MODELING AND SIMULATION OF LANE KEEPING SUPPORT SYSTEM USING HYBRID PETRI NETS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Carmela Angeline C. Padilla

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2019

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Lingxi Li, Chair

Department of Electrical & Computer Engineering

Dr. Brian King

Department of Electrical & Computer Engineering

Dr. Yaobin Chen

Department of Electrical & Computer Engineering

Approved by:

Dr. Brian King

Head of Graduate Program

To my family, for their love and support.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Dr. Lingxi Li. The door to Dr. Li's office was always open whenever I had questions or ran into roadblocks with my research. He was also a source for kind words and encouragement. A special thank you goes to Dr. Brian King, the head of the ECE Graduate Program, for his guidance since the beginning of my graduates studies. Additionally, I would like to thank my committee members, Dr. King and Dr.Yaobin Chen for sharing their time and constructive feedback on my thesis. I would like to thank my friends for their passionate participation, inputs and advice. I am also grateful for Sherrie Tucker for her time and helpful reminders. Lastly, I would like to thank all the other faculty and staff of the ECE department for their help and insights throughout the completion of my studies.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Padilla, Carmela Angeline C. M.S.E.C.E., Purdue University, August 2019. Modeling and Simulation of Lane Keeping Support System using Hybrid Petri Nets. Major Professor: Lingxi Li.

In the past decades, the rapid innovation on technology has greatly affected the automotive industry. However, every innovation has always been paired with safety risks that needs to be quickly addressed. This is where Petri nets (PNs) have come into the picture and have been used to model complex systems for different purposes, such as production management, traffic flow estimation and the introduction of new car features collectively known as, Adaptive Driver Assistance Systems (ADAS). Since most of these systems include both discrete and continuous dynamics, the Hybrid Petri net (HPN) model is an essential tool to model these. The objective of this thesis is to develop, analyze and simulate a lane keeping support system using an HPN model. Chapter 1 includes a brief summary of the specific ADAS used, lane departure warning and lane keeping assist systems and then related work on PNs is mentioned. Chapter 2 provides a background on Petri nets. In chapter 3, we develop a discrete PN model first, then we integrate continuous dynamics to extend it to a HPN model that combines the functionalities of the two independent ADAS systems. Several scenarios are introduced to explain the expected model behavior. Chapter 4 presents the analysis and simulation results obtained on the final model. Chapter 5 provides a summary for the work done and discusses future work.

1. INTRODUCTION

The car industry has greatly developed over the past few decades in making safer and more efficient cars. Back then, maneuvering a vehicle was always dependent on a driver's current situation and environment. This decision-making process has led to road accidents caused by miscalculations, fatigue and false predictions of the driver. Due to such experiences, vehicle operation has no longer been solely dependent on a driver but is now considered as a three-part system consisting of the driving environment, driver and vehicle [1].

In recent years, big data from driver-vehicle interactions and past road-related casualties are analyzed and shared to driver and vehicle industry to improve driving efficiency and vehicle features. These features are consistently modified and tested to promote road safety. Collectively, the features are called the advanced driver assistance systems (ADAS). ADAS is one of the fastest-growing area in the automotive industry with increasing rates of adoption in the quality. According to one report, the global ADAS market has accounted for $22.52 billion in 2015 and is expected to reach $89.09 billion by 2022 [2]. Due to its global popularity, research ventures on ADAS have produced efficiencies in reducing driver's load, predicting crash events and reducing fuel consumption. Some of the ADAS features are adaptive cruise control, anti-lock braking system, automatic parking, blind spot monitor, collision avoidance system, parking sensor, tire pressure monitoring, turning assistant, etc. Combining and utilizing these individual technologies are what we have in smart vehicles and road traffic models today.

1.1 Road Safety through Petri Nets

For quite some time, attention has been directed to discrete event systems for modeling ADAS and road traffic. The prevalent tool for the simulation and analysis of these models is the Petri Net (PN). Serving both as a mathematical and a graphical tool, the PN was shown to be suitable for the analysis of complex systems due to the numerous ways it can be used. Since traffic flow systems are driven by both continuous and discrete events, a Hybrid Petri Net (HPN) was used to create the road model. A HPN is a type of petri net that combines both discrete events, such as the change of traffic signals, and continuous events like the motion of vehicles. The authors of [3] have clearly demonstrated the benefits of using this model for traffic optimization. Yaqub has used Timed Hybrid Petri Nets (THPN) to further improve traffic flow. The THPN model not only showed the event evolution but also considered externals events that can only be changed through time. An external event could be interpreted as traffic during rush hour at a certain time in a day where the percentage of vehicles heading west is greater than those heading east. [4]. The authors of [5], utilize the car collision avoidance feature and create a controller using a Fluid Stochastic Petri Net (FSPN). The dynamics of the fluid flow is similar to that of continuous driven events such that rate of flow is a non-integer. Utilizing the FSPN, the work has provided information for predicting driving events based on driving behaviour and environment. Sensors were used to gather speed, acceleration and distance between vehicles to verify the functionality of the FSPN controller for collision-free driving. Finally, the automated parallel parking feature was used in creating an HPN model to control a vehicle parking autonomously. Distance is considered a flow event while sensors and camera detecting the surroundings of the vehicle described the discrete events, making the HPN model perfect for simulation [6]. Nevertheless, sensors have always been used throughout ADAS for data gathering and decision-making.

In the work mentioned above, the authors have stated that the use of Petri Nets, specifically Hybrid Petri Nets for hybrid modeling, provide an apt structure to understand the relationship between driver and vehicle to improve safety. After gaining perspective on this type of modeling approach, two ADAS features are selected and placed in focus to understand the relationship between hardware and software electronics for the shared goal of road safety and driver comfort.

1.2 Lane Departure Warning

The first half of the ADAS feature used in this work is the Lane Departure Warning System (LDWS). As the name implies, the electronic system sends out a notification to the driver if the vehicle departs its lane without intention. This warning can be an audible sound or visible icon on the dashboard. The LDWS involves three subsystems: driver, vehicle and road. The system makes use of the relation of the three to determine the vehicle-road relationship. Firstly, the vehicle-road state is obtained through computer vision. Through analyzing the relationship between vehicle and road, the lane departure detection approach works. In order for the electronic system or controller of the ADAS to determine which lane the vehicle is on, sensors and cameras are needed to take snapshots of the road. Cameras might be additional hardware mounted on the host vehicle or they maybe built-in the vehicle. For example, a single digital camera and a laser range finder has been used in [7] to identify the lane segments and categorize them as the master and slave lane boundaries. Another paper has used a built-in forward-looking camera for distraction-free driving. An advantage of any ADAS feature is that they can be integrated to increase road safety. The authors of [8] have combined LDWS with another ADAS feature called the forward collision warning system. This camera-based driving system was shown to be an enhanced method that can analyze the video captured and warn the driver when the host vehicle is too close to the front vehicle. On the other hand, Cualain,

Glavin and Jones have added depth into an image by using two cameras: forward and rear-facing. They have mounted cameras that are configured to capture the height and angle of the road surface. [9]

The images captured by these cameras are fed into the image processing module of the LDWS. Although this paper has assumed that post-processed data as input to the final model, several lane detection methods and algorithms in previous related works are presented in this section. With any LDWS set-up, image processing is required to detect lane markings on the road and to measure position of vehicles relative to the lanes. The cameras capture the "region of interest" (ROI). The ROI can be extracted through Hough Transform or canny edge detection [10] [11]. Hough transform detects lines, circles and other structures dependent upon the parametric equation given. The edge detection based on canny operation defines the lane boundaries by the sharp contrast between the road surfaces and the painted lines. Authors of [12] have improved an algorithm based on ROI segmentation. Hough Transform is applied after the road lanes are identified and divided into the left and right sub-regions. Without segmentation, processing an image would produce several Hough lines that will create ambiguities in lane departure. After the lanes have been identified and divided, lane departure is determined from the driver-road relationship. An, Wu and He have proposed a new method based on two existing methods, RRS and Time-To-Lane-Crossing (TLC) [13]. In their paper, two standards are used to weigh the lane departure detection methods: Warning Onset Time (WOT) [14] and Nuisance Alarm Rate (NAR). The former measures how much reaction time the driver has to prevent the outside tire from crossing the lane, while the latter measures how many nuisance alarms occur per hour. An ideal LDWS would have low NAR and the WOT would be sufficiently long. The RRS method triggers alarms when the outside of the left or right front tire touches the lane. This method has a high NAR and short WOT. Since some drivers are used to driving along the lane boundary, this method is mainly for awareness. The TLC method is the measure of the time remaining before a vehicle with a given path will depart the road. However, it is still the most

commonly used method even if it is the sensitive to error. The difference with their lane detection method is the addition of capturing the angle between lanes. This can accurately distinguish when the vehicle will depart the lane. From the changing rate of angle between lanes (CRAL) method, the steering angle was taken specifically taken consideration into the making of this paper.

1.3 Lane Keeping Assist

In opposition with the LDWS being a mere warning, the second half of this project proposal is the Lane Keeping Assist System (LKAS). It is an ADAS feature that detects the driver's steering effort and decides to assist with right or left steering to avoid the line boundaries. The general set-up of this system is divided into two parts: the camera for lane detection and the controls system for the steering reaction force. The main purpose of the camera is to detect the white line for the current position of the vehicle. One disadvantage of not using LDWS with LKAS is poor lane detection. In [15], the authors mentioned issues such as dirty lanes or places where the dynamic range of the image varies greatly. Examples of these places are tunnel entrances, exits in bright daylight or any lane marking change. Another example of this type disadvantage can be found in [16]. This LKAS restricts use of either right or left steering to avoid the right or left line boundary. However, during the interventions of this assist-system, the vehicle's longitudinal velocity should remain constant and the road lane is assumed to be straight at all times. Lastly, [17] used lateral offset to be an input in their system to reflect the distance difference of same lane markings in different frames.

In the papers mentioned above, LKAS is meant to assist the driver to keep the vehicle in a single lane only. However, the continuous demand for road safety development has made the authors of [18] to combine the two systems, LKAS and LDWS, to create an interactive lane keeping control system. Their proposed system requires a controller that simultaneously received the turn signal status, vehicle speed, steering

torque, steering wheel angle and road information such as curvature, slope and lateral displacement. Once all data has been processed, the controller sends out a message to the motor driver and the vehicle's electronic power steering to assist the driver. Similarly to how this project has been carried out, the authors have used MATLAB and Simulink for their simulations. Even though the authors did not mention the organization and build of their system controller and power steering connection, their work has significantly contributed to the design of the final model of this project.

1.4 Thesis Contribution

The variety of work in both Petri Nets and ADAS was the general motivation in the development of a hybrid petri net for the LDWS and LKAS. It has been shown that a Hybrid Petri Net (HPN) is the most suitable modeling tool for a system that involves both discrete and continuous events. Most of the papers provided adequate information about image processing and the mechanical background for steering assists. However, they did not provide the theory behind the controller of the two ADAS features presented. This thesis incorporates the control theory and the difference in the inputs from both systems. Some of these inputs include the steering torque and steering angle from the LKAS and the lane width and velocity sensors for the LDWS. It combines both functionalities and applies them sequentially. The assumptions made while creating the system model are mentioned in the next chapter.

1.5 Thesis Organization

This paper began with the introduction for ADAS and followed by a background on Petri nets. The Petri net section discusses the three types: discrete, continuous and hybrid. The third chapter describes the steps in modeling the system. It also contains the flaws and changes done to arrive at the final version of the model. The

fourth chapter mentions the programs used for modeling and simulation. This chapter contains all the model analysis and simulation results done on the final model. The final chapter summarizes the work done and recommendations for future work.

2. BACKGROUND ON PETRI NETS

2.1 Notations and Definitions

In the early 1960's, Carl Adam Petri developed a mathematical model for describing relations existing between conditions and events. Engineers have used this model, now called Petri Nets, for modeling discrete event dynamic systems. It is a graphical tool that manipulates certain events according to rules that are set. This allows representation of general control systems whose operation depend on potentially complex control algorithms [19]. It is composed by mainly two nodes,

- Transition - associated with an event that will occur. It is represented by a vertical or horizontal bar.
- Place - associated with conditions that are required for an event or transition to occur. It is represented by a circle.

