POLITECNICO DI TORINO

Corso di Laurea Magistrale in Mechatronic Engineering

Tesi di Laurea Magistrale

Runtime Monitoring of Cyber-Physical Systems Using Data-driven Models

Relatori:
Prof. Alessandro Rizzo
Prof. Milos Zefran, University of Illinois at Chicago

Candidato:
Michele Giovanni Calvi

Dicembre 2018

Acknowledgments

I would like to express my gratitude to professors Milos Zefran and Prasad Sistla, who with their patience and expertise have helped me throughout the entire research. I am thankful for their aspiring guidance, constructive criticism and warm advice during the project work.

I wish to extend this gratitude to professor Alessandro Rizzo, who, despite the distance, was always willing to help me.

A special thanks to my family, without whom I would not have had the opportunity to study abroad, whose support and encouragement have always pushed me to strive for greatness.

Last but not least, I would like to thank all of my friends, with whom I have had the immense pleasure to share this wonderful experience.

Summary

In recent years we have seen a rise in the complexity of the architecture of physical systems, as these architectures become more complex also the correct functioning of such systems becomes more complicated. Furthermore, it is possible that an accurate model of the system is not provided consequently, complex algorithms are adopted to verify its proper functioning, which can become very complicated. The goal of the thesis is to provide a data-driven model where the model of the cyber-physical system (in this case a self-driving car) is developed through a black-box model, an LSTM neural network. This network will learn the behavior of the system by providing the states at the next time step and will use these outputs to guarantee safety protocols. This approach could be extremely significant in many applications, especially in autonomous systems which are being developed in nowadays. In these systems deep learning could be used to elaborate complicated data, furthermore a soft computing component would decrease the cost of the system.

First, data from a simulator was obtained which was used to train a neural network to generate the control of the vehicle (steering angle and acceleration), so that a proper environment for a self-driving vehicle was available. Second, once the control was developed it was necessary to verify the behavior of the vehicle. This was achieved by using a particle filter which verifies the probability distribution of the states at the next time steps and eventually comparing these states with the safety properties imposed on the vehicle. Having obtained a monitor with good accuracies it was possible to monitor a data driven model of the vehicle.

Table of contents

List of figures

Chapter 1

Introduction and Previous Work

Improvements in technology over the last years has made it possible to build more reliable and secure systems. However, with the improvements in technology we see a rise in the complexity of the architecture of physical systems, as these architecture's complexities increase so do the methods of guaranteeing the correct functioning of the systems. Unfortunately, testing these systems is not enough to verify the correct behavior so it is important to verify and monitor the system during its working operation to guarantee safe operation. This is a very cumbersome approach as it is very hard to obtain all the sequences of states that lead to a faulty behavior, furthermore it is possible that the monitoring is implemented on an inaccurate model which could lead to harmful consequences.

As previously mentioned, testing is not enough thus it is essential to check the system at run time and verify that it is working as expected. The use of this approach on cyber physical systems is quite at its early stages, however, it has been widely used in computer science applications. From [1] it is possible to see how monitors have been implemented in controlling the correct behavior of components that cannot be verified as it is not possible to access the internal functioning. A problem that arises in [2],[3],[4], where it is possible to monitor temporal properties of deterministic systems in which it is possible to measure the system state, is that monitorability is defined with respect to a single property. Instead in [5] and [1] the systems are partially observable where the systems are modeled as Hidden Markov Chains (HMC), the monitor in this case would yield the measure of the false alarms and missed alarms. Other authors deemed better to monitor hybrid automata [6],[7],[8],[9],[10] where they detected whether the automata entered a failure state, however, the authors always modeled the error modes, meaning that only the modeled errors would be monitored. Other methods have been adopted to model the systems such as [11] where Extended Hidden Markov Chains (EHMC),

have been implemented.

The research described in this thesis is an innovative approach, which is to develop the model of the system with a recurrent neural network (RNN), more precisely a Long Short Term Memory (LSTM) network and evaluate the safety of it during real time operation. Deep learning algorithms have received a lot of attentions lately, this is due to their effectiveness of solving long-standing problems in both computer vision [12] and [13] and language processing [14], they have been used in self driving cars already as shown in [15] where the LSTM is used to predict lateral and longitudinal trajectories for vehicles on the highway. There are two approaches to the proposed method, either develop a *model based* or *data driven* models. The latter assumes that only the soft computing component is accessible and the real model of the vehicle is not accessible. In the model based approach it is assumed that the model is available and the LSTM is used as a classifier between faulty and good behavior of the vehicle. In this research only the data driven approach will be studied as by doing so it is possible to use the same model for different safety properties whereas the model based approach would work only for the trained safety properties.

Chapter 2 describes the theoretical background on which the research is based, the core of this chapter relies on the descriptions of the LSTM, CNN, Bayes' Filters and monitorability. Chapter 3 focuses on giving a description of the simulator used for the research and the algorithm that has been implemented to develop the control of the self driving car. The focus of Chapter 4 is to describe in detail the monitoring algorithm and especially the particle filter implemented. Chapter 5 instead will be about the development of the data driven model with the use of LSTMs and how it behaves with the monitor implemented in the previous chapter.

Finally Chapter 6 will provide the final results, in particular the differences between the data driven model and the actual model. It will also focus on the future developments and improvements that could be made to the research developed up to now.

Chapter 2

Theoretical Background

A Cyber Physical Systems (CPS) involves interactions between the software based control algorithms and the physical components of the system in which it operates. CPSs will have physical variables that represent the realistic operation of the system, represented by differential equations, and cyber variables which provide a model of computation like Finite State Machines. The focus of the work is the implementation of the monitor on a data driven model, in order to do so it is necessary to introduce a number of concepts.

CPSs must respect desired properties in order to guarantee safety, these properties are called Safety and Liveness properties which are specified by Streett and Buchi automata. Then, a model of the system must be designed. Previously these models were implemented as HMC, EHMC, these will be described briefly so that it is possible to understand better the transition to the proposed LSTM model of this research. Furthermore, Convolutional Neural Network (CNN) will be introduced too as these networks have been used to extract the features to train the control of the self driving vehicle. Eventually, Bayes' filters will be introduced as they are used to generate the belief propagation to evaluate in which state the system most likely is in.

2.1 Safety and Liveness Properties

Safety and liveness properties rely on the reading of a possibly infinite sequence $\sigma = s_0,s_1...$ over a set S. We assume that $\forall i \geq 0$, $\sigma[0,i]$ denotes a sequence up to s_i: $(s_0,s_1,...,s_i)$. Furthermore, if we consider two sequences α_1, a finite sequence, and α_2 a finite or infinite sequence, we can define $\alpha_1\alpha_2$ as the concatenation of the two, thus, obtaining either a finite or infinite sequence depending on the sequence of α_2. To differ between the set of finite and infinite sequences are denoted by S^* and S^w

respectively. We define:

- **Walk**: a sequence of states
- **Path**: a walk with no repeated states
- **Cycle**: a path that starts and finishes at the same state

An intuitive definition of a safety property is that it asserts that nothing bad will ever happen. We say that P is a *safety property* if the following condition holds $\forall \sigma \in S^w$:

"*If for all* σ' *which is a prefix of* σ *there exists an extension* σ'' *of* σ' *such that* $\sigma'' \in P$, *then* $\sigma \in P$."

What this means is that if a sequence does not respect the safety property then there does not exist an extension to that sequence.

On the other hand, the *liveness property* asserts that something good will eventually happen. It is not know when it will happen as it is not restricted to deadlines, otherwise it would be considered as a safety property. In other words, for a liveness property ψ then, finite sequence can be extended to an infinite sequence to satisfy the property ψ.

It is important to note that in general liveness and safety properties are mutually exclusive, except when the property is the set of all infinite sequences.

2.2 Markov Chains

A Markov Chain is a stochastic model in which the sequence of possible states depends entirely on the probability of the transition between them. It is often represented as FSM, see figure (2.1), however, the transition conditions are probabilities that represent the likelihood of a certain transition from time t to time $t+1$. A MC is a model that follows the *Markov Property* [16], which refers to the memoryless property of a stochastic process:

$$P(X_{n+1} = x|X_1 = x_1,X_2 = x_2,...,X_n = x_n) = P(X_{n+1} = x|X_n = x_n) \tag{2.1}$$

Meaning that the next state depends solely on the current state and not on the ones preceding it.

Markov chains are represented by a triplet $M = (S,R,\phi)$, where S is the set of discrete states, $R \subseteq S \times S$, meaning that it represents all the possible transitions from one state to another, and ϕ is the probability function assigning probability to each element of R. If a link exists from one state to another then the probability of the transition is non-zero, additionally, all the sum of the probability transitions from a state A must be equal to 1. An example of transition could be defined by table 2.1 where we identify two states, A and B and the transitions that occur between them.

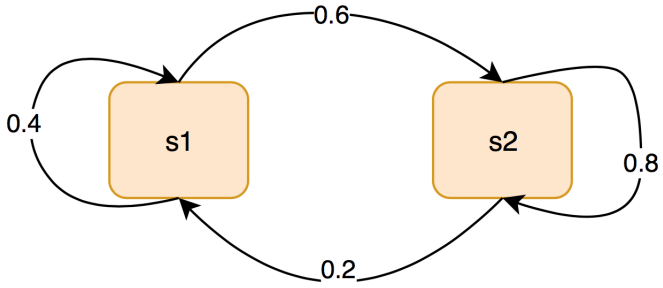


Figure 2.1: Example of a Markov Chain

Table 2.1: Transition Matrix

States	**A**	**B**
A	r_{aa}	r_{ab}
B	r_{ba}	r_{bb}

Where the transition between state A to itself is represented as:

$$(A,A) = r_{aa}$$

And the set of all transitions is represented by:

$$R = r_{aa},r_{ab},r_{ba},r_{bb}$$

The probability of making a transitions is:

$$\phi(A,B) = P(x_{i+1} = B|x_i = A)$$

Last, the sum of all probabilities of the transitions should sum up to one, table (2.2).

Table 2.2: Probabilities

States	**A**	**B**	ϕ
A	r_{aa}	r_{ab}	1
B	r_{ba}	r_{bb}	1
ϕ	1	1	

When transitions between two consecutive states exists over a whole sequence then the sequence is called path p.

This approach has been utilized to model systems, nonetheless, it is not a very effective technique because it often occurs that the states are not directly observable. However, from a series of observations it is possible to obtain the state in which the system currently is, which is exactly what Hidden Markov Chain do.

2.3 Hidden Markov Chains

The concept behind the Hidden Markov Chain (HMC) is very similar to the Markov chain, however, it provides a more realistic model as states are not directly observable but they can be estimated from the outputs of the system [17]. The HMC can be defined as a tuple $H = (M,O,s_0)$, where M is the Markov chain, recall that it is defined as the tuple described in the previous paragraph, O is the observation sequence that are given by the *output function* and s_0 is the initial state. Note, that $O(s_i)$ is the output obtained whenever the system enters the state s_i.

Since the observable parameters of the system are the outputs only, it is possible to assess the state which yielded such output. It is very likely that different states will produce the same output, hence, for every output there exists a probability distribution of the states that are likely to have generated the output measured.

