, related work like [13, 27, 12, 2] has been considered to build the structure. In [27], a model is built with deep reinforcement learning, while in [12, 2], a combination of Recursive Neural Networks (RNNs) for Prediction and MPC control is shown.

The main advantages they present are the low sampling time of the entire pipeline (compared to separate nodes) and the accuracy and the robustness they have if they are properly trained. We decided not to focus on these methods due to the complexity of acquiring good data for training and, principally, due to the lack of flexibility they present.

**Pipelines without AI**. To find a good alternative to AI, already created pipelines for mobile robots have been studied. First, we analysed some autonomous car [29, 4, 23, 31, 11, 24] and Formula Student vehicle [10, 22] structures. In [1, 18, 19] ROS structures were designed to communicate the global pipeline's different modules.

Division in modules has been considered, but, as one goal of the project is to design a structure for any mobile robot, algorithms need to be replaced. Path planning is applied to traffic scenarios or Formula Student competitions, whereas in this project, all types of scenarios need to be considered.

**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 certain 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) [15] and simple object detectors [21] 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) [7] 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 [14, 3].

Regarding Path Planning functions, obstacle avoidance has been prioritised in the research. Efficient methods like RRT and RRT* [16, 9] 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 [6] have been researched.

## 3 METHOD

Before going deeper into the method, it is important to remind that the project's main 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, and it consists of five interconnected pillars: State Estimation, Perception, Prediction, Path Planning and Control.

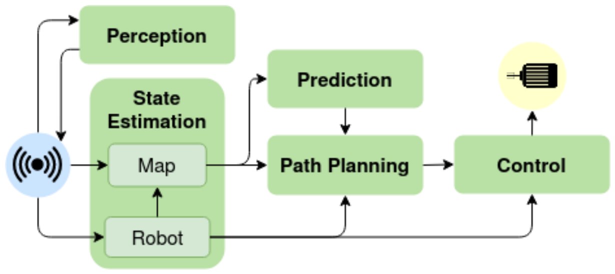

**Figure 3:** Main ROS structure pipeline.

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 show them later during the explanation of the different structure's parts.

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 |
|---|---|
| Camera | Raw Image |
| Bounding Boxes | Boxes with detections |
| $X_{IMU}$ | State given by IMU |
| $X_V$ | Robot's estimated state |
| $P_{obj}$ | Detection's position |
| $Map_{obj}$ | Map of previous detections |
| $T_{obj}$ | Predicted trajectories |
| Path | Robot's planned path |
| Control Commands | Optimal control commands |

**Table 1:** Main pipeline's connections.

## 3.1 Perception

As we use our senses to get an idea about the environment and we interpret the information inside our brain, we firstly present the Perception node, which 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 node inputs and outputs.

We implemented two different algorithms inside the ROS structure, both previously designed: a fast detector [21] and a more accurate but slower detector, Openpifpaf [15]. 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 to 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 main loop time increases. We show in Table 3 parameters that allow Perception's node flexibility.

The detection information goes back to the autonomous system, where we search the depth sensor outputs for singular ranges of the image, returning the position (1).

| Parameter | Description | Value |
|---|---|---|
| $\Delta t_{perc}$ | Perception sample time | $0.1s$ |
| $Algo_{perc}$ | Perception algorithm | Minion |

**Table 3:** Perception parameters.

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

Once we know the positions, we can send them to other pillars, 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 we are or how fast we are running in a certain moment.

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.

| Inputs | Outputs |
|---|---|
| $X_{IMU}$ | $X_V$ |
| $P_{obj}$ | $Map_{obj}$ |

**Table 4:** 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 | Value |
|---|---|---|
| $\Delta t_{e_R}$ | Robot State sample time | $0.05s$ |
| $\Delta t_{e_M}$ | Map State sample time | $0.05s$ |
| $map_{active}$ | Map State activated | True |
| state map | State estimated by SLAM | False |
| $Algo_{Robot}$ | Robot State algorithm | Initialise |
| $Algo_{Map}$ | Map State algorithm | Simple |

**Table 5:** State Estimation parameters.

#### 3.2.1 Robot State Estimation

The first and usually the simplest way of state estimation consists of combining sensors information, taking into consideration that they may not be accurate. If we compare it to the human driving process, proprioception and kinaesthesia are considered, while memory is not used. We recommend Kalman Filters (KF) [30] for generating immeasurable variables from a model.

In mobile robots, we design the model depending on the speeds or accelerations we are expecting. For low values ($v < 5m/s$) 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, but also more accurate than the kinematic.

In the project, we decide not to include Kalman Filters (KF) inside our default structure, because we dispose of states' information directly from sensors, which already use KF to correct their noise. We use a state formed by 3 components: 2-D position and heading ($x$, $y$, $\theta$). We show it in (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, see 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 present 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 (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 also 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) [7]. 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 singular 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 certain value ($motion_{max}$). On the other hand, we only take into consideration 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 [8]. As we show in Table 6 and to substitute the complex intuition process, we need detections as inputs ($P_{obj}$), and we calculate, as outputs, 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 | Value |
|---|---|---|
| $\Delta t_{pred}$ | Prediction sample time | $0.05s$ |
| $T_{h_{pred}}$ | Prediction time horizon | $2.0s$ |
| $N_{pred}$ | Number of predictions | 3 |
| $N_{past_{pred}}$ | Prediction past observations | 1 |
| $Algo_{pred}$ | Prediction algorithm | Linear |

**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 simpler 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 accuracy in Prediction. 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).