A discrete place can contain a discrete number of tokens. This indicates whether the condition/s necessary for the occurrence of an event is available. Some places are taken as pre-conditions for a certain event to occur. These are viewed as the "input" to a transition. While other places are viewed as the "output" of a transition, they are the post-conditions that are affected when a certain event has occurred. The two nodes are connected to each other through arcs. The transitions, places and relationships between them through the arcs makeup the the Petri Net graph. It is a weighted bipartite graph because arcs cannot connect two nodes of the same type, i.e. two places or two transitions [20]. Fig 2.1 illustrates the graphical representation of the components of the graph.

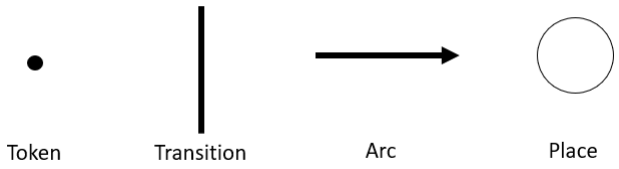

Fig. 2.1.: Petri Net Notations

The precise definition of a Petri net graph or Petri net structure is based on four elements and represented as,

$$N = (P, T, A, \omega) \tag{2.1}$$

where,

P is a finite set of places and is written as,

$$P = \{p_1, p_2, ..., p_n\}, \tag{2.2}$$

T is a finite set of transitions and is written as,

$$T = \{t_1, t_2, ..., t_m\}, \tag{2.3}$$

A is a set of arcs from places to transitions or vice versa. It is mathematically written as,

$$A \subseteq (P \times T) \cup (T \times P), \tag{2.4}$$

and w is the weight function on the arcs. This denotes the weight for each arc connected the two different notes. The weight function is represented as,

$$\omega : A \to \{1, 2, 3, 4...\}. \tag{2.5}$$

The conventional form of an arc is written as either (p_i, t_j) or (t_j, p_i). From this notation, the set of input places to a transition t_j is represented as $I(t_j)$ while the set of output places to a transition t_j is represented as $O(t_j)$. These sets of places are mathematically represented as:

$$I(t_j) = \{p_i \in P : (p_i, t_j) \in A\}, \; and \tag{2.6}$$

$$O(t_j) = \{p_i \in P : (t_j, p_i) \in A\}. \tag{2.7}$$

The same notation is used for the input and output transitions to a place p_i as well. They are mathematically represented as:

$$I(p_i) = \{t_j \in T : (t_j, p_i) \in A\}, \; and \tag{2.8}$$

$$O(p_i) = \{t_j \in P : (p_i, t_j) \in A\}. \tag{2.9}$$

In drawing Petri net graphs, there is a need to differentiate between the two nodes, circle for places and bars for transitions. The arcs connecting the places and transitions represent the elements of the set A. This tells us that an arc directed from place p_i to transition t_j means that p_i is an element of $I(t_j)$. Moreover, if $w(p_i, t_j) = k$, then there are k arcs from p_i to t_j. The default value of an arc weight is assumed to be 1 if not shown explicitly. An arc weight greater than one, three for example, denotes that there are three arcs with a weight of one each between two nodes. Instead of drawing three arcs, this can be represented as a single arc of weight three. Fig. 2.2 is an example of a simple Petri net graph. The notations and definitions presented above will be used to describe the example.

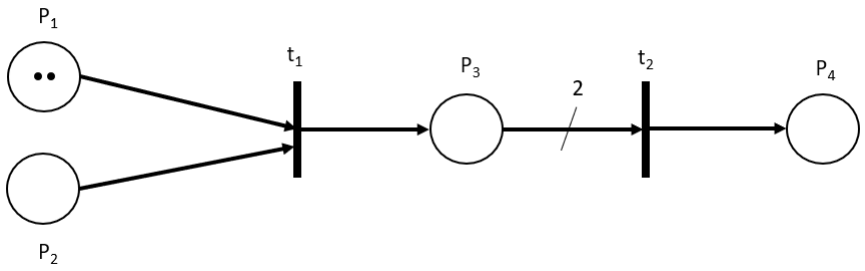

Fig. 2.2.: Simple Petri net graph

The example is defined as follows,

$$
\begin{aligned}
P &= \{p_1, p_2, p_3, p_4\} \\
T &= \{t_1, t_2\} \\
A &= \{(p_1, t_1), (p_2, t_1), (t_1, p_3), (p_3, t_2), (t_2, p_4)\} \\
\omega &= \{1, 1, 1, 2, 1\} \\
I(p_1) &= I(p_2) = \emptyset,\ I(p_3) = \{t_1\},\ I(p_4) = \{t_2\} \\
I(t_1) &= \{p_1, p_2\},\ I(t_2) = \{p_3\} \\
O(p_1) &= O(p_2) = \{t_1\},\ O(p_3) = \{t_2\},\ O(p_4) = \emptyset \\
O(t_1) &= \{p_3\},\ O(t_2) = \{p_4\}
\end{aligned}
$$

2.2 Petri Net Marking

A PN graph is typically used to represent a discrete event system graphically. The events that drive the system are represented through transitions and the information that describes the conditions for the events to occur are the places. In order for the system to operate, a certain medium is required to indicate if the conditions are met or not. This mechanism is shown through the use of tokens. Each place contains an integer number of tokens that essentially indicates that the condition described in the specific place has been met. The way in which tokens are assigned is defined as a marking. In a PN graph, it is represented as black dot as shown in Fig. 2.1. A marking x is represented as,

$$x : P \rightarrow M = \{0, 1, 2, 3, ...\}\,. \tag{2.10}$$

Similarly, $M(p_i)$ or m_i denotes the number of tokens in place p_i where, i is the number of the place. Thus, marking x defines a column vector called the marking vector and the number of elements in the vector is equal to the number of places in the Petri net. If a place has no tokens in it, then that particular place has 0 in the marking vector. For example, the marking of Fig. 2.2 is,

$$x = \begin{bmatrix} M(p_1) \\ M(p_2) \\ M(p_3) \\ M(p_4) \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

It is noted that the places p_2, p_3 and p_4 have no tokens and hence have a marking of 0. The marking defines the state of the Petri net. The initial marking of the PN before any transition has fired is called M_o. The initial marking of Fig. 2.2 is the marking vector x presented above. It follows that M_1 is the next marking after a transition has fired then M_2 is the next marking after another transition fires because of M_1 and so on. For simplicity, a marked PN is the same as a PN. The above definitions do not clearly describe the medium for the state evolution. The markings only describe the state of the Petri net. So, state evolution corresponds to marking evolution, which is caused by firing the transitions. The dynamics of the Petri nets shall be discussed in detail in the next section.

2.3 Petri Net Dynamics

The dynamics signify the sequence of events that happen in the system. The transitions denote which events occur at a certain time. In order to simulate the system behavior, the sequence can be identified by the method of token transfer from a place to another. This can be shown by enabling and firing transitions. Generally, a transition is enabled if each of the input places of that particular transition contains at least one token. The token signifies the condition for that event or transition to occur. Since, we have mentioned the use of weighted arcs, the definition is slightly modified below.

A transition $t_j \in T$ is said to be enabled provided,

$$M(p_i) \geq w(p_i, t_j) \quad \forall\, p_i \in I(t_j)\,. \tag{2.11}$$

where $I(t_j)$ is the set of input places to transition t_j. This equation means that transition t_j is enabled when the number of tokens in each of the input places of t_j, denoted by the marking $M(p_i)$, exceeds or equals the arc weight from p_i to t_j. Once a transition is enabled, it can fire at any time instance. However, it does not imply that the enabled transition will immediately fire.

According to [20], once an enabled transition fires, a state transition f can be defined. This describes the change in the state of the Petri net due to the firing of the enabled transition. The state transition function $f : \mathbf{N}^n \times T \rightarrow \mathbf{N}^n$, is defined for transition $t_j \in T$ if and only if the transition is enabled. Mathematically,

$$M'(p_i) = M(p_i) - w(p_i, t_j) + w(t_j, p_i), \qquad i = 1, 2, ..., n\,. \tag{2.12}$$

In other words, we can express the marking of an input place p_i after a transition has fired if $f(M, t_j)$ has been defined. Eq. 2.12 denotes that when a transition fires from the input place p_i, tokens equal to the arc weight relating that place to t_j, or $w(p_i, t_j)$, are removed first. Then, tokens equal to the arc weight relating t_j to p_i, $w(t_j, p_i)$, are added to place p_i. Let us consider the following example below.

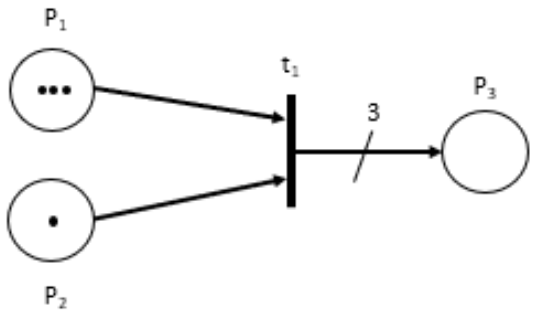

Fig. 2.3.: Enabling a transition

In Fig. 2.3, the arc weight of both p_1 and p_2 to t_1 is one. Place p_1 has three tokens that are greater than the arc weight while p_2 has one token that is equivalent to the mentioned arc weight. Thus, transition transition t_1 is enabled based on Eq. 2.11. The same graph is used in Fig. 2.4 with the slight difference in the number of tokens.

The input place p_1 has two tokens which is greater than the arc weight going into t_1 while the input place p_2 has no tokens. This makes transition t_1 not enabled. The condition for enabling transitions must be satisfied by all input places connected to that specific transition. This is proven in Fig. 2.3, making transition t_1 ready to fire.

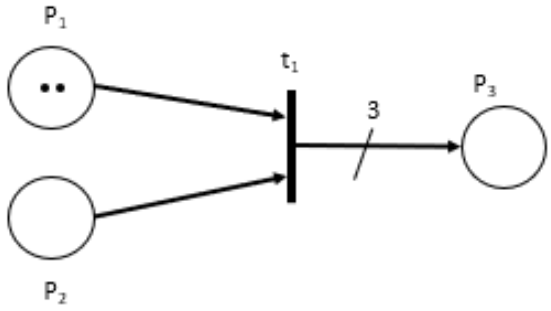

Fig. 2.4.: Transition not enabled

Next, the enabled transition t_1 is fired. After a transition fires, the number of tokens added in each output place is equivalent to the arc weight that connects the transition that fired and the output place. An example of this is presented in Fig. 2.5. In this PN structure, place p_1 has three tokens and place p_2 has a single token. When transition t_1 fires, two tokens are taken out from p_1 while one token is taken out from p_2. Four tokens are then added to place p_3 as dictated by the arc weight. It is observed that the tokens removed from the input places do not need to be equal to the tokens added to the output places. The removal and addition of tokens are solely dependent on the arc weights. Hence, conservation of tokens is not considered in the firing of transitions. Also, it is noted that enabled transitions fire one at a time. This allows observation and interpretation of a system's behavior through a sequence of events.