With more complex systems it is hard to get an accurate model to the point that HMC might not be enough to design them. Hence, with the use of deep learning (LSTMs) it has been tried to learn the behavior of the model to try and obtain a more accurate result.

2.4 Long Short Term Memory

Our main goal is to model the temporal evolution of the system and as we have seen it is possible to do so with the HMC, nevertheless, recent years have seen a growth in the research of neural networks and especially recurrent neural networks (RNN). These are very effective when a time series of data is present, however, they have a small flaw. With long time series the RNN tends to forget and it is hard for them to remember information that is stored many steps ahead. Long Short Term Memory (LSTM) networks tackle this problem [18], this is because they are designed with gates which allows the LSTM to store only useful information and to delete the stored information when not needed [19].

As shown in figure 2.2, the LSTM produces an output o_t from input x_t and it could either work as a classification or as a regression neural network. They are extremely powerful for making predictions, which is the technique adopted in this research. Moreover, LSTMs are a type of *supervised learning*, meaning that the network will train on both inputs and outputs data. Always referring to figure 2.2 it is possible to identify the four major components that make the LSTM more powerful than the other RNN:

- *State Cell*: Also called memory cell, this cell memorizes the entire history of the network, it is eventually controlled by the forget gate and input gate on weather it should or should not learn certain parameters.

- *Forget Gate*: With the sigmoid activation function which ranges from [0,1] it is decided how much of the past information is going to be *"forgotten"*. If the output of this function is 0 then everything is forgotten, on the other hand if 1 is the output then the entire state will be remembered;

- *Input Gate*: This gate will decide what new information will be stored in the memory cell. Again the gate has activation function with range [0,1], however, this gate will be able to add information to the already existing memory but not delete it. It checks if the new input is relevant or not during the elaboration of the data. Furthermore, it includes into its own recurring layer a tanh activation function to fix the problem of the vanishing gradient for a long time steps.

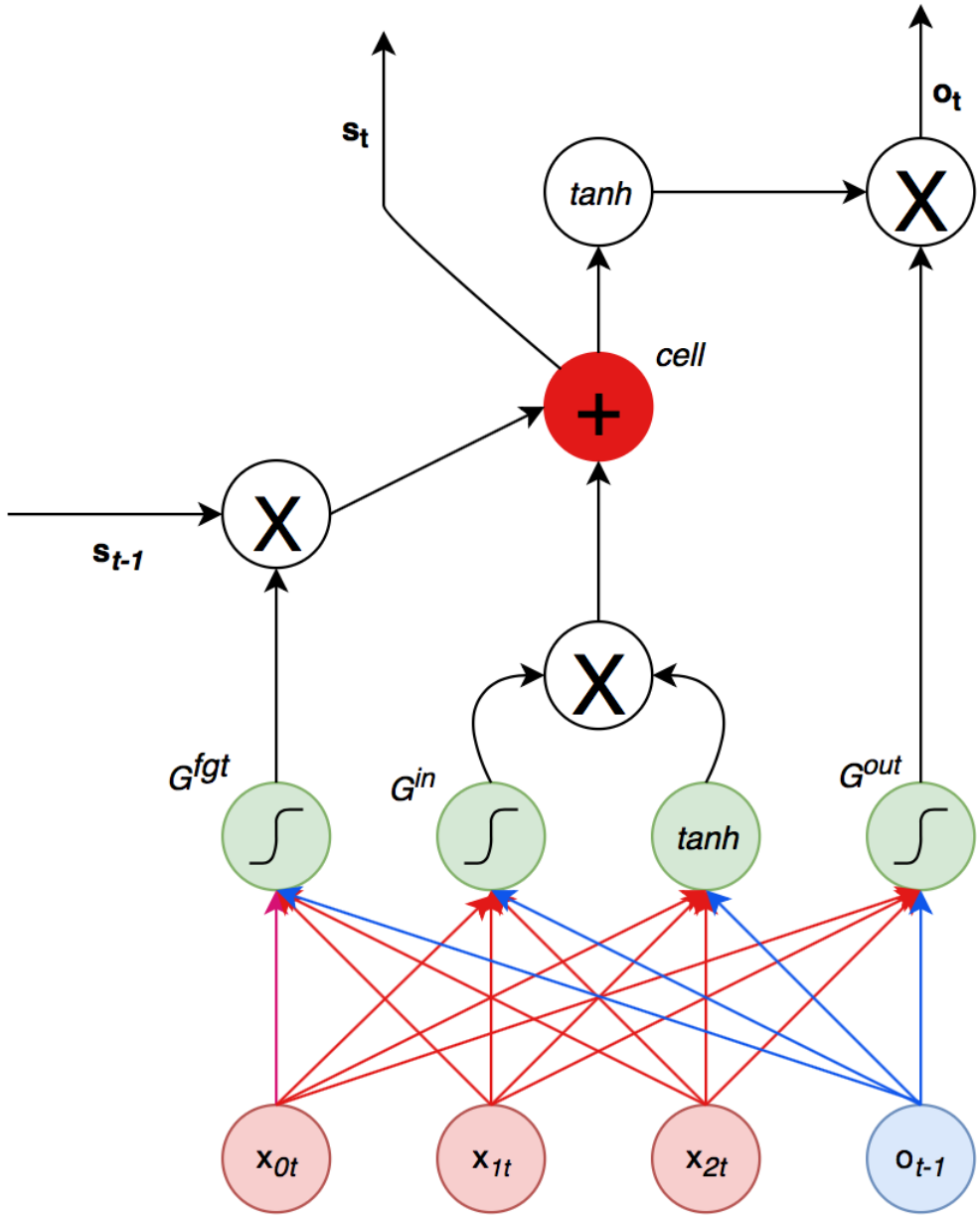


Figure 2.2: LSTM Network
LSTM recurrent network with an input with three features. The forget, input and output gate are called G^{fgt},G^{in} and G^{out} respectively, each having a sigmoid activation function and the memory cell marked in red.

- *Output Gate*: This cell evaluates which information will be passed to the next memory cell at the time step t_{i+1}.

2.5 Convolutional Neural Networks

Convolutional neural networks (CNN) are deep networks mainly used for the classification, clustering and feature extraction of images. As their name imply they perform the mathematical operation of the convolution. The convolution is the integral that expresses the amount of overlap between a shifting function g over a function f resulting in a final function. The convolution is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.2}$$

In the image analysis of the CNN the fixed function would represent the input image to be analyzed while the mobile function would be a filter or kernel as it studies the image by extracting its features.

Images can be represented by a matrix of pixel values, they can be represented as a 3-D matrix when the red, blue and green (3 channels) components are present or a 2-D matrix if the image is in the grayscale (1 channel) format. In both cases the pixels are in a range [0,255]. For simplicity the CNN will be explained for the grayscale format only, where 0 represents black and 255 indicating white.

In figure 2.3 an example of convolution over a 3×3 image is performed. The kernel slides over the original image and performs the dot product between the elements of the two matrices and maps the result on a feature matrix. Note, that for a three channel image the convolution will be performed on every channel and there is a different filter for every channel which learns a different feature.

When working with big images the feature maps can be very big, and it might get features that are not really necessary for the image analysis. Thus, it is possible to use the *spatial pooling*, which reduces the dimension of the feature map without loosing the important features. The different pooling techniques are the following:

- **Max Pooling**: Get the maximum value in a window of a feature map;
- **Average Pooling**: Get the average value in a window of a feature map;

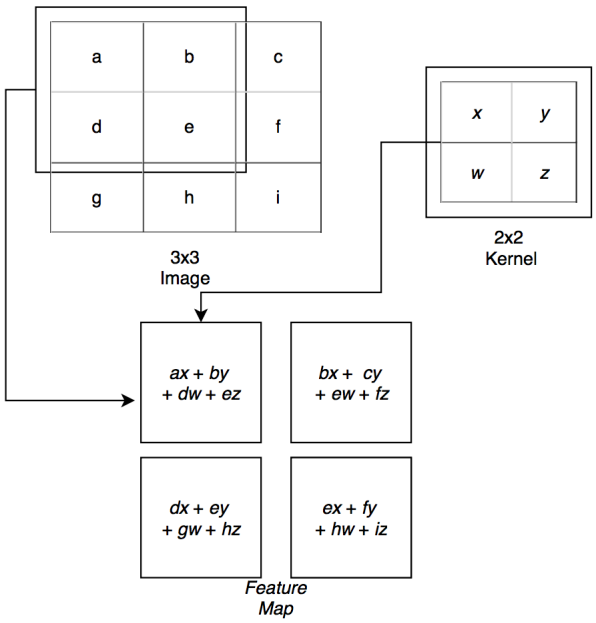


Figure 2.3: Example of convolution with a 2x2 kernel

- **Sum Pooling**: Get the sum in a window of a feature map

Where max pooling is considered as the one which delivers the best results. An example of max pooling is shown in figure 2.4.

2.6 Automata

The automaton is machine that follows a determined sequence of operations. These operations will vary depending on the input and output of the machine, as different conditions are met then the state of the automaton will vary. A common kind of automaton is a Finite State Machine (FSM), see figure 2.5.

The appropriate definition of an automaton would be a 5-tuple (Q,Σ,δ,q_0,F), each represents:

- Q is the collection of states q_i
- Σ is the a finite set of the automaton

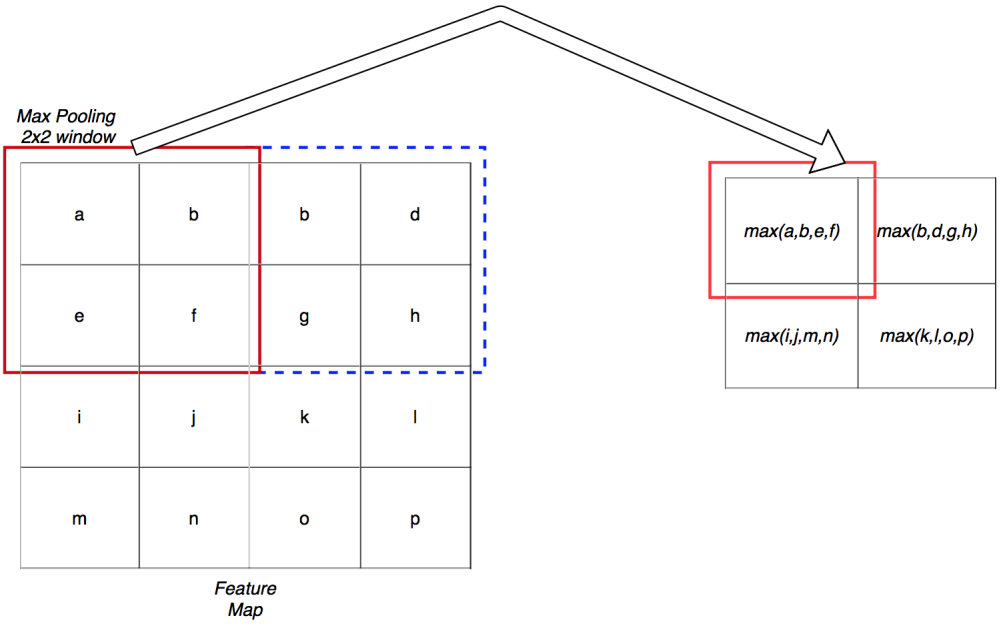


Figure 2.4: Example of Max Pooling

- δ is a transition function to the posteriori state
- q_0 is the initial state
- F is a subset of Q acceptable states

2.7 w-automata

A w-automata is an automaton that accepts infinite sequences as it's input. Since it accepts infinite sequences they have a different acceptance instead of finite automaton. As w-automata it is possible to identify two different automata with two different acceptances:

Buchi Automaton: this kind of automaton accepts all the runs in which there is a final state which is visited infinitely often.