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

$$T_k = \begin{bmatrix} \begin{bmatrix} x_{obj} \\ y_{obj} \end{bmatrix}_t, \dots, \begin{bmatrix} x_{obj} \\ y_{obj} \end{bmatrix}_{t+T_{h_{pred}}} \end{bmatrix}_k \tag{7}$$

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. 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++ [14], 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 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 ($T_{obj}$). The 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 |
|---|---|
| $T_{obj}$ <br> $Map_{obj}$ <br> $X_V$ | $Path$ |

**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

| Parameter | Description | Value |
|---|---|---|
| $\Delta t_{path}$ | Path planning sample time | $0.2s$ |
| $speed$ | Mobile robot speed | $0.3m/s$ |
| $goal$ | Goal coordinates (x,y) | $(3,1)m$ |
| $T_{h_{path}}$ | Path planning time horizon | $2.0s$ |
| $Algo_{path}$ | Path planning algorithm | PRRT* |

**Table 9:** Path Planning parameters.

requirements, all parameters, presented in Table 9, can be changed depending on the singular mobile robot problem.

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 Prediction 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 important 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's aim consists of avoiding 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), while the second one is smoother and considered optimal, thanks to the trajectory restructure in every iteration. Moreover, this is the main reason why we choose RRT* algorithm for our project.

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.

**Prediction RRT***

1. Get a random node, inside a delimited area and meeting the established constraints between nodes: $d_{max}, \psi_{max}$.
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 $\Rightarrow$ Go back to 1.
   (b) Else $\Rightarrow$ Continue.
4. Find the previous calculated neighbour node that minimises the total path cost to get smoothness.
5. Append node to the path node list.
6. Calculate the angle between the current and the first desired positions and rotate all the path.
7. 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 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 (maximum speed, angular speed, acceleration, 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 $\Rightarrow$ Continue.
   (b) Else $\Rightarrow$ **Shortest feasible path**. Go to step 4.
3. **RRT* algorithm including Prediction** avoiding obstacle collision.
4. Add kinematic constraints and split it into points separated in $\Delta t_c$ 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 main 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<br>$X_V$ | 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 inside if required.

| Parameter | Description | Value |
|---|---|---|
| $\Delta t_{control}$ | Control sample time | $0.2s$ |
| $n_{states}$ | Number of states | 3 |
| $T_{h_{control}}$ | Control time horizon | $1.0s$ |
| $Algo_{control}$ | Control algorithm | MPC |

**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). In this project, we determine that using a kinematic model instead of a dynamic one is better because it is simpler to design, faster to compute and 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.

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

Depending on the error between future predicted states and desired path ($e_i$) and the output variation ($\Delta u_i$), 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).

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

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

**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 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, in 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 project.

We first expose the autonomous system we use for testing, adding relevant information about complementary software that enables the connection between 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 (sensors, PCBs, ...). We also took advantage of the robot's previous software work [20], which made it easier to develop and test the structure in a shorter time.

In Figure 4, we show the coordinate system we use in all the ROS structure, where $G$ components are global, and $L$ coordinates are local.

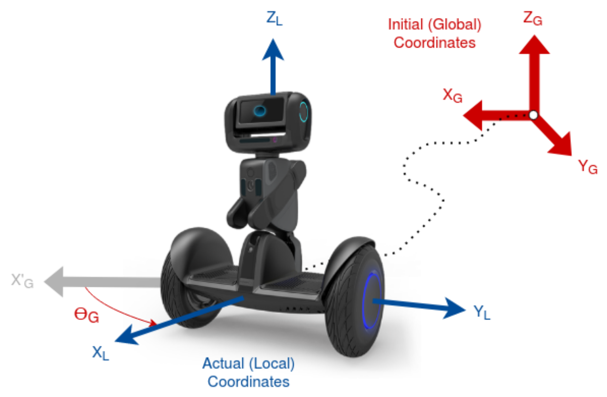

**Figure 4:** Loomo coordinate system.

In addition to the ROS structure, we have modified an algorithm that was already implemented [20], 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.

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.

| **Inputs** | **Outputs** |
|---|---|
| Bounding Boxes | Camera |
| Control Commands | $P_{obj}$ |
| | $X_{IMU}$ |

**Table 12:** Autonomous Vehicle inputs and outputs.

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 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, presented by the manufacturer, and 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

TO-DO

#### 4.2.1 No objects' presence

#### 4.2.2 Presence of objects in a closed-space

#### 4.2.3 Presence of objects in an open-space

## 5 Conclusions and Future Work

TO-DO

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

[7] 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.

[8] 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.

[9] Fahad Islam et al. "RRT-Smart: Rapid convergence implementation of RRT towards optimal solution". In: Aug. 2012, pp. 1651–1656. isbn: 978-1-4673-1275-2. doi: 10.1109/ICMA.2012.6284384.

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

[11] 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.

[12] 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].

[13] 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. eprint: https:

//onlinelibrary.wiley.com/doi/pdf/10.1002/9781118868065.ch1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118868065.ch1.

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

[15] 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.

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

[17] 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.

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

[19] 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.

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

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

[22] 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.

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

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

[25] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: vol. 3. Jan. 2009.

[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. URL: http://dx.doi.org/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.