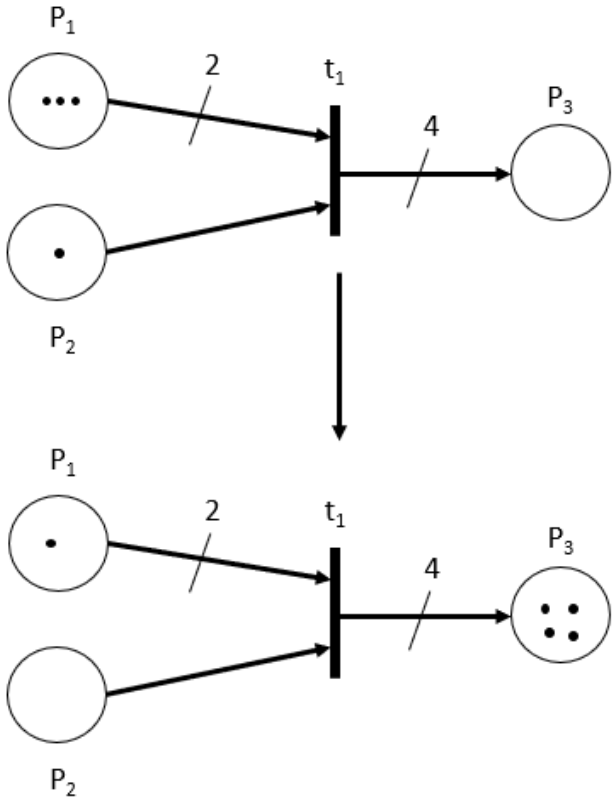

Fig. 2.5.: Firing of an enabled transition

To be able to observe the sequence of transitions that will be fired and their corresponding marking evolution, a reachability tree is needed. This is a graphical representation of the reachable states or markings of the system. Mathematically,

$$R[(P,T,A,\omega,m)] \;:=\; \{y \in \mathbf{N}^n \;:\; \exists \in T^*(f(m,s) \;=\; y)\}\,. \tag{2.13}$$

The equation above defines the set of reachable states where f is the state transition function and m is the marking. The reachability analysis is based on the construction of a tree where the nodes are the states and arcs represent the transitions. The key idea for this analysis is simple and best understood through the example, Fig. 2.6. The first node of the tree is the initial marking or state. The tree ends when it reaches

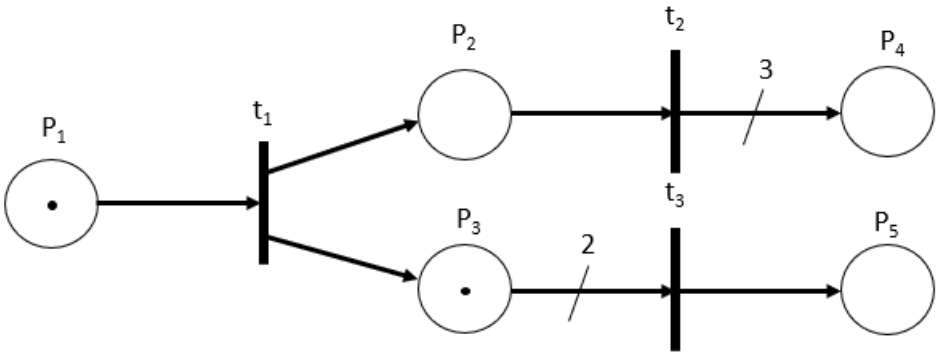

Fig. 2.6.: Petri net example for reachability analysis

a terminal or repeating node. Each node is considered to be a reachable state from the initial marking if a sequence of firing transitions exists. The remaining nodes of the tree is discussed step-by-step below.

1. The initial marking of the Petri net is
$$m_o = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \end{bmatrix}^T.$$

2. Transition t_1 is enabled and will be fired since it satisfies Eq. 2.11. One token is removed from p_1 and one token will be added to places p_2 and p_3. The second marking or state is,
$$m_1 = \begin{bmatrix} 0 & 1 & 2 & 0 & 0 \end{bmatrix}^T.$$

3. Referring back to Eq. 2.11, both t_2 and t_3 are enabled. However, both cannot fire at the same time. This will branch out to two different firing sequences: $S_1 = \{t_1, t_2\}$ and $S_2 = \{t_1, t_3\}$

4. For sequence S_1, one token is removed from p_2 and three tokens are added to p_4. This marking is,
$$m_2 = \begin{bmatrix} 0 & 0 & 2 & 3 & 0 \end{bmatrix}^T.$$

5. After t_2 fires, t_3 will fire for sequence S_2 since it follows Eq. 2.11. Two tokens are removed from p_3 and one token is added to p_5 and this results to a marking,

$$m_3 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \end{bmatrix}^T.$$

6. The markings m_2 and m_3 are called the terminal nodes. The node denotes the end state of the system. It means that no transitions can be enabled/fired after this state. The reachable states from the initial marking are m_1, m_2 and m_3. These three states are said to be reachable because a sequence of transitions that fired connects it to m_o. Graphically, it is shown in Fig. 2.7.

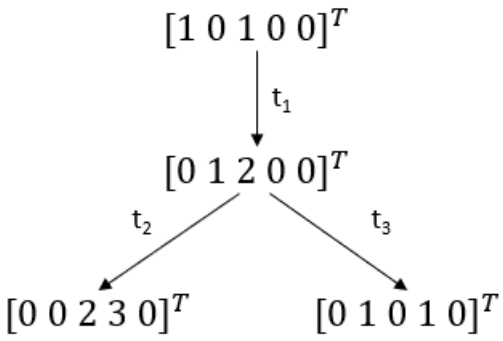

Fig. 2.7.: Simple reachability tree

As presented in Fig. 2.6, the reachability tree is easy to create. However, it may be infinite. An example of which is shown on Fig. 2.8. The underlying task for this Petri net analysis is to find a finite representation, called the coverability tree. If the tree is finite, then the reachability tree and coverability tree are the same.

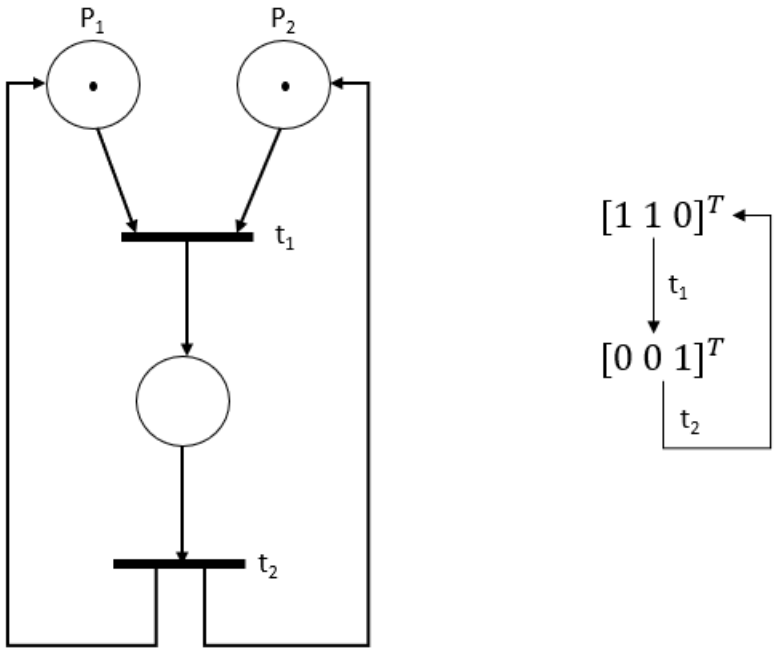

Fig. 2.8.: Petri net with a loop and reachability tree

2.4 Incident Matrix and State Space Equation

2.4.1 Incident Matrix

Any Petri net graph, with n places and m transitions, can be analyzed through its incident matrices. The three incident matrices are an alternative to describing the size of PN.

- Output Incident Matrix, B^+ is an $n \times m$ matrix that captures the arc weights from transitions to output places.
- Input Incident Matrix, B^- is an $n \times m$ matrix that captures the arc weights from input places to transitions.
- Incident Matrix, B is the difference of the output and input incident matrices with the same dimension. It is defined by the mathematical equation,

$$B = B^+ - B^- \tag{2.14}$$

We apply the concept to Fig. 2.6 and generate three unique matrices. There are cases where arc weights are not present between certain places and transitions. Thus, their entry on either the input or output incident matrix is zero. It is also noted that these matrices are only structural properties of the PN, they are independent of the markings or states of the system.

$$B^+ = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad B^- = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{and } B = B^+ - B^- = \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & 0 & -2 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.4.2 State Space Equation

The examples shown previously are small systems that can easily be represented graphically. However, it may be difficult to perform analysis on larger and more complex systems. An alternative is through algebra, the state space equation. It uses both the concept of incident matrices and the enabling of transitions. This equation describes how the state of each place changes when a transition is enabled then fired. It is written as,

$$M_{k+1} = M_k + BX_k \tag{2.15}$$

where,

- M_{k+1} is an $n \times 1$ vector that contains the marking when time is at k+1.
- M_k is also an $n \times 1$ vector that contains the marking at a previous time, k.
- B is the incident matrix.
- X_k is the firing vector with dimension $m \times 1$. It contains a single nonzero entry with value of 1 that indicates which transition is currently firing.

Using Fig. 2.6 as an example, we define its state equation. The initial state is $M_0 = [1\,0\,1\,0\,0]^T$ and when t_1 fires the next state is,

$$M_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & 0 & -2 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

Similarly, M_2 can be defined by using the state space equation. The firing vector has the nonzero entry on the second row to accommodate for transition t_2 and the marking M_k is now M_1. This process is repeated until all enabled transitions fire. As seen from the calculations, only two variables change in the equations. Thus, the use of the state space equation makes identifying the reachable states of the system simpler to determine for larger systems compared to the graphical means of analysis, the reachability tree.

2.5 Continuous Petri Net

In the previous sections, Discrete Petri nets (DPN) have been discussed in-depth. This type defines the sequences of discrete events that can occur in a system. However, most of the the systems in the real-world are not event-driven. The behavior or state of the system changes continuously, like the flow of traffic and supply chain flow at a manufacturing factory. These type of systems cannot be graphically represented by DPN. The solution is to use Continuous Petri nets (CPN) which are driven by time. Most of the definitions, notations and dynamics coincide for both Petri nets. However, there are some distinguishing features that are only applicable to the CPN.

- Referring back to Fig. 2.1, continuous transitions are represented as unshaded boxes rather than shaded bars and the places are represented as two concentric circles instead of one.

- The arc weight connecting places to transitions or vice versa is no longer an integer but any non-negative real number. This enables the system to fire in a continuous flow.
- Since the arc weight can be any real number, the number of tokens need not be an integer as well.
- Each transition has a specific firing quantity, also a non-negative real number.

2.5.1 Dynamics

After differentiating CPN and DPN, we can now define in detail the graph and its dynamics. According to [21], a Continuous Petri net graph is represented as (N, m_o) where N is the Petri net structure, from Eq. 2.1, and m_o is the initial marking of the system. The enabling and firing of transition is as follows,

- A continuous transition $t_j \in T$ is enabled at any marking m, if and only if,

$$\forall\, p_i \in I(t_j), m_i > 0\,. \tag{2.16}$$

- The enabling degree of a particular transition t_j is,

$$enab(t_j, m) = \min_{p_i \in I(t_j)} \left\{ \frac{m_i}{B^-(p_i, t_k)} \right\} \tag{2.17}$$

- An enabled transition t_j can fire in any real amount, α, or mathematically, $0 \leq \alpha \leq enab(t, m)$

2.5.2 Token Flow Matrix and State Space Equation

The incident matrix and state space equation is similar to that of the discrete Petri net. However, the incident matrix is called a token flow matrix for a continuous Petri net. The matrix has the same dimension of $n \times m$, given a Petri net with n places and m transitions.

$$M' = M + BV \tag{2.18}$$

where,

- M is an $n \times 1$ vector that contains the initial marking of the system.
- M' is an $n \times 1$ vector that contains the new marking reached from the initial marking.
- B is the token flow matrix.
- V is the firing vector with dimension $m \times 1$. This contains either a zero or nonzero firing rate that corresponds to the continuous transitions. This firing rate can be any real number.