Street Automaton: the acceptance condition varies from the Buchi automaton as it accepts the runs ρ such that for all the pairs (A_i,B_i) of sets of states in a set Ω,

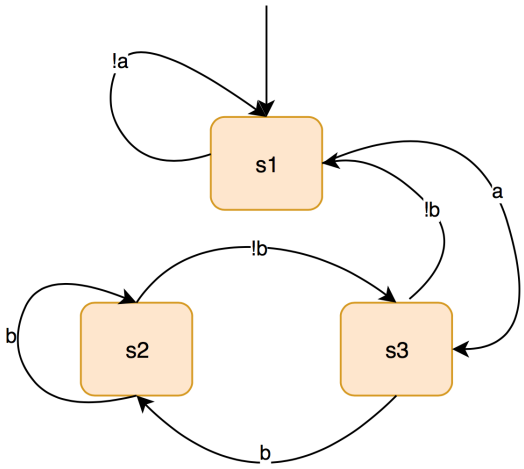


Figure 2.5: Example of Finite State Machine

$Ai \cap Inf(\rho) \neq 0$ or $Bi \cap Inf(\rho) \neq 0$. This implies that if one of the element within the pair is a recurring state then so should be the other element.

2.8 Monitor

"A monitor is a function that recursively verifies the behavior and correctness of a system and raises an alarm when it calculates a violation of the safety property."

Testing and verification are no longer sufficient to guarantee the correctness of a CPS as they have become extremely complicated models and because the faults in the system occur rarely, hence it might be possible that these faults are never tested. Hence, monitoring is a good alternative as it checks the behavior of the CPS in real time. It does this by observing the input and outputs of the system and from this it establishes if the system is operating regularly. The correct use of monitors would be to intervene with a shutdown of the system or trying to fix the faulty behavior before it occurs.

Formally, a monitor is defined as:

$$M : \Sigma^* \rightarrow \{0,1\} \tag{2.3}$$

Where M is the monitor and Σ^* is the alphabet of possible finite output sequences. Then for any $\alpha \in \Sigma^*$, if $M(\alpha) = 0$ then $M(\alpha\beta) = 0$ for all $\beta \in \Sigma^*$. Note that $M(\alpha) = 0$ implies that the sequence has been rejected, contrary, if $M(\alpha) = 1$ then the sequence has been accepted. This means that for any sequence α that leads to a fault detection, then its concatenation with another sequence β leads to a fault detection too. Analogously, for an infinite sequence $\sigma \in \Sigma^w$, if there exists a prefix γ of σ that is rejected by M then M rejects σ. If instead γ is not rejected it implies that M does not reject σ.

Recall the definition of safety property as the assertion that nothing bad will every happen. Now, consider the set of infinite sequences $L(M)$ that is accepted by the monitor M, it is possible to see how $L(M)$ is a safety property.

A parameter that should always be kept in mind when working with monitors is the time it takes to detect the failure of the system. The delay is given by the moment in which the failure occurs to when the monitor is able to detect the failure [20].

The last comment that should be made regarding monitors are the two different techniques for monitoring:

- *External Monitoring*: The correct behavior of the system is measured by the sequences of outputs only. This is relatively easy as there is no need for state estimation but just a check that the measure output is equivalent to the expected output.

- *Internal Monitoring*: The correct behavior of the system is measured by the states sequences. This is a more cumbersome approach as the states are estimated by the observations made.

2.8.1 Accuracy Measures

The implementation of the monitor does not guarantee that all the failures of the components will be detected, therefore to increase its reliability it is necessary to introduce two new parameters; the *Acceptance Accuracy (AA)* and the *Rejection Accuracy (RA)*.

Definition 2. The acceptance accuracy of the model H with automaton A is the probability that a sequence accepted by A will also be accepted by the monitor M.

In other words, it measures the accuracy of the monitor in detecting good sequences.

$$AA(H,A,M) = \frac{g_a}{g_a + g_b} \tag{2.4}$$

Where g_a represents the number of good runs of the systems that are accepted and g_r the number of good runs of the systems that are being rejected. Of course $(1 - AA)$ is the probability of having false alarms.

Definition 3. The rejection accuracy measures the probability that a bad sequence of the system rejected by A is also rejected by the monitor.

$$RA(H,A,M) = \frac{b_r}{b_a + b_r} \tag{2.5}$$

Where b_r represents the number of bad runs of the systems that are rejected and r_a the number of bad runs of the systems that are being accepted. $(1 - RA)$ measures the probability of having missed alarms.

2.8.2 Monitroability and Strong Monitorability

Definition 4. When a monitor has $AA = 1$ and $RA = 1$ it is said to be *strongly monitorable*. This implies that the monitor will always be able to fully identify if the system is in a good or bad mode. This can occur if the good sequences' outputs do not share the bad sequences' outputs. The relationship between the progression of the mode transitions and the outputs is unique so it is possible to perfectly determine when the system is behaving correctly or not.

Definition 5. When the system does not meet the conditions presented before then the system is defined as *monitorable*:

$$AA(H,A,M) \geq \rho \quad \text{and} \quad RA(H,A,M) \geq \rho \tag{2.6}$$

Where ρ is in the range (0,1).

2.9 State Estimation

An essential component for the monitors are state estimators, whose job is to evaluate, by developing a belief propagation, in which state the system currently is. This is due to the stochastic transition between states, hence, a probability transition is needed. The state estimator used for this research is the particle filter, which is a recursive Bayesian filter which estimates the inner state of the system [21].

2.9.1 Bayes' Filter

Before describing the Bayes' filter it is necessary to recall the Bayes' theorem:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{2.7}$$

Thanks to equation 2.7 it is possible to perform the Bayesian inference, which is a technique used to update the probability distribution for an event to occur while new information is provided. Therefore, the probability to be in a certain state will change according to the new observations made. $P(x)$ would be the prior probability distribution, which would be an estimate on the event x to happen before the observation y is made. $P(x|y)$ is the posterior probability distribution, which would the update on the prior probability distribution once it has received an observation. $P(y|x)$ indicates how the observation y is related to x.

In robotics the Bayes' filter works in a similar way and it follows two steps: prediction and correction.

- Prediction: Given the prior belief $bel(x_{t-1})$, input u_t and model of the system $p(x_t|u_t,x_{t-1})$ then the predicted belief $\overline{bel}(x_t)$ is:

$$\overline{bel}(x_t) = \int_{x_{t-1}} p(x_t|u_t,x_{t-1}) \cdot bel(x_{t-1})dx_{t-1} \tag{2.8}$$

- Correction: Once the observation has been made y_t and with the sensor model $p(y_t|x_t)$ then the final belief is:

$$bel(x_t) = \eta p(y_t|x_t)\overline{bel}(x_t) \tag{2.9}$$

Where η is a correcting factor.

2.9.2 Particle Filter

The particle filter is a recursive algorithm which approximates the inner state of the system, what makes this filter so powerful is that it can be applied to both linear and non-linear systems. The filter has a finite number of particles, each assigned with a weight, with which it calculates a discrete distribution to estimate the posterior probability. The higher the number of particles the higher the reliability of the correct behavior, however, longer the computational time. The particles are distributed randomly across the space, thus different weights will be assigned to each particles, hence, an importance value is assigned to each particle.

For the case studied it is assumed that the system's evolution depends only on the previous states and the current sensor's readings.

$$x_t = f_t(x_{t-1}, y_{t-1}) \tag{2.10}$$

Note, that this being a stochastic model the sensor will be affected by a noise v.

Before describing the steps necessary to recursively evaluated the posterior probability density function $p(x_t|y_{1:t})$ it required to make one more assumption: the initial probability distribution must be available. The stages of a particle filter can be summarized with the following steps:

1. Prediction: In this stage an approximation of the belief of the state $x_t^{[m]}$, where m is the particle's index, is calculated from equation 2.8;

2. Weighing the particles: A weight is given to every particle by evaluating the particle's estimation with the real reading. This step helps to differ the most important particles from the least important ones. Eventually the weights are normalized;

3. Re-sampling: New particles are introduced which replace already existing particles, favoring the particles with biggest weights. Then all the particles are assigned the same weight again to restart from step 1.

A diagram presenting the monitoring process is present in figure 2.6.

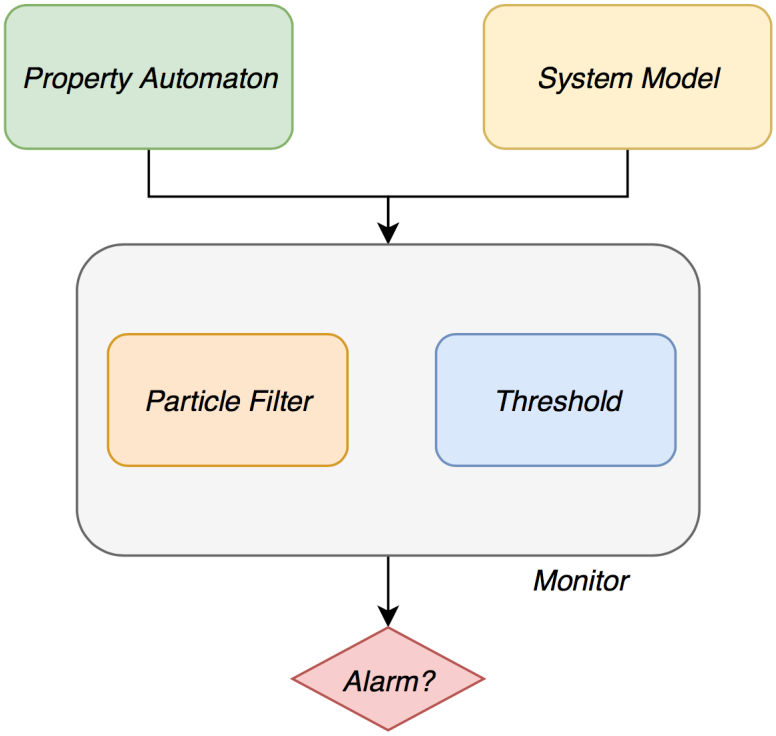


Figure 2.6: Full diagram of the monitoring process

Chapter 3

Tools and Development of the Autonomous Car

3.1 Simulator

Before proceeding with the development with the black box representing the systems model it is necessary to build the system model. This could have been achieved either from a hardware point of view or a software point of view, however, being this the first time this particular approach is being used it was thought that working from a software environment would be better. Hence, it was chosen to perform the research on the *Udacity Self Driving Car Simulator*.