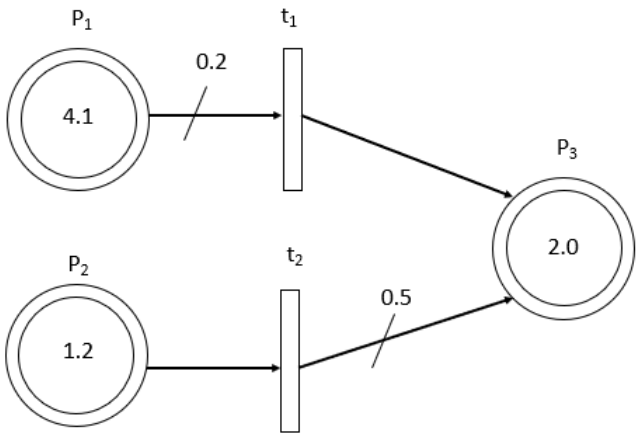

Fig. 2.9.: Example of a continuous Petri net

Applying the concepts to the example in Fig. 2.9, we observe that transitions t_1 and t_2 are enabled by Eq. 2.17. From the initial marking is $M = [4.1\ 1.2\ 2.0]^T$, the new marking reached is $M' = [3.9\ 1.2\ 3.0]^T$ when t_1 is fired. Then if t_2 fires, the new marking is $M' = [4.1\ 0.2\ 2.5]$. Both transitions are still enabled as long as the number of tokens in p_1 satisfy the conditions mentioned previously. From the scenario indicated, an infinite number of reachable marking or states can be created due to the constant firing rate. Thus, limiting the use of the reachability tree. Hence,

macro marking is used to represent the possible reachable states in a finite way [22]. Mathematically, for a CPN with n places, the total number of markings will be 2^n. Using the same example, Fig. 2.9 has 3 continuous places. This means that there are $2^3 = 8$ macro markings. These markings are $[0\ 0\ 0]^T$, $[m_1\ 0\ 0]^T$, $[0\ m_2\ 0]^T$, $[0\ 0\ m_3]^T$, $[m_1\ m_2\ 0]^T$, $[0\ m_2\ m_3]^T$, $[m_1\ 0\ m_3]^T$ and $[m_1\ m_2\ m_3]^T$. Since the marking of p_3 can never be zero, the macro markings that have m_3 or $m(p_3)$ are omitted for this case. The final generalized macro marking for the given example is provided in Fig. 2.10.

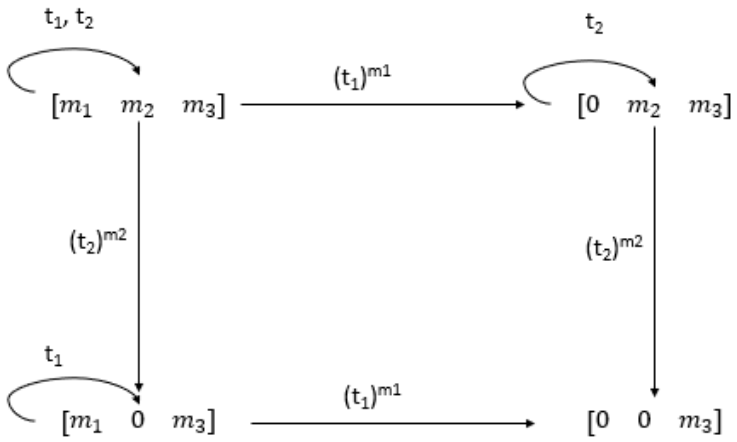

Fig. 2.10.: Macro marking

2.6 Hybrid Petri Net

So far, we've discussed the similarities and differences between DPN and CPN. By letting the two systems interact, a dynamic Hybrid Petri net is created. In a DPN, the marking may correspond to a Boolean state, e.g. door open or close, or an integer value. It is then followed to find all the reachable states of the system as general method for analysis. However, in the case of markings that contain a large number of tokens or when the transitions are time-driven, this method is not practical. This observation has opened the way to use CPN and Hybrid Petri Nets for refined model analysis.

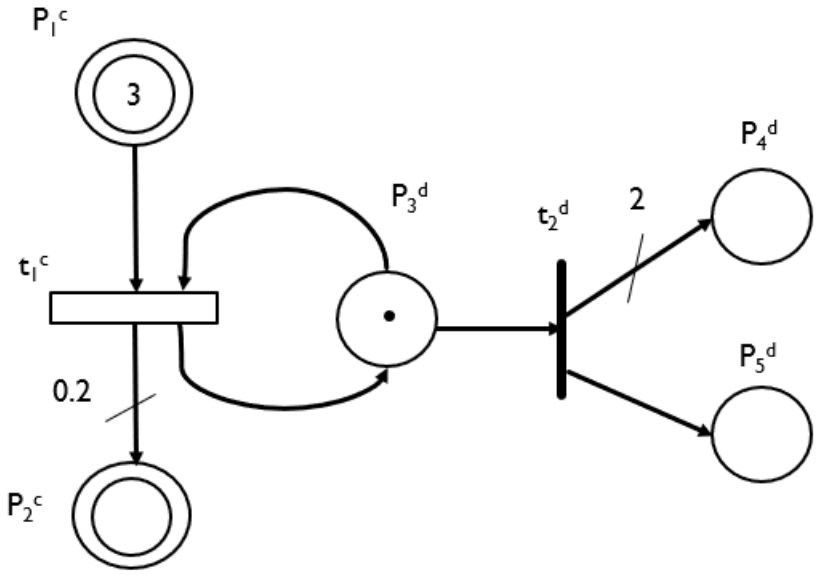

Fig. 2.11.: Example of a Hybrid Petri net

2.6.1 Notations and Definitions

According to [23], a Hybrid Petri net is a sextuple $Q = \{P, T, Pre, Post, m_0, h\}$ such that,

- $P = \{p^d \cup p^c\}$ is a finite set of discrete and continuous places.
- $T = \{t^d \cup t^c\}$ is a finite set of discrete and continuous transitions.
- Pre denotes the input incident matrix, B^-
- $Post$ denotes the output incident matrix, B^+
- m_0 is the initial marking or state of the Petri net.
- h is a hybrid function that denotes whether a node is discrete, connecting a discrete place p^d and transition t^d, or continuous for connecting a continuous place p^c and transition t^c. Mathematically, $h: \ P \cap T \ \rightarrow \{D, C\}$

One important requirement for modeling this type of petri net is that if an arc connects a continuous transition to an input discrete place, then an arc that is a reciprocal of this relationship must exist as well. It ensures that the marking of any discrete place is always an integer for any transition that fires. An example is the arcs connecting t_1^c and p_3^d in Fig. 2.11. If there is a discrete place p_i^d and continuous transition t_j^c, this rule can be verified and expressed as,

$$B^-(p_i^d,\, t_j^c) = B^+(p_i^d,\, t_j^c)$$

.

Using Fig. 2.11, the simple hybrid Petri net is described as follows,

- $P = \{p_1^c\,, p_2^c\,, p_3^d\,, p_4^d\,, p_5^d\}$
- $T = \{t_1^c\,, t_2^d\}$
- $Pre = B^- = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
- $Post = B^+ = \begin{bmatrix} 0 & 0 \\ 0.2 & 0 \\ 1 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}$
- $m_0 = [3\ 0\ 1\ 0\ 0]^T$

2.6.2 Dynamics

The conditions for enabling and firing the transitions are dependent on the type. Discrete transitions follow the same rule implied by Eqs. 2.11 and 2.12. Continuous transitions should satisfy either of the two conditions,

- For each discrete input place p_i^d, $m(p_i^d) \geq B^-(p_i^d, t_j^c)$.
- For each continuous input place p_i^c, $m(p_i^c) > 0$, from Eq. 2.16.

The format of the state equation is similar and is written as,

$$M_1 = M_0 + Bs\,. \tag{2.19}$$

In the equation above, B is the incident matrix, M_0 is the initial marking or state of the system and M_1 is the next reachable state after a transition has fired. s is the characteristic vector that contains a single nonzero entry, which is either an integer for the number of firings of a discrete transition or a real number for the firing quantity of a continuous transition. It is also observed that in Eq. 2.15 all the components of X are integers while s contains either integers non-negative real numbers.

3. HYBRID PETRI NET MODEL

The modeling for this thesis was done to combine the performance of Lane Departure Warning (LDW) and Lane Keeping Assist (LKA) systems. These two systems can act independently of each other or sequentially depending on the car make. This specific model follows the sequential mode where LDW occurs before LKA and both systems are taken as one. It should also be noted that the LKA system is not the same as Lane Centering Assist [24]. This Petri net model uses an LKA system that provides steering intervention as the vehicle performs unintentional lane departure after the warning has been sent. Other assumptions were taken into account before the model design. They are as follows,

- The vehicle of interest is driving on a straight path.
- The road is even with visible lane markers.
- Post-processed data is readily available for the system to use.

3.1 System Flowchart

The first step in modeling is the creation of a flowchart. It is a sequence of steps and decisions the system will follow. Since the Lane Keeping Support System (LKSS) is a decision-based process, it needs inputs from the environment and compare them with pre-defined values to deduce if the vehicle is safe or not. The flow in Fig. 3.1 begins when the engine is switched on. This powers up the cameras and sensors of the vehicle. The images are captured and post-processed while the sensors gather the steering angle, torque, speed and current position of the vehicle. At this instant, the system has a clearer picture of the situation of the vehicle from all the inputs acquired. The initialization stage comes after. This stage checks three requirements for LKSS

to be activated. The first condition is that the turn signals are off, which means that there are no intentions of lane departure. The current speed of the vehicle should be in a safe or cruising range secondly. The last condition is that the steering angle denotes that the car is currently driving in a straight path. If all three conditions are met, then the initialization stage is complete and the LKSS is activated. If one requirement is not met, the system will continue gathering data and checking for all three requirements before system activates. This gives the driver more control over the vehicle and is able to show the clear intention of possible lane change.

Once the LKSS turns on, input data is refreshed and the system checks if the vehicle in driving in a safe range defined in the system. The three values checked are the speed, steering torque and position of the vehicle with respect to the lane markers. If any of the three parameters is out of the safe range, the timer is checked if two seconds have passed. If the timer has not started, this is the system's first encounter of unwanted lane departure. Based on the data gathered, the system decides if it's a right or left lane departure and immediately sends out the warning by means of an audible sound, vibration of the steering wheel and/or a visible warning on the dashboard. As soon as the warning is sent, the timer begins. This period freely gives the driver the control to return inside the lane boundaries. While the driver is making the minor changes to his steering, the system continuously checks if any of the three mentioned parameters, speed, steering angle and position, are still out of the safe range or not. If any condition has been met and that timer has passed a second, the lane keeping assist sub-system takes over. It gently steers the vehicle back to its through a torque command and brakes are applied dependent on the gravity of the vehicle's situation. The sub-system takes control of the vehicle for a few seconds. The vehicle is once again in a safe driving mode, the timer resets, all warnings removed and the LKSS flow repeats. From this flowchart, the detailed procedure for modeling the system is discussed in the next sections.

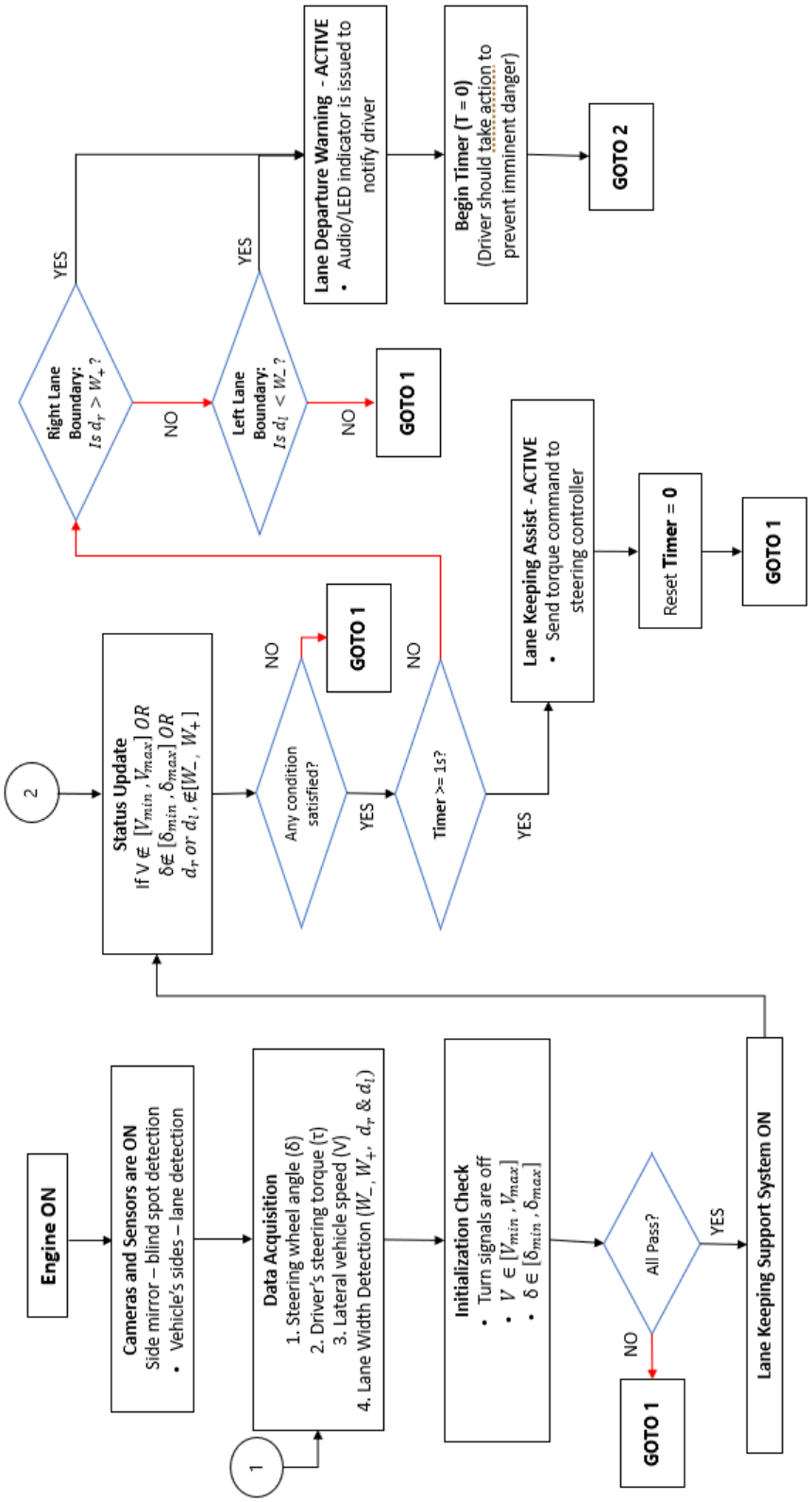

Fig. 3.1.: Flowchart for Lane Keeping Support System

3.2 Discrete Petri Net Model

The beginning stage of the Petri net model was designed by using only discrete transitions and places. This provides an easier representation since most of the decisions that occur in the system are high-level and event-triggered. Two versions of the discrete Petri net model are described and include the drawbacks and improvements made to reach a final model. The notation introduced in Section 2.6.1 for distinguishing continuous and discrete Petri nets has been omitted in this section. The superscripts are taken out for simplicity.

3.2.1 Version 1

The first version is divided into two areas, data acquisition and system operation. The modeling was done separately and later combined. Fig. 3.2 begins with the starting the engine in p_1 that activates all cameras and sensors in the vehicle. The functionality of the sensors are taken as on place, p_{sensor} to signify if they are working properly. This is a condition for data acquisition. Initialization stage follows after and checks all three conditions. If these are met, the Lane Keeping Support System (LKSS) turns on. The system identifies if one of the three parameters, steering angle speed and width, are in the safe range. If one of these parameters, represented as the places p_{aref}, p_{vref}, p_{wref} respectively, show an unintentional lane departure then a token is sent to the other half of the model. As soon as the vehicle is considered in an unsafe condition, the LDW sub-system verifies if it's a lane departure on the left side, right side or a false alarm. A warning is sent to the driver as soon as it confirms the current state of the vehicle. This warning also sets the timer to begin counting up. Once p_{37} receives a token, LKA system immediately takes over and steers the vehicle back to the center of the lane and returns to data acquisition.

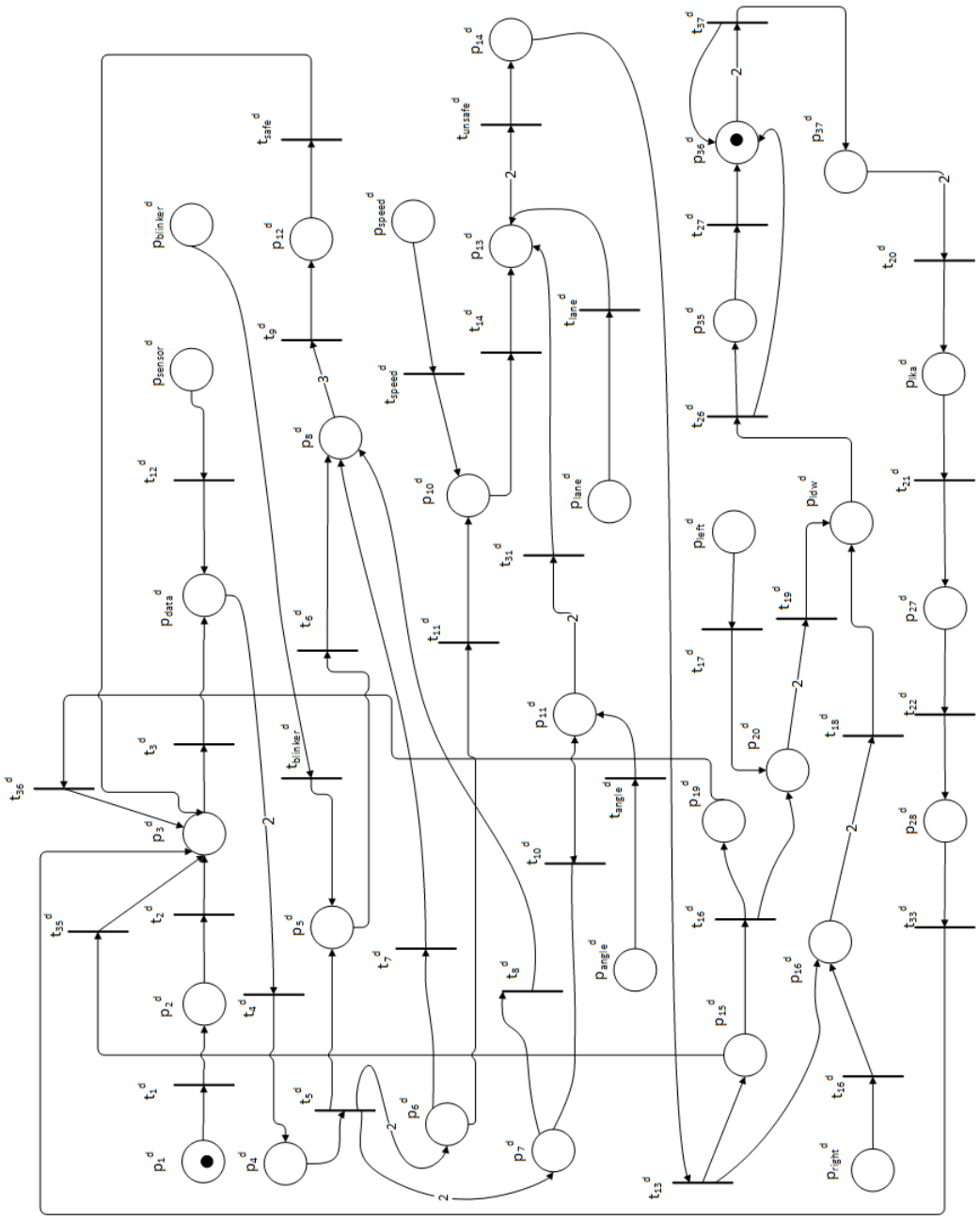

Fig. 3.2.: Version 1 of Discrete Petri Net Model for LKSS

Flaws of the Model

This version has posed several flaws. One is the complex timer logic. The function of the timer does work, however it unnecessarily increases the size of the incident matrix. Also, there are extra transitions that can be merged into one when lane departure is checked. Multiple input arcs, with a weight greater than one, and corresponded to an output arc weight of one have produced unbounded places for simulation. Unbounded places mean that a certain place can have an infinite number of tokens. This may give false information to the system for making time-based decisions. The model also included redundant places that can be combined to a single place such as p_{13}, p_{15} and p_{33}. All these flaws were removed in the next version of the model.

3.2.2 Version 2

The flow of the second version is kept original, based on the flowchart in Fig. 3.1. The modeling was done as a whole that combines both LDW and LKA systems. Referring to Fig. 3.3, the flow begins at p_1. The system powers all cameras and sensors in the vehicle. All places for determining an unintentional lane departure has been retained as well as the places for describing if the vehicle in in a safe condition or not. The three major changes for this model are the timer, the arc weights, and the introduction of system looping. Firstly, the timer was simplified to creating a small loop with just two places rather than four. All of the arcs have a weight of one, except for the placeholder for the timer, p_{15}. Lastly, output arcs from two discrete transitions, t_{safe} and t_{unsafe}, have been added to incorporate a system looping. In reality, the places p_{sensor}, $p_{blinker}$, p_{aref}, p_{wref} and p_{vref} will receive tokens only if the vehicle encounters each event. This set-up does not produce accurate results for continuous simulation. Hence, the arcs were added to force these places to receive a token every single time.