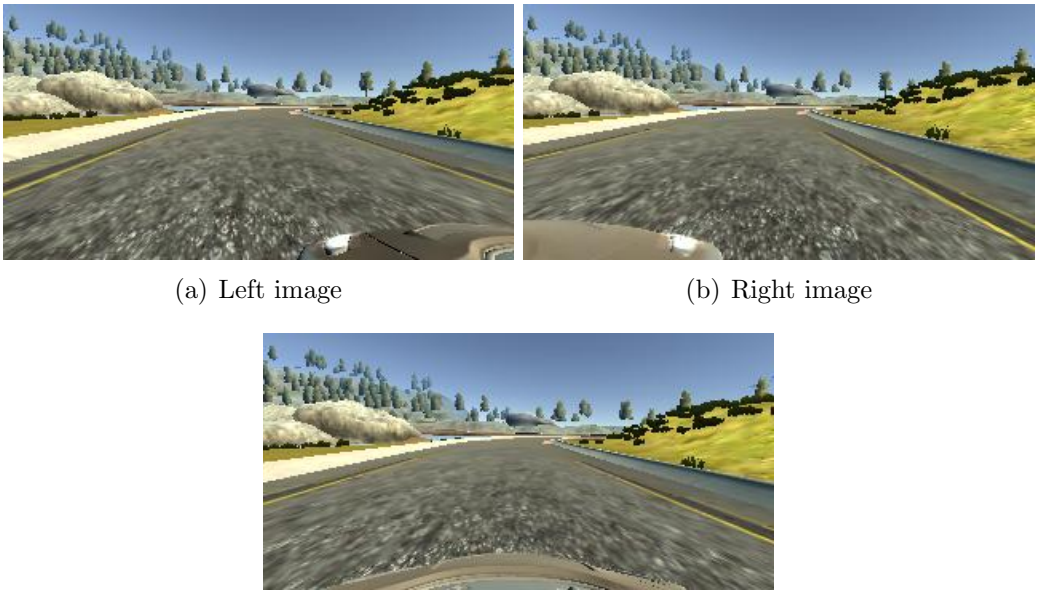
The simulator is an open source tool developed with Unity, a platform to create video games, which has already been previously used for different research [22] and [23]. It is quite an intuitive platform, divided into two main use cases. The first one is called *Training Mode* and as the name suggests it is used to collect data to eventually train the control of the vehicle. In this option the user will drive the car around a track, there are two available, and the simulator will automatically save the data in a *.csv* file which will later be used to train the model. The second use of the platform is the *Autonomous Mode* which runs the trained model on the simulator and the car behaves completely autonomously, from this option it is possible to verify if the model developed is an effective one or not.

The data collected during the training mode includes the following:

- Left camera image;
- Right camera image;
- Center Camera image;

- Current throttle;
- Current speed;
- Current steering angle

The three cameras will record together at the same time instant, as shown in the picture below:



(a) Left image (b) Right image

(c) Center Image

Figure 3.1: Images from the three cameras.

As previously stated there are two tracks provided by the platform, if a network was trained with data collected from one track it would score poorly in the other one. Of course, to fix this problem a network providing both set of images would allow the car to work good on both tracks. For this research's sakes only one track was considered.

The last comment to make regarding the simulator is that it acts as server, it provides images for the trained control network which on the other hand will act as a client.

3.2 Training Tools

The development of the control's neural network has been implemented with Python 3.0 as it is possible to find many useful frameworks for the development of neural networks and image analysis. The core tools used for this research are Keras, Tensorflow and OpenCV. The latter is used on image processing whereas the first two are frameworks used to develop neural networks.

- **Tensorflow**: This is an open source library developed from the Google Brain team used for machine learning implementation. With this library it is possible to develop very flexible neural networks that can be trained on either CPUs ore GPUs;
- **Keras**: Keras is an open source high level API that works on top of Tensorlfow. Keras is a more direct way to exploit Tensorflows' functionalities as most of Tensorflow's functions are already set by Keras itself. This allows for an easier but always effective method to train neural networks.
- **OpenCV**: Another open source library for elaborating images. It provides functions for computer vision applications for machine learning. This tool is very useful to elaborate the images to pass to the neural network and furthermore to extract important features from the images.

3.3 Data Processing

The training data set was collected by driving around the track for three laps and it compromised of 8037 time samples, however, due to the fact that there are three cameras it was possible to triple the number of samples. Thus there was a total of 24111 samples. Furthermore, the data was split between training and validation sets each compromising 80% and 20% of the collected data respectively. The test data set compromised of 1024 images, for the testing of the network only the center camera was used.

Due to the fact that the simulator offers always good conditions while driving: no bad weather, cameras have always a clear vision of the road, no obstacles etc, it is good to add some noise to the images in order to make it more realistic. Furthermore,

these variations will also help to add additional images to the training data set, hence a data augmentation is performed. Below the list of augmentations:

- **Brightness Augmentation.** In real driving conditions the brightness of the environment varies, sunny days, cloudy days, etc. This is achieved by converting the RGB image to the HSV (Hue, Saturation, Brightness) and working on the brightness channel only. With a random uniform function every image was assigned a different brightness noise. The image was then reconverted to the RGB format.

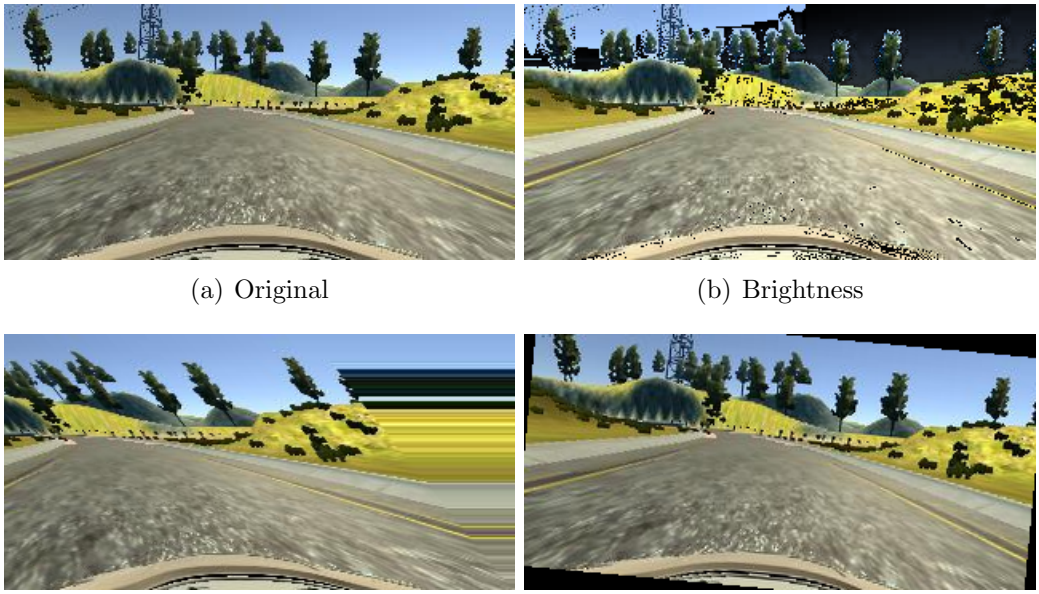
- **Translation.** It is possible that when collecting the data the car was always at the center of the road, hence, to simulate conditions when the car is in different positions on the road the camera was translated, by translating the camera it was necessary to fix the steering angle too.

- **Rotation Augmentation.** This helps to avoid overfitting as it makes the trained network less dependable from the fixed camera rotation. Of course, as the camera orientation varies so does the steering angle.

- **Shear Augmentation.** The random shear occurs with a very low probability and it yields an image which is contorted. This is used to replicate the case in which there is a broken lens on one of the cameras.

3.4 Structure of the Network

In this section it is explained how the control for the self driving car is obtained, this allows the car to be fully independent from the user and will be able to perform decisions by itself. All modern literature regarding the control of autonomous vehicle reference the NVIDIA paper [24], which provides a good convolutional neural network model to control the steering command. The neural network proposed by them has been used with adaptations to the research, as the environment and the number of samples was different. Before describing the model used several new concepts are introduced in order to understand better how the neural network behaves.



(a) Original (b) Brightness

(c) Shear (d) Rotatation

Figure 3.2: Augmented Images

3.4.1 Loss Function

For this particular problem the neural network will be used for regression purposes. What the neural network will try to implement is to estimate a relationship between the input (the images) and the steering angle and see how the latter behaves as new images are fed to the NN. For regression problems it is important to evaluate the error between the estimated output and the expected output to evaluate how close the output is to the actual value. This value will be of vital importance to identify the reliability of the model. The error is measured with the *Root Mean Squared Error* seen in equation 3.1.

$$RMSE = \sqrt{\frac{1}{n}\sum (y_i - \widehat{y}_i)^2} \tag{3.1}$$

3.4.2 Optimization

When building a neural network it is very important to consider what kind of optimization is being used. The optimizers are essential because they define the way in

which the weights are updated, hence, identifying a proper optimization technique will improve the NN's performances. The optimizer that yielded the best RMSE results was the *Adam* optimization. This is an extension of the stochastic gradient descent algorithm in which the learning rate η is fixed throughout the entire training, see equation 3.2 , where J is the loss function and w is the weight of the NN. The adam optimization has instead a varying learning rate which varies according to the mean of the gradients and their variance [25].

$$w = w - \eta \nabla J(w) \tag{3.2}$$

3.4.3 CNN Architecture

As previously stated the architecture was drawn from the NVIDIA paper, however, adaptations were made or the purpose of this research. The final architecture can be seen in figure 3.3.

Description of the Architecture

- *Normalization*: The purpose of this layer is to normalize all the pixel values in the range (0,1). By doing so it is possible to say that all the different input are on a comparable range. This will give equal importance to all the input values of the NN and will lead to a better training;
- *2D-Convolutions*: As explained in Chapter 2 the CNN extracts important features from images, in this neural network we are trying to obtain as many features as possible, taking into account that too many features will result in a bad training, hence, multiple CNN architectures with different feature extractions have been tried until the optimal has been identified;
- *Max Pooling*: Due to the high number of features that are being extracted from the images we want to consider only the most important ones. With the max pooling with a size 2×2 it is possible to speed up the training and to consider only the most important information stored.

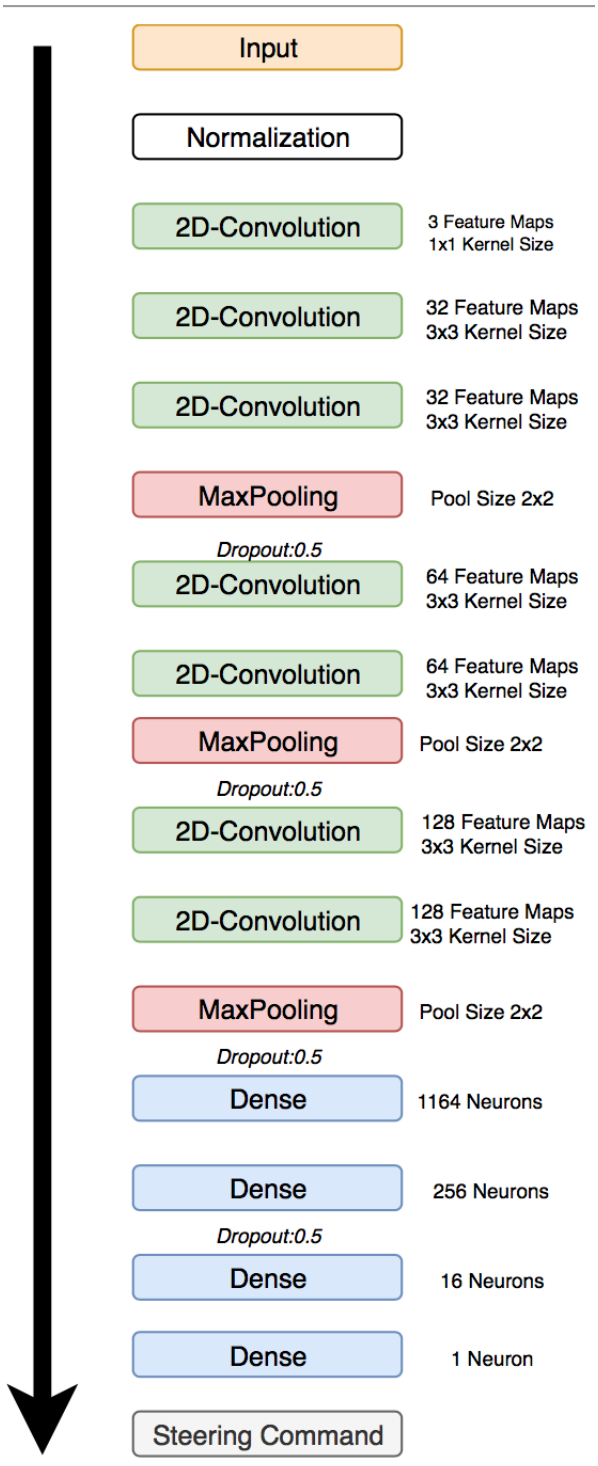


Figure 3.3: CNN Architecture

- *Dropout*: Overfitting is a big problem in neural networks, this occurs when the NN learns the data too well and with some small presence of noise it will negatively impact the operation of the NN. Hence, the dropout method has been implemented. The dropout method sets random activations functions to zero so that some of the weights are not updated. This is a common technique used to avoid overfitting. Overfitting is noticeable when the validation data set's RMSE value is extremely high, this implies that the network is too dependent from the training data set.

- *Dense*: The dense layers are just regular layers of neurons. Every neuron in the dense layer receives the output from the previous layer (densely connected). The dense layers are used to narrow down the features and to have just one output from the NN which would represent the steering angle.

Note that NN have activation functions to render the problem non-linear. In this architecture a good activation function for the CNN is the *Rectified Linear Unit* (ReLU), which output is equation 3.3 . If the input is less than zero then the output is 0 otherwise it assumes the value of the input.

$$f(x) = max(0,x) \tag{3.3}$$

ReLU are effective because they are able to train a lot faster than other activation functions without affecting the accuracy of the output.

The NN was trained with 20032 samples and with a batch of 64 samples, below a graph representing it's loss during each epoch is presented. As soon as RMSE started to increase the training was interrupted and the weights of the best performance epoch were saved as seen in figure 3.4.
The final RMSE value was 0.1263, with an RMSE on the validation set of 0.7115.

3.4.4 Improved Architecture

Not being satisfied with the neural network developed new techniques to improve it were analyzed. From [26], the NN that performs the best result is a LSTM network stacked after a CNN. Considering that RNN are a good method for time series it makes perfect sense to use the LSTM, by knowing the previous steering angle it

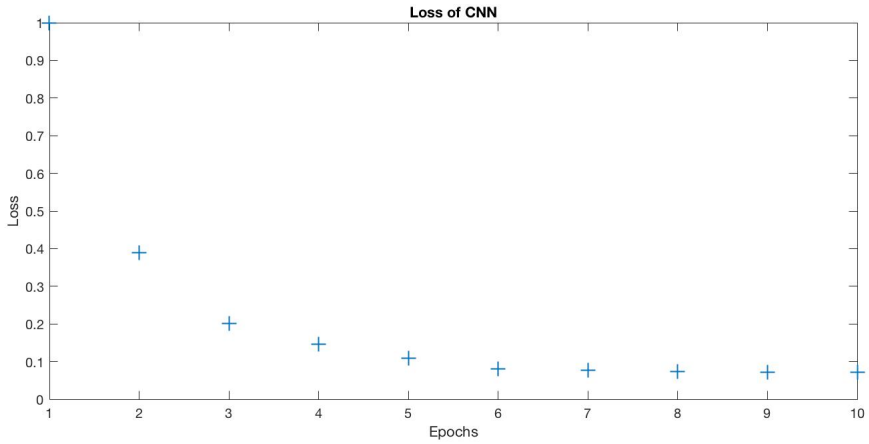


Figure 3.4: CNN loss

is plausible to make conclusions on what the next time step steering angle control should be. Hence the new architecture: figure 3.5.

A difficult task with LSTMs is to define the appropriate number of time steps. LSTMs are composed of memory cells which remember values over a time interval which is arbitrary chosen. In a text recognition the time step chosen could be the number of words in a sentence, the cell's memory will be cleared once the sentence is over. In the problem which is being addressed in this thesis, the appropriate time steps are difficult to choose.

Several time-steps were studied, recall that longer the time step the longer it will take to train the NN and the more more information will be stored. This could be a problem because the memory is limited and in a runtime operation it could cause complications. From table 3.1 no big difference can be noticed from the different time steps, however, it is clear that just with 5 time steps the model already performs better than the one presented before, however, having trained all the different NN it has been decided to use the network with 20 time steps.

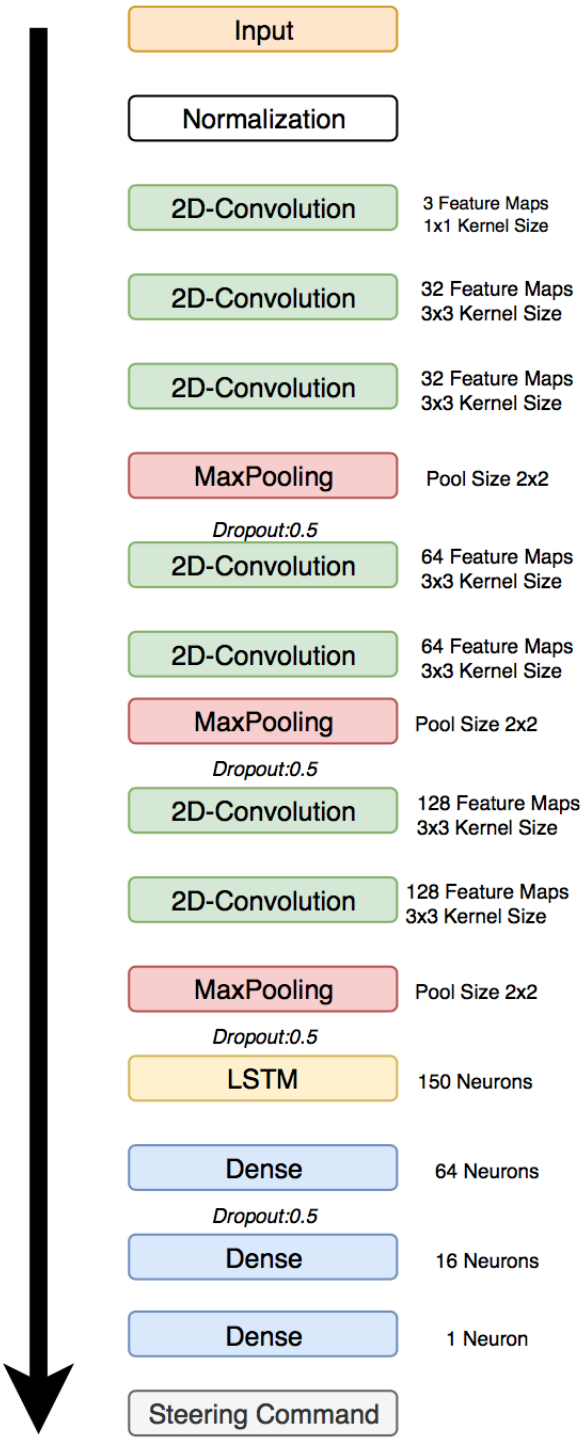


Figure 3.5: CNN+LSTM Architecture

Table 3.1: Different Time Steps Performances

Time Steps	Loss Training	Loss Validation
5	0.0489	0.0932
10	0.0446	0.0927
15	0.0413	0.0929
20	0.0405	0.0912

3.5 Controlling the Throttle

Until now it has only been considered to control the steering angle whereas the throttle has been controlled heuristically by a very simple control. The throttle changed during the operation according to the steering angle, the bigger the steering angle, the lower the throttle so that the car does not go off road. The heuristic control works as stated by figure 3.6 where $s_0 > s_1 > s_2$. Due to coherency the throttle was later implemented by a NN.

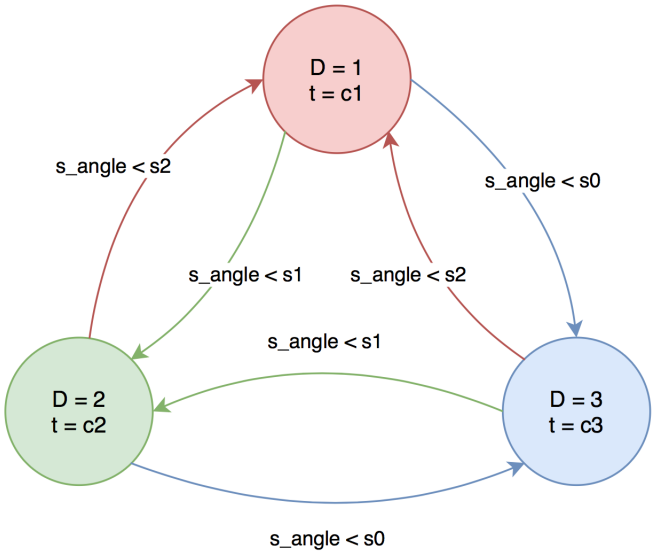


Figure 3.6: Heuristic control of the throttle

Initially, the NN with both CNN and LSTM was modified in order to have both the steering command and the throttle command as outputs, resulting in very poor performances. To fix this problem it was decided to train the neural network for the

throttle command according to the heuristic control, making the throttle dependent from the steering command and not the images directly. The final NN for the throttle was solved as a simple dense layer and again being this a regression problem the *RMSE* was calculated for verifying how close the output is to the expected output. The final control of the autonomous car is depicted in the image below:

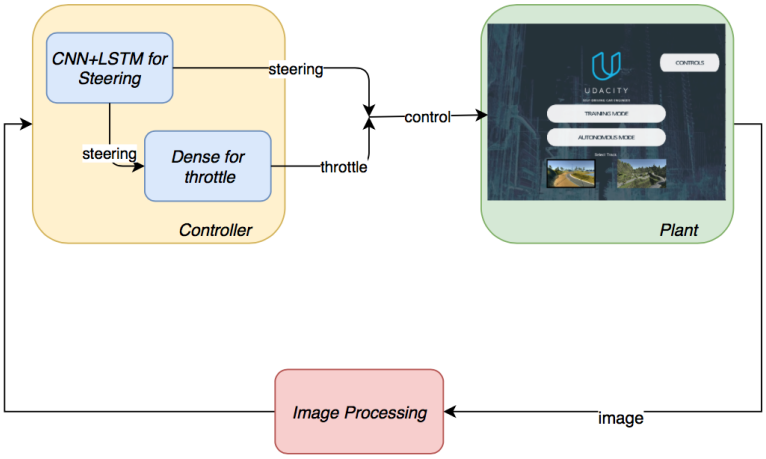


Figure 3.7: Complete control of the autonomous vehicle

Chapter 4

The Monitor

The main monitor's components are the system's model, the safety properties and the particle filter. For this first stage it was decided to verify only the correct behavior of the monitor, hence, the simulator along the control algorithm developed in the previous chapter has been considered as the model of the plant. In this chapter the monitor will be described and how it verifies the correctness of the system states by the use of a particle filter. To proceed the states of the model must be specified along with the safety properties, eventually the monitor's algorithm will be defined.