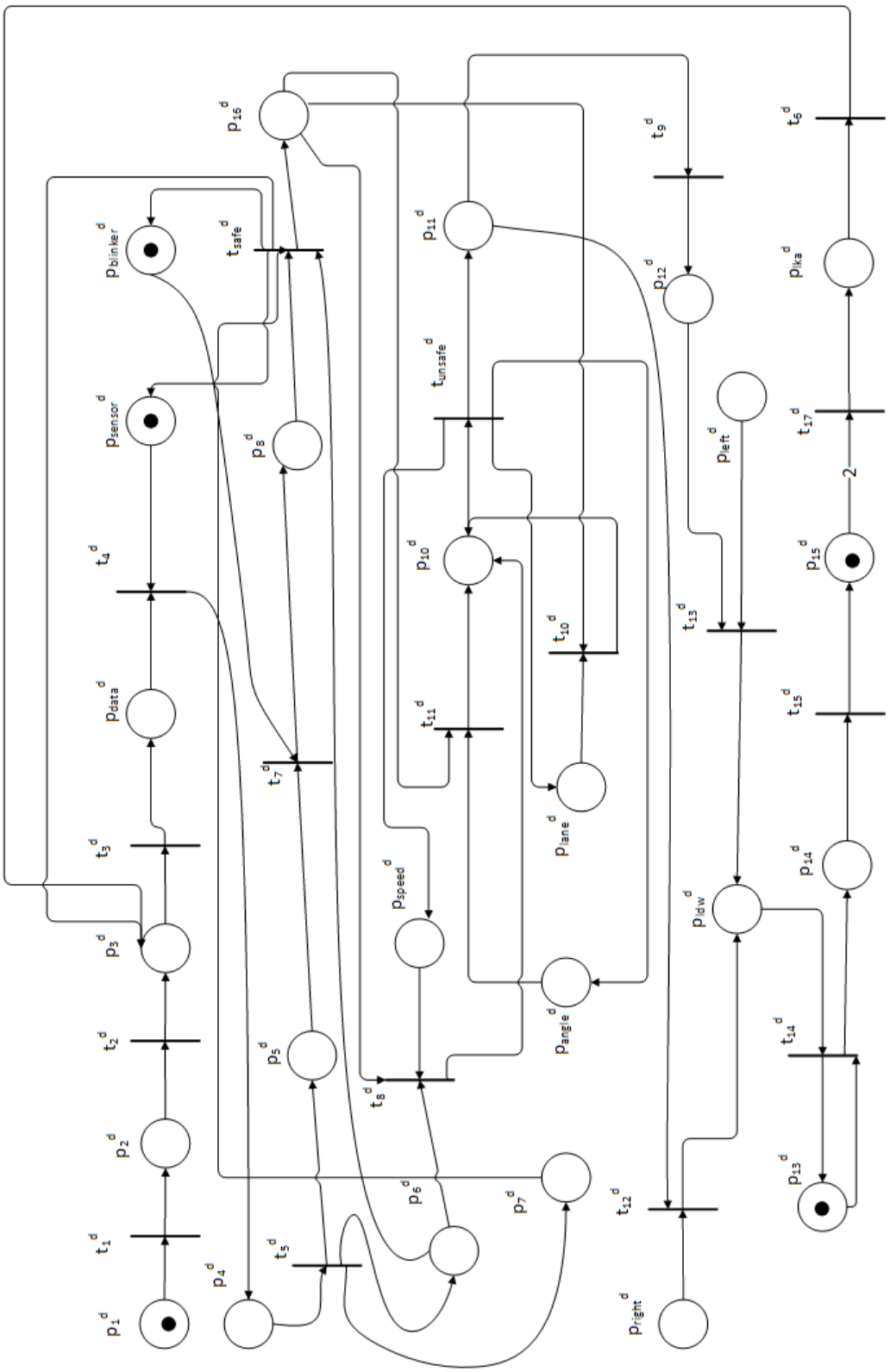

Fig. 3.3.: Version 2 of Discrete Petri Net Model for LKSS

Flaws of the Model

Overall, this model has greatly reduced the size of the incident matrix by removing redundant places and simpliying the timer. It has also changed the Petri net firing dynamics with arcs weighted at one. However, the system looping presented a more difficult task of analyzing the results. A successful and well-described simulation requires introducing various cases, represented by tokens in different places, and intentionally placing a loop on the model will yield to unreliable results. This has posed a new challenge to the model that led to the final version of the discrete Petri net model.

3.2.3 Final Model

The last and final Discrete Petri net model of the Lane Keeping Support System has retained the same functionality with the previous versions. The timer for the second version has also been kept the same. Referring to Fig. 3.4, the place, p_{13}, has an initial token of one to compensate for the reduced arc weights. The solution for the forced system looping in the previous version is to remove the extra arcs and instead create different scenarios for simulation purposes. The scenarios are distinguished by the tokens that are initially place to signify a possible situation during vehicle operation. These scenarios are discussed in detail in the succeeding section.

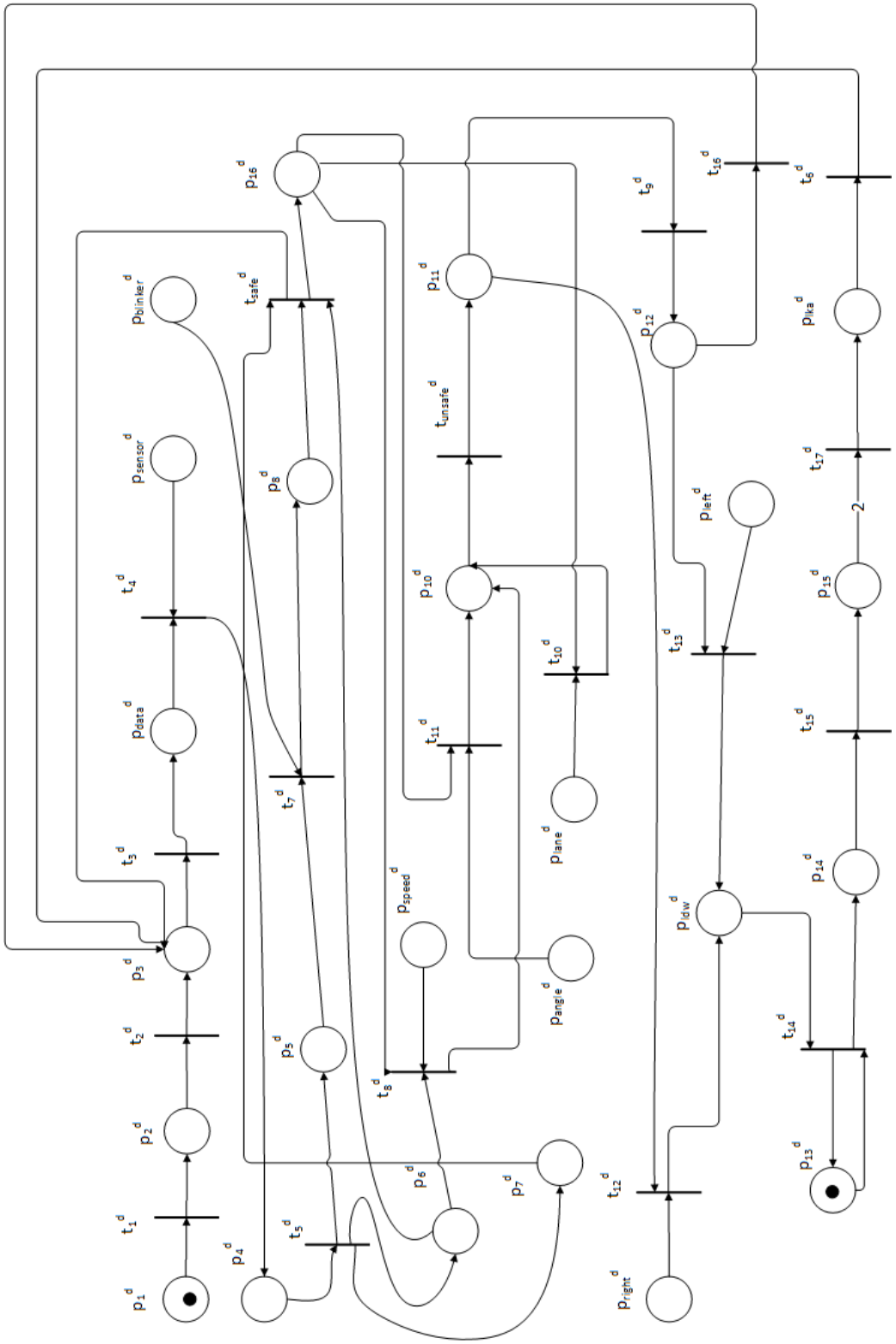

Fig. 3.4.: Final Discrete Petri Net Model for LKSS

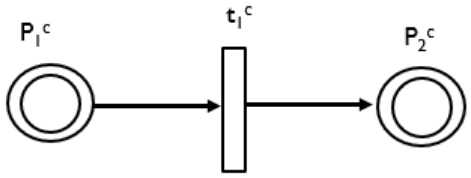

Fig. 3.5.: Final Discrete Petri Net Model for LKSS

3.3 Hybrid Petri Net Model

The final discrete Petri net model is sufficient to show the different cases a driver may encounter and activate the LKSS. Still, we can optimize the model by reducing redundant discrete places and introducing continuous places for a real-time system such as this. The continuous dynamics taken into account for this model is the time. The integration of the continuous Petri net is shown in Fig. 3.5. The continuous place, p_1^c denotes the threshold time before the LKA system takes over the vehicle while p_2^c keeps collecting the tokens from the threshold, denoting the time elapsed. The continuous transition, t_1^c, keeps firing. When the marking of p_2^c reaches two, a discrete transition fires and deposits a token at the assigned output place. This indicates that the timer has reached two seconds and the system should check if there is still unintentional lane departure after the warning has been sent. The period between zero and two seconds is the chance given to the driver to steer back the vehicle to the center of the lane. If the driver has failed to do so, a token will be sent to the LKA system to perform steering assist and guide the vehicle back to safety. Once this happens, the timer will be reset to zero by the system internally. The hybrid petri net model for a Lane Keeping Support System is shown in Fig. 3.6.

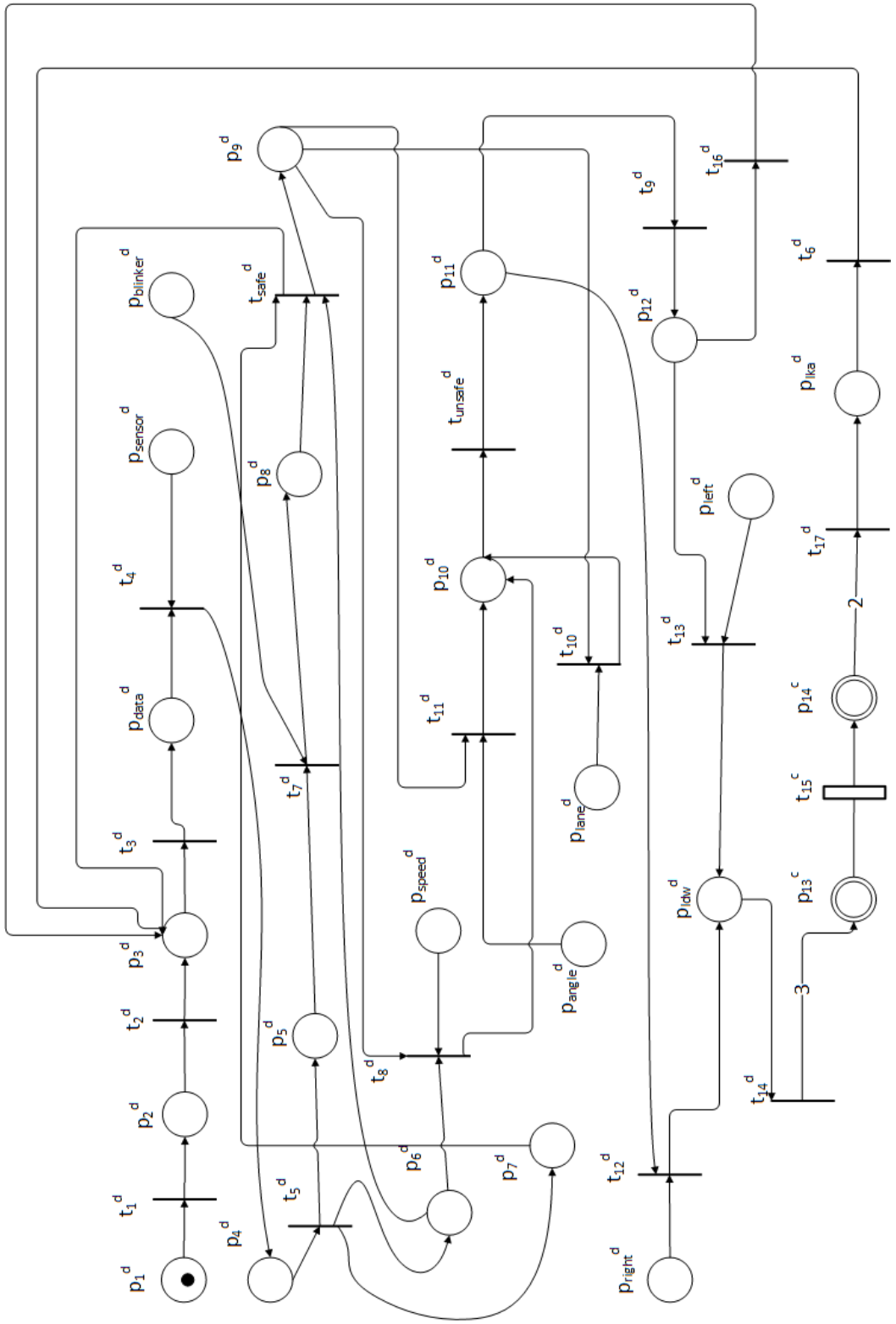

Fig. 3.6.: Final Hybrid Petri Net Model for LKSS

3.3.1 Description of Nodes

The transition to a Hybrid Petri Net has reduced the number of discrete places to twenty-four. The nomenclature for differentiating the continuous and discrete nodes illustrated in Fig. 2.11 is used. The description for each place and transition is provided in Tables 3.1 and 3.2.