4.1 Model's states and controls

The model is described by the controls and state variables which are listed below:

1. Acceleration control, a;
2. Steering angle control, s;
3. Measured speed state variable $\hat{v}$

In order to verify the correct behavior for the speed it is necessary to define the model with some dynamical equation using the two controls that have been previously evaluated. Note that both controls will be affected by noise as we want a stochastic behavior:

$$\begin{aligned} s_{noise} &= s + n_s \\ a_{noise} &= a + n_a \end{aligned} \tag{4.1}$$

The kinematic model provided by [27] has been used to define the speed:

$$\begin{aligned} \dot{x} &= v\cos(\psi+\beta) \\ \dot{y} &= v\sin(\psi+\beta) \\ \dot{\psi} &= \frac{v\cos\beta(\tan\delta)}{l} \\ \beta &= \arctan(\frac{\tan\delta}{2}) \\ v &= \int u_1 dt \\ \delta &= u_2 \end{aligned} \tag{4.2}$$

Where u_1 and u_2 are the throttle and steering command respectively. Having the speed defined also in the system's model it then makes it possible to verify its proper behavior. Furthermore, from the dynamical model it is possible to define the position of the vehicle which in turn could be used for other correctness properties.

4.2 Property Automata

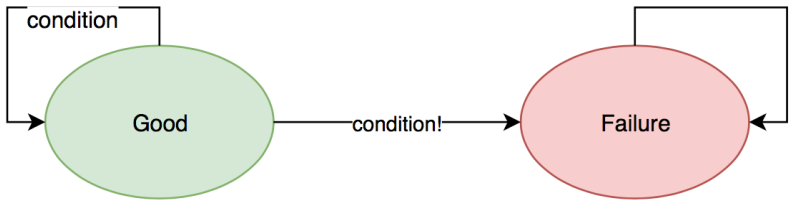
There are two techniques to identify the errors in the operation of the CPS. The first technique is to model the errors with the model of the system. The second instead would be to model the errors separately from the model. The first technique, even if very effective, is not recommended because the monitor would be able to identify only the modeled errors. This could create problems in real life applications because it reduces the potential of the monitor. Hence, the second approach will be investigated as it offers a wider spectrum of operation.

When driving there exist many circumstances that can be considered as dangerous and that should be avoided. Only three of these situations have been monitored, even though many more could be tested:

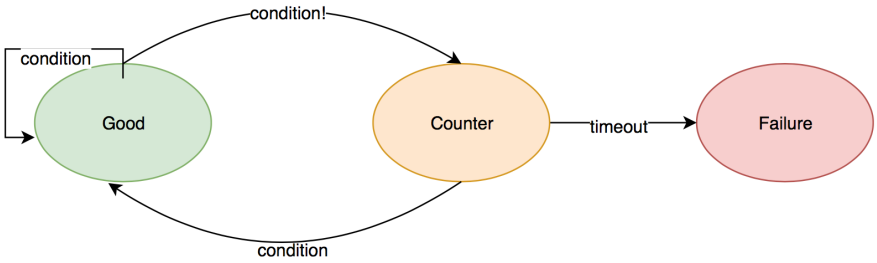
- The speed of the vehicle should never exceed the road limit;
- The centrifugal force should not go over the safety limit. This occurs when the vehicle turns with a high speed during a curve and the centrifugal force pushes the vehicle away from the axis of rotation;

- Misalignment of the camera. It is possible that while driving one of the vehicle cameras could tilt, due to collision or other factors, which could lead to wrong readings, thus yielding wrong commands to the vehicle.

There exist different property automata that could be implemented. For the first two a *counter automaton* has been implemented whereas for the misalignment of the camera a normal automaton has been used. The normal automaton figure (4.1.a) works only in two modes: good and failure. If the conditions are not met then the mode changes from good to fail. The counter automaton figure (4.1.b) identifies between good, error which starts a counter, and failure modes. If certain conditions are not met then the mode changes from good to bad, during this mode the system has a limited amount of time to find the problem and fix it, otherwise it moves to the failure mode.



(a) Automaton



(b) Counter Automaton

Figure 4.1: Types of Property Automata

The same automaton will be used for the speed and centrifugal force, two different counters have been adopted so that it is possible to differ which error occurred. And

a normal automaton has been used to verify the camera's misalignment. Thus, two different monitors will be implemented which work exactly the same but they look for different faulty behaviors. It is important to mention that there is no restriction on the number of monitors that can be implemented.

4.2.1 Implementing Properties

To detect the failures of the properties defined certain thresholds have been implemented. They have been accurately chosen so that it was possible to see runtime failures.The speed limit was easy to select, an arbitrary limit was chosen by checking the distribution of the speed training data set. Whereas the centrifugal force was calculated from the training data set. Remember that the centrifugal force is calculated as:

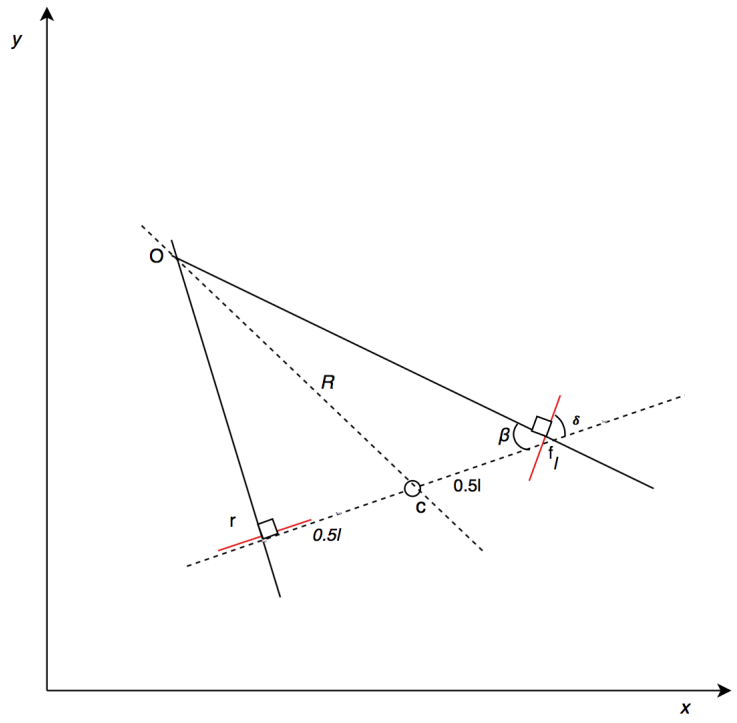


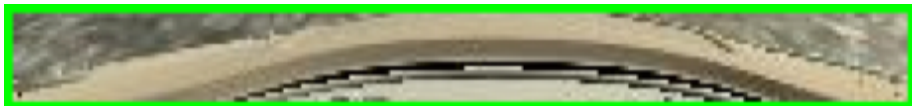
Figure 4.2: Kinematics of vehicle motion

$$F_c = \frac{mv^2}{R} \tag{4.3}$$

Where R is the rotational radius. From the training data set the maximum and the average centrifugal forces have been calculated and from these values an arbitrary value that would allow to detect errors has been chosen. From geometry, see figure (4.2), we obtain the following limit:

$$\frac{v^2}{\sqrt{\tan^2\beta + \frac{1}{4}}} < \frac{F_{c,max}}{m} \tag{4.4}$$

For the misalignment of the camera the problem was less straightforward as it required some image processing. During the real time operation of the vehicle only the center camera was selected to provide the input images of the NN and from these images it was possible to see that there were some fixed points that never changed in all the image frames. This was the front of the car. So the idea behind was to evaluate contours on the image and detect the front of the car only. Once the contour of the front of the car was detected and a central point of the contour was identified, its angle and magnitude were calculated to define the position of the center. By doing this on every image of the central camera it was noticed that the contour was always detected and the central point was roughly the same in all images. The images were first cropped in order to check only the lower bound of the image as it is the only part of the picture to which the algorithm is interested in and then the contour was found. See figure 4.3.



(a) Aligned Camera

(b) Misaligned Camera

Figure 4.3: Contours of Camera Images

By doing this it was necessary to add two additional states to the model of the system: the magnitude of the contour and the angle of it:

$$\begin{aligned} \dot{\theta} &= 0 \\ \dot{m} &= 0 \end{aligned} \tag{4.5}$$

Being the camera fixed, a random probability was assigned to the camera to rotate. This means that during runtime there is a probability threshold α that if it is lower than the random uniform probability then the image is rotated signaling a misalignment of the camera, and the monitor should identify this error.

4.3 Implementation of the Monitor

Recall that the important tool that allows the monitor to perform a good analysis of the data is the particle filter. In Chapter 2 the particle filter operation was divided into three main stages: prediction, weigh evaluation and re-sampling the particles. Hence, what is important is to define well is the number of particles that should be evaluated for the prediction of the system's states. Bare in mind that a higher number of particles will result in a better prediction, nonetheless it will increase the computational time that could delay the monitor's detection time because for every particle there is going to exist a measurement of the array x state.

$$x = \{x_{0,t}, x_{1,t}, x_{2,t}, ..., x_{N,t}\} \tag{4.6}$$

The operation of the monitor can bi divided into four steps: initialization, estimation of the new states, propagation of the states and the final decision.

4.3.1 Initialization

This process is used to start the simulator and the python script, which in turn are used to start the multiprocessing of the scripts so that the monitor can obtain data in real time. Moreover, the initialization is also used to allocate particles uniformly, with each particle containing the system's states, at the time instant $t = 0$. All the counters are set to 0 too and the property automata is set in the *good* state. Once

all parameters are initialized the simulator is allowed to run and the monitor will start collecting the necessary data to perform its operation.

4.3.2 Estimation of the new states

Once the particles have been initialized they are propagated in time to develop predictions of the system states. These values are then compared with the expected values of the system states and with their comparison it is possible to assign weights to the particles. Bigger the difference smaller the weights. Then the particles are normalized and resampled. The particles with bigger weights have a higher chance of being resampled than particles with a smaller weight. With the normalized particles it is possible to determine in which mode of the property automaton the model currently is.