Table 3.1.: List of transitions in the HPN model

Beginning of Table	
Transition	**Description**
t_1^d	Engine is on, sends signal to turn on cameras and sensors.
t_2^d	Cameras and sensors are on.
t_3^d	System acquires post-processed data from sensors.
t_4^d	Data acquisition is done, triggers system initialization.
t_5^d	Initialization done, begin with signal and in-range checks.
t_6^d	LKA is active, sends steering torque command to engine.
t_7^d	Transition indicates that turn signal is off.
t_8^d	Transition indicates that vehicle's velocity is not in range.
t_9^d	Transition indicates that the right side boundary is safe then proceeds to check the left side.
t_{10}^d	Transition indicates that vehicle's lateral position is out of range.
t_{11}^d	Transition indicates that vehicle's steering angle is out of range.
t_{12}^d	Transition indicates that front right tire is on the right line.
t_{13}^d	Transition indicates that front left tire is on the left line.
t_{14}^d	Lane departure warning has been sent to driver and triggers timer.
t_{15}^c	Transition for continuous time flow.

Continuation of Table 3.1	
Transition	**Description**
t_{16}^{d}	Transition indicates that there is no lane departure.
t_{17}^{d}	Transition indicates that time for correction is done and the LKAS will take over the vehicle.
End of Table	

Table 3.2.: List of places of the HPN model

Beginning of Table	
Place	**Description**
P_{1}^{d}	Engine is on.
P_{2}^{d}	Cameras and sensors are on.
P_{3}^{d}	Data gathering begins.
P_{4}^{d}	Vehicle controller processes data gathered and sets all initial conditions.
P_{5}^{d}	Vehicle controller checks if turn signal is on or off.
P_{6}^{d}	Vehicle controller compares actual and reference vehicle speed.
P_{7}^{d}	Vehicle controller compares actual and reference steering angle.
P_{8}^{d}	Place indicating the final status of the turn signal. A token signifies that turn signals are off.
P_{9}^{d}	Lane Keeping Support System is activated. A token inside signifies that the vehicle is in an safe condition.
P_{10}^{d}	A token inside signifies that the vehicle is in an unsafe condition.
P_{11}^{d}	Vehicle controller checks for right line boundary.
P_{12}^{d}	Vehicle controller checks for left line boundary.
P_{right}^{d}	Vehicle controller has determined that front right tire has crossed right white line.

Continuation of Table 3.2	
Place	**Description**
P^d_{left}	Vehicle controller has determined that front left tire has crossed left white line.
P^d_{ldw}	Place indicating that unintentional lane departure has been detected and warning is sent to driver.
P^d_{lka}	Place indicating lane keeping assist feature takes over.
P^d_{sensor}	Place indicating that all sensors are working properly.
P^d_{turn}	Place indicating the current status of turn signal. A token means signals are off.
P^d_{speed}	Place indicating if current speed is safe or unsafe. A token means speed is outside the safe range.
P^d_{angle}	Place indicating if current steering angle is safe or unsafe. A token means angle is outside the safe range.
P^d_{lane}	Place indicating if current position of vehicle is safe or unsafe. A token means that position is outside the safe range.
P^d_{data}	Place indicating that sufficient data has been gathered.
P^c_{13}	Place indicating the time duration.
P^c_{14}	Place indicating the time elapsed.
End of Table	

3.3.2 Functionality of the Model

For accurate and reliable simulation results, the forced system looping from the previous version of the model has been removed. The solution for this is to test the model based on four different scenarios the driver and vehicle may encounter. The working of the model is discussed in detail based on the scenario or case.

Case 1: LKSS activates and vehicle is safe

The first scenario depicts that all three conditions are met for the initialization stage to be successful and the vehicle is in a safe condition. The flow begins when the engine is started at p_1. The sensors and cameras are activated and denoted by a token in p_2. Data gathering is initiated by p_3. When all the sensors are functioning as expected, indicated by a token in p_{sensor}, and enough data has been gathered, t_4 fires to p_4. This part of the model is where the reference values are set and the three parameters, turn signals, speed and angle are checked. The places p_6, p_7 and p_8 denote that these three parameters have passed the requirements and tokens are sent to the transition, t_{safe}. Once enabled, a token is then fired from this transition to p_9 which activates the Lane Keeping Support System (LKSS). Since this scenario denotes that there is no unintentional lane departure detected, the same transition sends a token back to p_3. The process then repeats and continues checking if the same three parameters pass the requirement or not. The flow is only involved with the activation of LKSS which describes the first scenario. The next scenario explains the case system not reaching activation.

Case 2: LKSS never activates.

The second scenario shows the two causes of why the system fails activation. The first reason being if the turn signal is one. In reality, if the driver decides to depart from the current lane then the left or right turn indicator is lit up. There is no reason for the system to activate for this case. Starting from p_1, and with the sensors working as expected, the system gathers and processes data. Once the initialization stage is reached, the first parameter fails the requirement. The tokens remain at places, p_5, p_6 and p_7, until p_{turn} contains a token reflecting that turn signals are off. The transition, t_safe will never fire therefore not activating LKSS feature.

The other reason for the system not being activated is any of the sensor or cameras malfunction. As soon as the engine is started, the sensors and cameras are turned on and denoted by a token in p_2. However, if one of these devices have failed to turn on or gather data, the place p_{sensor} has no tokens. Effectively, transition t_4 will never be enabled and cannot fire to begin the initialization stage. Hence, LKSS never activating as well. The flow does not continue to checking specific lane departure and timer countdown because this scenario does not include that. The next two cases present unintentional lane departure.

Case 3: LKSS activates and unintentional lane departure was corrected.

This scenario goes through with LKSS activation like the first case. Once the engine starts, the sensors and cameras begin data gathering, data post-processing and goes through with the initialization stage.The places p_5, p_6 and p_7 have a token each. Continuing at the enabled transition, t_{safe}, the system suddenly encounters an unsafe operating condition. This may be represented by a token in either, p_{speed} which indicates that the current speed of the vehicle is unsafe, p_{angle} which indicates that the steering angle of the vehicle is not at zero degrees, or p_{lane} which indicates that the current position of the vehicle is outside the lane boundaries. This will enable transitions t_8, t_9, or t_{11} respectively. The enabled transition depicting that the vehicle is unsafe sends a token to p_{10} which then enables transition, t_{unsafe}. This transition fires a token to p_{11} which begins the process for determining a left or right line boundary. During this time, there is a possibility that the driver notices the vehicle drifting away from the lane and has responded quickly to steer the vehicle back to the center of the lane. This action does not yield any token on either p_{right} or p_{left}. Thus, sending the token from p_{11} to p_{12} and then back to p_3 for data gathering. In this case, unintentional lane departure was identified by the LKSS, however the warning was never sent due to the driver's quick action.

Case 4: LKSS activates and takes over the vehicle.

The last case presents the instance where the driver fails to correct the unintentional lane departure before timer's done. The LKSS activates and the system recognizes an unsafe condition denoted by a token in p_{11}. The system now checks which specific direction the vehicle is drifting away, denoted by a token in p_{right} or p_{left}. In the case of a right line boundary, transition t_{12} is enabled. A left line boundary allows t_{13} to be enabled. Once either transitions are enabled, a token is fired to place p_{ldw} and a warning is immediately sent to the driver. The enabled transition t_{14} fires to the continuous place p_{13}. With the continuous transition, t_{15}, surrounded by arc weights of 0.5, there is a continuous flow similar to that a timer counting up. Since the previous enabled transition, t_{16}, the system begins with the data gathering and checks if the vehicle has been steered back to safety during this time. If there is unintentional lane departure, another token is fired from t_{14} to the continuous place p_{13} to continue counting. As soon as the place p_{14} contains two tokens, t_{17} is enabled and fires to p_{lka}. This gives permission for the LKSS to take over and steer the vehicle back to center of the lane. The system does not end here and a token is fired to p_3 again to initiate data gathering.

4. SIMULATION RESULTS

In order to determine if the four scenarios in the previous chapter can be observed in the model, simulations should be carried out. Firstly, the simulation tools, PN Toolbox, MATLAB and SimHPN, are discussed in this section. The results obtained from each scenario is then presented and justified.

4.1 Simulation Tools

This section provides a background on each tool that was used for simulation. The purpose, interface and operation of each is also discussed. One tool was used only for modeling and simulating the discrete Petri net model while the other tool was used for simulating the hybrid Petri net model. Both tools are embedded in the MATLAB environment. The same environment was used to support and justify the results for the former tool.

PN Toolbox

The first tool, short for Petri Net toolbox, is a third-party software tool incorporated in the MATLAB environment. It is a tool used for modeling discrete event dynamic systems [25]. It has two main features - draw and simulate. The former allows the user to draw, store and retrieve Petri nets while the latter feature permits simulation and analysis of the created PNs. It has a simple graphical user interface (GUI), seen in Fig. 4.1 that has pre-defined nodes and arcs. As mentioned, this tool was used only for modeling and analysing the DPN model. It was used to acquire the incident matrix and the reachability tree.

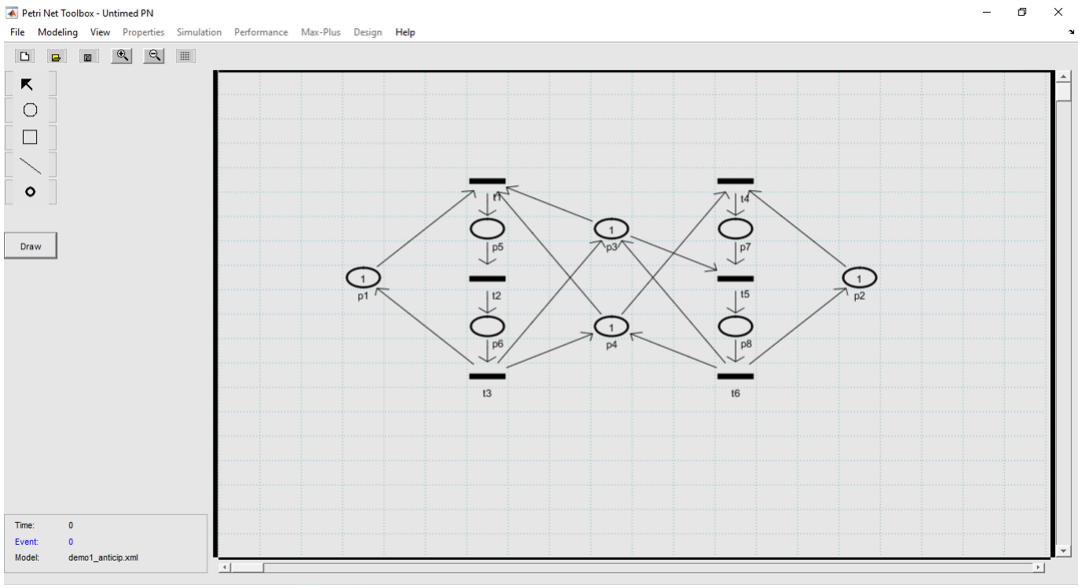

Fig. 4.1.: PN Toolbox GUI

SimHPN

SimHPN is another embedded package in the MATLAB environment. The simulator supports infinite server and product server semantics for the firing of both, discrete and continuous, transitions. Data related to hybrid Petri net model and the output results are stored as MATLAB variables. Fig. 4.2 shows the GUI of the simulator. The inputs describing the net structure are named as follows,

- *Pre* is an $n \times m$ matrix, where n is the number of places and m is the number of transitions, which contains the arc weights from places to transitions.
- *Post* is an $n \times m$ matrix which contains the arc weights from transitions to places.
- M_o is an $n \times 1$ vector that contains the initial marking.
- *Lambda* is an $m \times 1$ vector that contains the firing rates of the transitions.

- $T.Type$ or $Trans.Type$ is an m x 1 vector that indicates the type of transition. It is c for continuous transition, d for stochastic discrete and q for deterministic discrete transitions [26].