4.3.3 Decision

It was chosen to use a *Threshold Monitor* for the purpose of this research. This means that if the normalized value of the particles being in a failure mode of the property automata is bigger than a certain threshold z then the monitor publishes an error. Note, that if the threshold increases then so does the acceptance accuracy. This is because it becomes harder to reject a run.
In figure 4.4 it is possible to see the full diagram of the monitoring algorithm

4.3.4 Results

The monitor establishes the probability that the safety automaton reaches a bad mode and raises an alarm when such probability is bigger than z. To verify the performance of the monitor it was chosen to use $N = 400$ particles. To determine the effectiveness of the monitor its acceptance and rejection accuracies have been computed according to equations 2.4 and 2.5.

Both accuracies have been evaluated for both monitors, to verify both the AA and the RA values. Furthermore, different threshold values have been implemented to monitor the system so that the monitor that best resembles a strongly monitorable system can be identified.

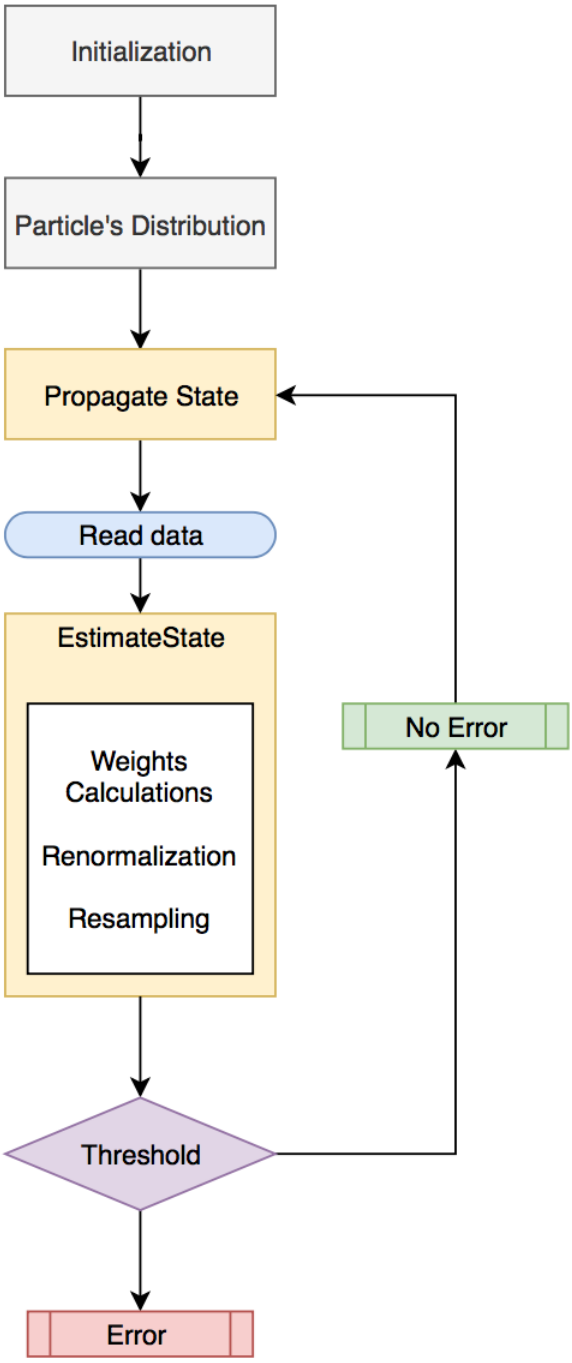


Figure 4.4: Contours of Camera Images

Table 4.1: Results of the two monitors for different values of z

(a) Monitor 1

z	AA	RA
0.8	0.9017	0.9916
0.6	0.9263	0.9692
0.4	0.9498	0.9347
0.2	0.9701	0.8965

(b) Monitor 2

z	AA	RA
0.8	0.8853	0.9810
0.6	0.9008	0.9448
0.4	0.9468	0.9372
0.2	0.9833	0.8961

In table 4.1 the obtained results of the monitors are depicted. As expected as the threshold z increases then so does the acceptance accuracy because the probability has such a high value that it becomes hard for the monitor to reject certain runs. It is also noticeable how the rejection accuracy decreases as the threshold increases, as it is difficult to enter in the fail state when the probability is too high. Of course if z_2 raises an alarm and $z_1 < z_2$, then z_1 would raise the alarm too, the same cannot be said for the opposite.

The table for monitor 1 expresses the AA and RA for the monitor of the first two properties: respecting the speed limit and the centrifugal force. The table for monitor 2 instead expresses the accuracies for monitor of the camera misalignment.

Chapter 5

Data Driven Model

5.1 Introduction

The monitor designed in the previous chapter demonstrated that it had good performances, having acceptance and rejection accuracies really high, when the model of the system was the defined from the simulator. Nevertheless, the core idea behind the research is to verify runtime monitoring of data driven models which will be discussed in this chapter. Often, the system's model is not available as components belong to third parties, or it may be extremely hard to obtain a reliable model. When either of the problems presents itself machine learning is adopted to develop a model of the system. By doing this a black box model will be developed to which runtime monitoring can be applied to. As it was previously suggested, there are two ways of adopting machine learning to runtime monitoring:

- **Learning the Direct Classifier**: Using this approach implies using runtime monitoring as a classification problem. The training data would consist of the system's inner states used as inputs and the outputs would be the modes of the property automaton. In this case the model would identify solely the correctness property for which it was trained. Making this approach limited and computationally consuming as it would imply a model for every safety property.

- **Learning the Plant Model**: This alternate approach means learning the model of the cyber physical system. By learning the plant model a black box will be obtained which will replicate the behavior of the real model. In this case the monitor offers a wider spectrum of applications as it can detect one or more correctness property with less computational expense. As for the same model different property automata can be applied to it, figure (5.1).

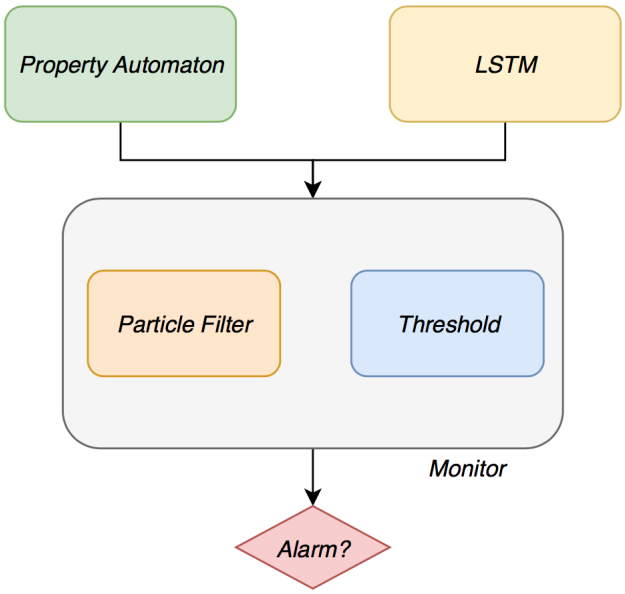


Figure 5.1: Monitor with LSTM model

For the purpose of this research only the second approach will be considered as it is deemed more interesting and complete to study due to the fact that it offers a wider application on the field.

5.2 The Model

To define the model it is necessary to obtain a proper training set, so it is needed to consider which values are needed for the formulation of the problem. Considering that the particle filter the reliability of the predicted states is compared with the expected ones, it is necessary to develop prediction with the LSTM of the system's states. Hence, the following:

$$p(x_{t+1}|x_t,u_t) \tag{5.1}$$

It is wanted to predict the states at the next time step x_{t+1} having access to the control u_t and the states at the current time step x_t.

Recall that the property automata defined previously are verifying that the speed does not exceed the speed limit, the centrifugal force should be lower than a certain

boundary and the camera's position. Hence, the states to verify are:

$$x_t = \begin{bmatrix} v_t \\ \theta_t \\ c_t \end{bmatrix} \tag{5.2}$$

Where v is the speed, θ is the angle of the center of the contour and c is the magnitude of the center of the contour. The control input u_t instead is:

$$u_t = \begin{bmatrix} \delta_t \\ a_t \end{bmatrix} \tag{5.3}$$

Where δ and a are the steering command and throttle command respectively.
The data was collected by running the simulator and saving the values of the states and control inputs on a *.csv* file for a total of 100,000 samples.

Many neural networks architectures have been implemented to model the system, each presenting some problems due to a variety of reasons:

- *Training Data*: It is necessary to obtain a lot of data to train this kind of neural network, additionally the vehicle travels mostly at constant speed and the track has more straight segments than curvature segments so the neural network is biased towards a straight track. This lead to the NN to not consider the data collected when the vehicle was in a curve;

- *Overfitting*: As explained previously it occurs when the validation score is way worst than the training score, meaning that the NN was too dependent from the training data set;

- *Time Steps*: Different time steps lead to different results. To decide how much information should be stored is not a straightforward application, as there is no direct way of finding the optimal one.

To find the optimal architecture all these parameters have been tuned, as well as the number of layers. The problems listed before have been addressed in the following ways:
Training Data

When testing the trained NN the performances were quite low, the outputs were not resembling the testing data outputs. This is because the data collected mainly emphasizes on when the car is going straight as there are more straight points than curves in the simulator. So what it has been tried is to collect more data in curves. By doing so the NN will have more information and will not consider the curves as meaningless data. A second approach that has been implemented was by using dynamical equations of the system, see equations (4.2), to generate data which reflected the operation of the vehicle in a curve.

Overfitting

To overcome this problem multiple NNs architectures have been implemented and the one resulting with best results in the validation and testing data set have been chosen. Too complex architectures will result in overfitting as they are susceptible to noise. A lot of data has been developed too so that the NN has a variety of combinations. Moreover, the dropout method has been applied in this case too.

Time Steps

Arbitrary values have been chosen for the time-steps, what has been tried is using 5,10,15 and 20 time steps.

For the architecture of this NN it has been noticed that having multiple layers of LSTM provided better results as it yields more accurate results and a better description of the system. For challenging sequencing problems stacked LSTMs work better than single layer LSTM networks. Again, to test this neural network, being a regression problem, the $RMSE$ value of the output was calculated in order to measure the accuracy of the network.

The complete architecture of the NN is depicted in figure 5.2, two stacked LSTM have used along three dense layers to narrow down the output.

5.3 Results

The monitoring algorithm didn't change at all, minor changes were performed in order to adapt the LSTM model to work with the monitoring algorithm. In this section the results of the model designed with an LSTM will be analyzed and how the monitor's reliability, acceptance and rejection accuracies, are affected by a black box model.

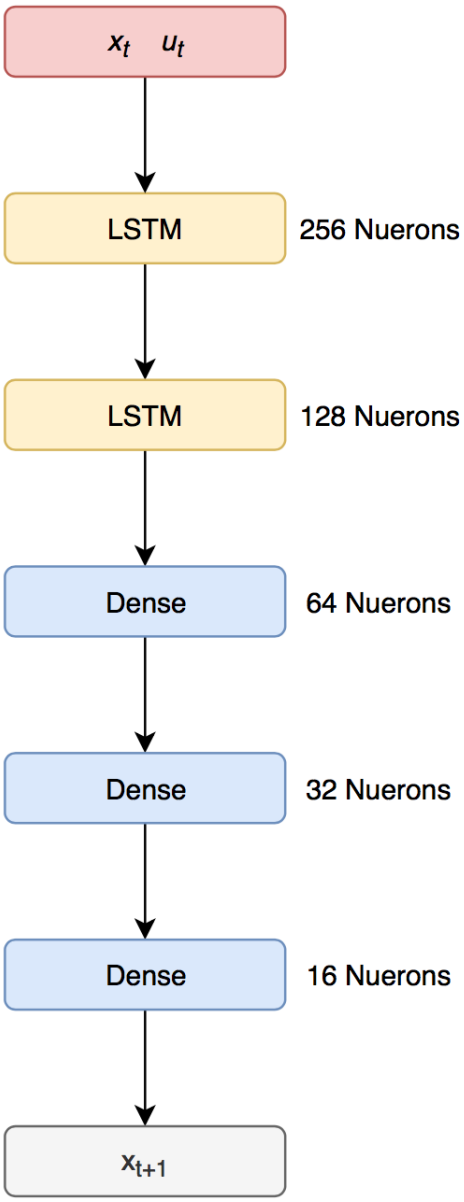


Figure 5.2: Complete architecture of the system's model

5.3.1 Predictions

The trained NN performance was verified by having 10,000 samples in the test data seta. The architecture presented previously, which is the architecture that performed better, was tested for four different time steps. The results are listed in table 5.1,

where for every different time step the RMSE for the training and testing data sets are given.

Table 5.1: LSTM model of the plant performance

Time Steps	**Loss Training**	**Loss Testing**
5	28.1032	32.7785
10	24.4268	27.1071
15	25.008	28.8784
20	27.4569	30.0837

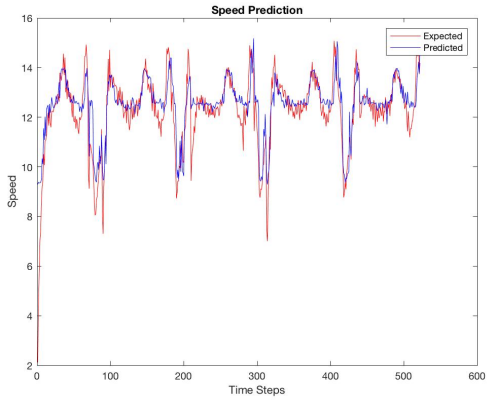
From the obtained results the LSTM network that performed the best is the one with 10 time steps. In figure 5.3 graphs of the predicted states are compared with the expected states. The results are not very encouraging, especially when predicting the states of the angle and magnitude of the centroid of the image contour. Whereas the prediction of the speed is more reliable.

The results make sense as the camera's misalignment does not depend on previous data. The camera's rotation was forced by a random function. Whereas the speed is influenced from previous readings it is not possible to say the same for the camera's position. From the graphs it is clear to see how the predicted speed resembles the expected one, hence it makes sense to verify the accuracy for the monitor that monitors the speed limit and centrifugal force. Due to the bad performance of the LSTM on predicting the centroid's position in the contour it wouldn't be wise to test the second monitor. In order to obtain the proper behavior of the camera misalignment it would be required to measure different states that could be obtained from different sensors.

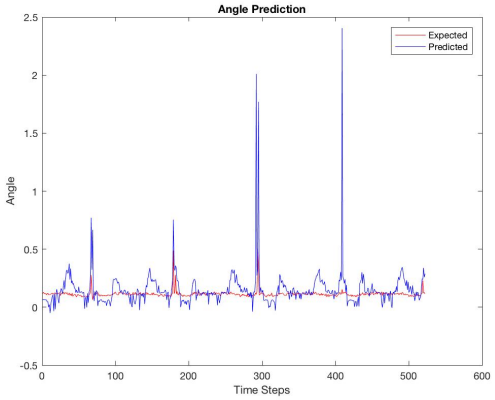
5.3.2 Monitor's Accuracy

The monitor was tested with the data driven model, in this case only the monitor for the speed and centrifugal force properties was tested because the black box did not manage to learn the behavior of the camera's misalignment.

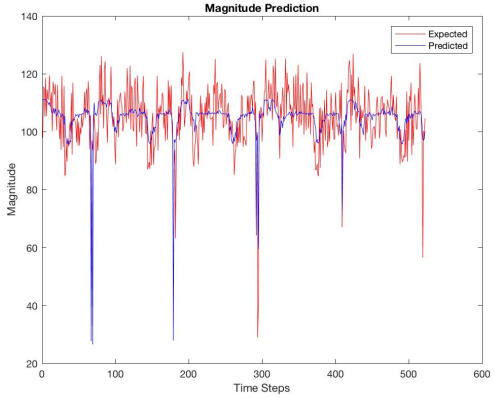
The acceptance accuracy and rejection accuracy have been calculated for the model in table 5.2, and it provides relevant data regarding the use of data driven models to monitor. The data obtained is promising as the AA and RA behave as expected, in the same way as they were defined in Chapter 4.



(a) Speed



(b) Angle



(c) Magnitude

Figure 5.3: Predictions of LSTM vs expected states

Table 5.2: Monitor's Performance for model based design

z	AA	RA
0.8	0.8426	1
0.7	0.8673	1
0.6	0.8708	0.9692
0.5	0.8780	0.9230
0.4	0.8819	0.9259
0.3	0.9435	0.7333
0.2	0.9818	0.6667
0.1	1	0.4444

Note, that the values reported in 5.2 are the measurements respecting the monitor's reliability and not the reliability of the model. The reliability of the model derives from the RMSE calculated from the NN. The results shows that safety monitoring can be used on model based learning algorithms.

Chapter 6

Conclusion

The development of an appropriate model can be a challenging task, previous research has proven that the monitoring of cyber physical systems modeled with HMC can be achieved. The innovative idea of this research is to introduce a different model from the ones that have previously been implemented and to study its effectiveness when monitored during runtime. This has been achieved by using a self driving car simulator and appropriate control techniques that have been implemented on real physical systems. The results collected are promising, being this the first application of this method it is possible to claim that the final remarks are satisfactory.

When the monitor was implemented on the simulator only it behaved really well, scoring very high values for both the AA and RA, it did not make the system strongly monitorable, however, it was very close to an ideal monitor. Eventually, by collecting data from the simulator it was possible to define a model of the system using deep learning algorithms. By using LSTMs it was possible to increment the number of time series to save in the memory cell, other RNNs have a problem with long time series and would loose information on the long run.

Unfortunately not all the states that have been considered for this research could have been modeled, this is because they were independent of time, furthermore, the variables varied randomly without following any specific function. However, the NN model has been able to learn the behavior of the speed, thus, two of the three properties defined could have been monitored. The monitor accuracies were high, demonstrating that using a data driven model for monitoring is a valid method that should be considered for future research.

Being this an innovative approach there are many improvement that can be made in order to obtain better results. In order to fix the problem regarding the misalignment a better approach could be used. This could be achieved by modeling the scene, where the visual information has a relationship with the control inputs

hence with scene could possibly be model with a RNN. Furthermore, this would allow to verify all the correctness properties related to visual operations.

Moreover, using a real physical system instead of the simulator. Simulators, often, do not provide realistic case studies and usually software development has less problems as time delays are not as high as in hardware components. It would be interesting to verify the monitor on the black box model of a real physical system and check if the results are encouraging.

Bibliography

[1] A. Prasad Sistla and Abhigna R. Srinivas. Monitoring temporal properties of stochastic systems. In Francesco Logozzo, Doron A. Peled, and Lenore D. Zuck, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 294–308, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[2] Mahesh Viswanathan and Moonzoo Kim. Foundations for the run-time monitoring of reactive systems – fundamentals of the mac language. In Zhiming Liu and Keijiro Araki, editors, *Theoretical Aspects of Computing - ICTAC 2004*, pages 543–556, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[3] A. Pnueli and A. Zaks. Psl model checking and run-time verification via testers. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, pages 573–586, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[4] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. Runtime verification of safety-progress properties. In Saddek Bensalem and Doron A. Peled, editors, *Runtime Verification*, pages 40–59, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[5] Kalpana Gondi, Yogeshkumar Patel, and A. Prasad Sistla. Monitoring the full range of ω-regular properties of stochastic systems. In Neil D. Jones and Markus Müller-Olm, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 105–119, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[6] Xenofon Koutsoukos, James Kurien, and Feng Zhao. Estimation of distributed hybrid systems using particle filtering methods. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control*, pages 298–313, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[7] V. Verma, G. Gordon, R. Simmons, and S. Thrun. Real-time fault diagnosis [robot fault diagnosis]. *IEEE Robotics Automation Magazine*, 11(2):56–66, June 2004.

[8] Marcelo d'Amorim and Grigore Roşu. Efficient monitoring of ω-languages. In

Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification*, pages 364–378, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[9] Sheila McIlraith, Gautam Biswas, Dan Clancy, and Vineet Gupta. Hybrid systems diagnosis. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, pages 282–295, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[10] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, September 2011.

[11] A. Prasad Sistla, Miloš Žefran, and Yao Feng. Runtime monitoring of stochastic cyber-physical systems with hybrid state. In *Proceedings of the Second International Conference on Runtime Verification*, RV'11, pages 276–293, Berlin, Heidelberg, 2012. Springer-Verlag.

[12] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

[14] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.

[15] Florent Altché and Arnaud de La Fortelle. An LSTM network for highway trajectory prediction. *CoRR*, abs/1801.07962, 2018.

[16] S.P. Meyn and R.L. Tweedie. Markov chains and stochastic stability. *Cambridge University press*, abs/1801.07962, 2009.

[17] L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, Jan 1986.

[18] Sepp Hochreiter and JÃ$\frac{1}{4}$rgen Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in Neural Information Processing Systems 9*, pages 473–479. MIT Press, 1997.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[20] A. Prasad Sistla, Miloý Žefran, Yao Feng, and Yue Ben. Timely monitoring of partially observable stochastic systems. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, HSCC '14, pages 61–70, New York, NY, USA, 2014. ACM.

[21] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, Feb 2002.

[22] S. Raj, S. K. Jha, A. Ramanathan, and L. L. Pullum. Work-in-progress: testing autonomous cyber-physical systems using fuzzing features from convolutional neural networks. In *2017 International Conference on Embedded Software (EMSOFT)*, pages 1–2, Oct 2017.

[23] A. Buyval, A. Gabdullin, R. Mustafin, and I. Shimchik. Realtime vehicle and pedestrian tracking for didi udacity self-driving car challenge. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2064–2069, May 2018.

[24] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[26] Andrew Simpson. Self-driving car steering angle prediction based on image recognition. 2017.

[27] Danwei Wang and Feng Qi. Trajectory planning for a four-wheel-steering vehicle. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 4, pages 3320–3325 vol.4, May 2001.