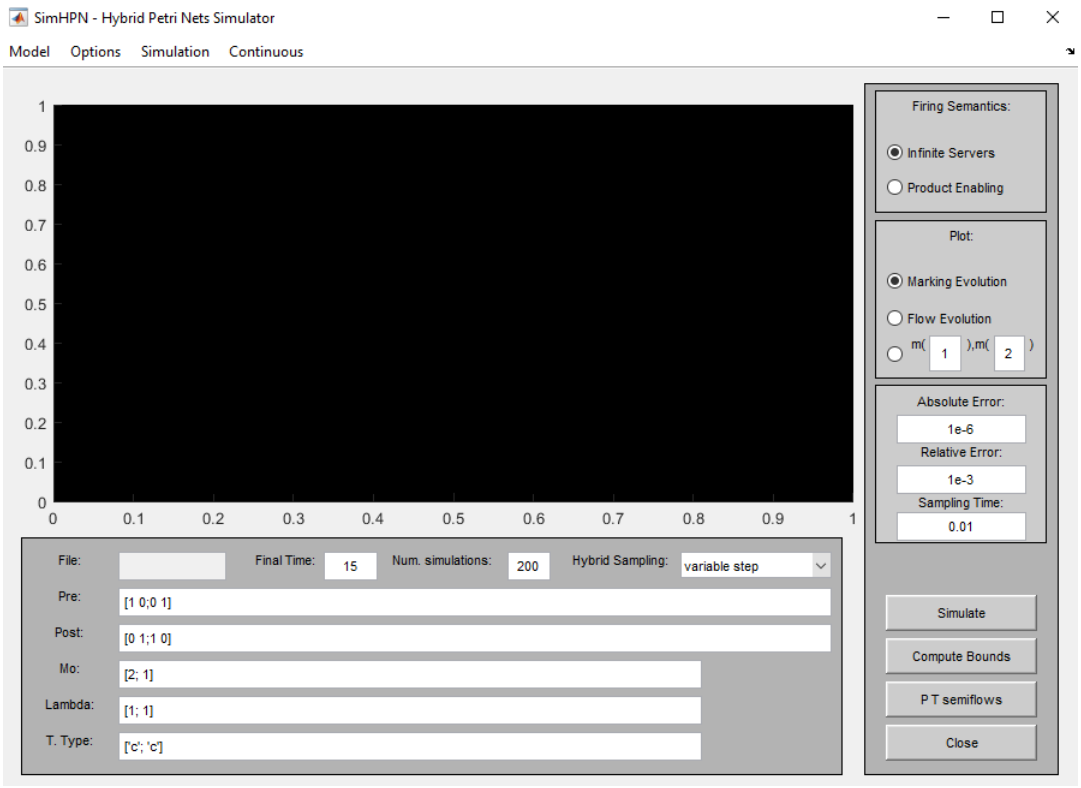

Fig. 4.2.: SimHPN GUI

The output of the tool is saved as a *.mat* file. It may be represented as either the marking evolution or the flow (transition) evolution of the HPN model considered. This tool was used for observing the behaviour of the final HPN model when the four scenarios are introduced. It graphically presented the initial to final marking of all the places.

MATLAB

The last tool is the general environment for the entire simulation. The simulation results from each toolbox was supported by creating an algorithm that finds the reachable states by using the state space equation. This provided a mathematical representation of the behaviour of the HPN model based on each scenario.

4.2 Results

Before the results are presented, all the inputs to the MATLAB software are discussed. Recalling from Chapter 2, difficulties may arise when a large Petri net is analyzed. Since the model includes twenty four places and nineteen transitions, the state space equation, found in Eq. 2.19, is used for analysis and simulation. The first element required is the incident matrices. The PN toolbox was used to generate the matrices for the discrete Petri net model. Then, a few changes were done to transition the incident matrices for HPN analysis. The input and output incident matrices for the final HPN are,

$$\mathcal{B}^{-} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

$$\mathcal{B}^{+} = \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&3&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

Using Eq. 2.14, the incident matrix is,

$$\mathcal{B} = \begin{pmatrix}
-1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&-1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&-1&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&1\\
0&0&0&1&-1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&-1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&-1&0&0&0&0&0&0&0&0&0&0&-1\\
0&0&0&0&1&0&0&0&0&0&-1&0&0&0&0&0&0&0&-1\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&-1\\
0&0&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&-1&0\\
0&0&0&0&0&0&0&0&-1&0&0&-1&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&1&0&0&0&-1&0&0&-1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&3&-1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&-2&0&0\\
0&0&0&0&0&0&0&-1&0&-1&-1&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&-1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&-1&0&0&0&0&0\\
0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&-1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&-1&0&0&0&0&0&0&0&0\\
0&0&1&-1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

4.2.1 Case 1

The first case depicts a situation wherein the LKSS successfully activates but there is no unintentional lane departure detected. The turn signals are off and the cameras and sensors are assumed to be working properly. So, there is one token in p_1, p_{sensor} and p_{turn}. The initial marking is,

$$M_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}^T$$

Since this case only affects the discrete components of the HPN model for the LKSS, the toolbox was used to generate the coverability tree of the system. The same marking served as input to the MATLAB program for finding the reachable states of the system. Figs. 4.3 and 4.4 show the results obtained.

```
Reachable places:
     1     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     1     0
     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     1     0     0
     0     0     0     0     0     1     1     0     0
     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     1     1
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     1     1     1     1     0     0     0     0     0
     1     1     1     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     1

 firing sequence 1
t01 -->t02 -->t03 -->t04 -->t05 -->t07 -->t019 -->t03 -->
```

Fig. 4.3.: Reachable places from MATLAB for Case 1

Coverability Tree - Text Mode

From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To
M0	t1	M1	**M1**	**t2**	**M2**	M2	t3	M3	**M3**	**t4**	**M4**	M4	t5	M5	**M5**	**t7**	**M6**
M6	tsafe	M7	**M7**	**t3**	**M8**												

M[p1,p2,p3,p4,p5,p6,p7,p8,pwref,p10,p11,p12,p13,p14,p15,p16,pright,pleft,pldw,plka,psensor,pblinker,pvref,paref,pdata]

M0 = [1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,0]	M1 = [0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,0]	M2 = [0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,0]
M3 = [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,1]	M4 = [0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0]	M5 = [0,0,0,0,1,1,1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0]
M6 = [0,0,0,0,0,1,1,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0]	M7 = [0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0]	M8 = [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,1]

Fig. 4.4.: Coverability Tree from PN Toolbox for Case 1

Referring to Fig. 4.3, the program identifies a single firing sequence and the reachable states are saved in a matrix. The first column represents the initial marking, followed by the next marking on the second column and so on. When transition, t_1, fires, the cameras and sensors are activated. This is indicated by the second column having a token on the second row. When the enabled transition, t_2, fires to begin data gathering the next marking in the third column occurs and the process continues for the rest of the matrix. We can confirm that we observe the expected behavior by looking at the last two columns. When transition t_{019}, which is t_{safe}, fires, a token is sent to the the 3rd row, p_3 that data gathering has begun again. Also, a token in p_{15} tells us that the vehicle is safe and LKSS has activated. The last transition that fires shows that the system has enough data gathered to recheck the current status of the vehicle again. Finally, both tools have given eight reachable markings. Thus, verifying that the expected behavior of the model for the first scenario has occurred.

4.2.2 Case 2

The second case is a situation wherein the LKSS is not able to activate due to either of the two reasons. The first being that the a turn signal is on and the cameras and sensors are assumed to be working properly. So, there is one token in p_1 and p_{sensor}. The initial marking is,

$$M_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

Once more, only the discrete components of the HPN model are affected. The toolbox and the program were used to generate the coverability tree. From Fig. 4.5, the program generates a single firing sequence. When the last transition t_5 fires, a token in p_5, p_6 and p_7 that indicates initialization stage has begun to check the three parameters - turn signal, speed and steering angle. Although the speed and steering angle places, 22^{nd} and 23^{rd} row in the last column of the matrix respectively, are in a safe range, the turn signal is on. Hence, the marking temporarily ends there and the

```
Reachable places:
     1     0     0     0     0     0
     0     1     0     0     0     0
     0     0     1     0     0     0
     0     0     0     0     1     0
     0     0     0     0     0     1
     0     0     0     0     0     1
     0     0     0     0     0     1
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     1     1     1     1     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     1     0     0


 firing sequence 1
t01 -->t02 -->t03 -->t04 -->t05 -->
```

Fig. 4.5.: Reachable places from MATLAB for Case 2 - Turn signal on

system waits until the turn signals are off. Comparing Figs. 4.5 and 4.7, both tools have generated the same reachable places. Therefore, the first reason for the second case has been verified.

The other reason for the LKSS not activating is if a camera and/or sensor is malfunctioning. It is indicated by the place p_{sensor} having no tokens. The initial marking is,

$$M_o = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}^T$$

The program produces a single firing sequence. Looking at Fig. 4.6, the flow is similar to Fig. 4.5 except that there are only three reachable places excluding the initial marking. The last enabled transition has fired and sent a token to begin data gathering. However, the system has detected a sensor or camera malfunction. This may result to unreliable data and false alarms. Hence, the system halts operation until the sensor is fixed. Both tools have generated the same and expected reachable places which confirms the behavior for this case.

```
Reachable places:
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     1     1     1     1
     0     0     0     0
     0     0     0     0
     0     0     0     1


 firing sequence 1
t01 -->t02 -->t03 -->
```

Fig. 4.6.: Reachable places from MATLAB for Case 2 - Sensor Malfunction

Coverability Tree - Text Mode

From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To
M0	t1	M1	**M1**	**t2**	**M2**	M2	t3	M3	**M3**	**t4**	**M4**	M4	t5	**M5**			

M[p1,p2,p3,p4,p5,p6,p7,p8,pwref,p10,p11,p12,p13,c,p14,c,p16,pright,pleft,pldw,plka,psensor,pblinker,pvref,paref,pdata]

M0 = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]	M1 = [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]	M2 = [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]
M3 = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1]	M4 = [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]	M5 = [0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

Fig. 4.7.: Coverability Tree from PN Toolbox for Case 2 - Turn signal on

Coverability Tree - Text Mode

From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To
M0	t1	M1	**M1**	**t2**	**M2**	M2	t3	**M3**												

M[p1,p2,p3,p4,p5,p6,p7,p8,pwref,p10,p11,p12,p13,p14,p15,p16,pright,pleft,pldw,plka,psensor,pblinker,pvref,paref,pdata]

M0 = [1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0]	M1 = [0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0]	M2 = [0,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0]
M3 = [0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,1]		

Fig. 4.8.: Coverability Tree from PN Toolbox for Case 2 - Sensor Malfunction

4.2.3 Case 3

The third case branches out from the first. Once the LKSS activates, one of the key parameters is detected to be out of the safe range. The turn signals are off and the cameras and sensors are assumed to be working properly. Any of the key parameters, lane position, speed or steering angle can be used to simulate an unsafe condition for the vehicle. Fortunately, the driver has steered the vehicle to the center of the lane before the lane departure warning is sent. For the results shown below, the angle is considered to be out of range. So, there is one token in p_1 and p_{angle} while two tokens are set for p_{sensor} and p_{turn}. This ensures that the simulation is continuous and the assumptions for turn signal, cameras and sensors are maintained. The initial marking is,

$$M_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 1 & 0 \end{bmatrix}^T$$

Figs. 4.9 and 4.10 shows the a graphic version of the coverability tree and the reachable places generated the PN toolbox. The program has generated three firing sequences which indicates that two transitions are enabled are the same time and can lead to two different reachable places. This is found on Fig. 4.11. We focus on the last nodes at the bottom of the tree in Fig. 4.9. All possible final markings are generated after firing either transition t_3, which indicates data gathering or t_7, which means that data has been gathered and the initialization stage begins. Preceding these markings is firing of either the mentioned transitions or t_{16}, which indicates that the vehicle is considered unsafe and lane departure is being checked. This is the expected behavior of this model since this scenario involves the driver correcting an unwanted lane departure before the warning is sent. Hence, the resulting markings and the enabled transitions that can fire provided a loop for the system. The loop intends to continue and repeat data gathering until an unwanted lane departure has been detected. This verifies the expected behavior of the model for this scenario.

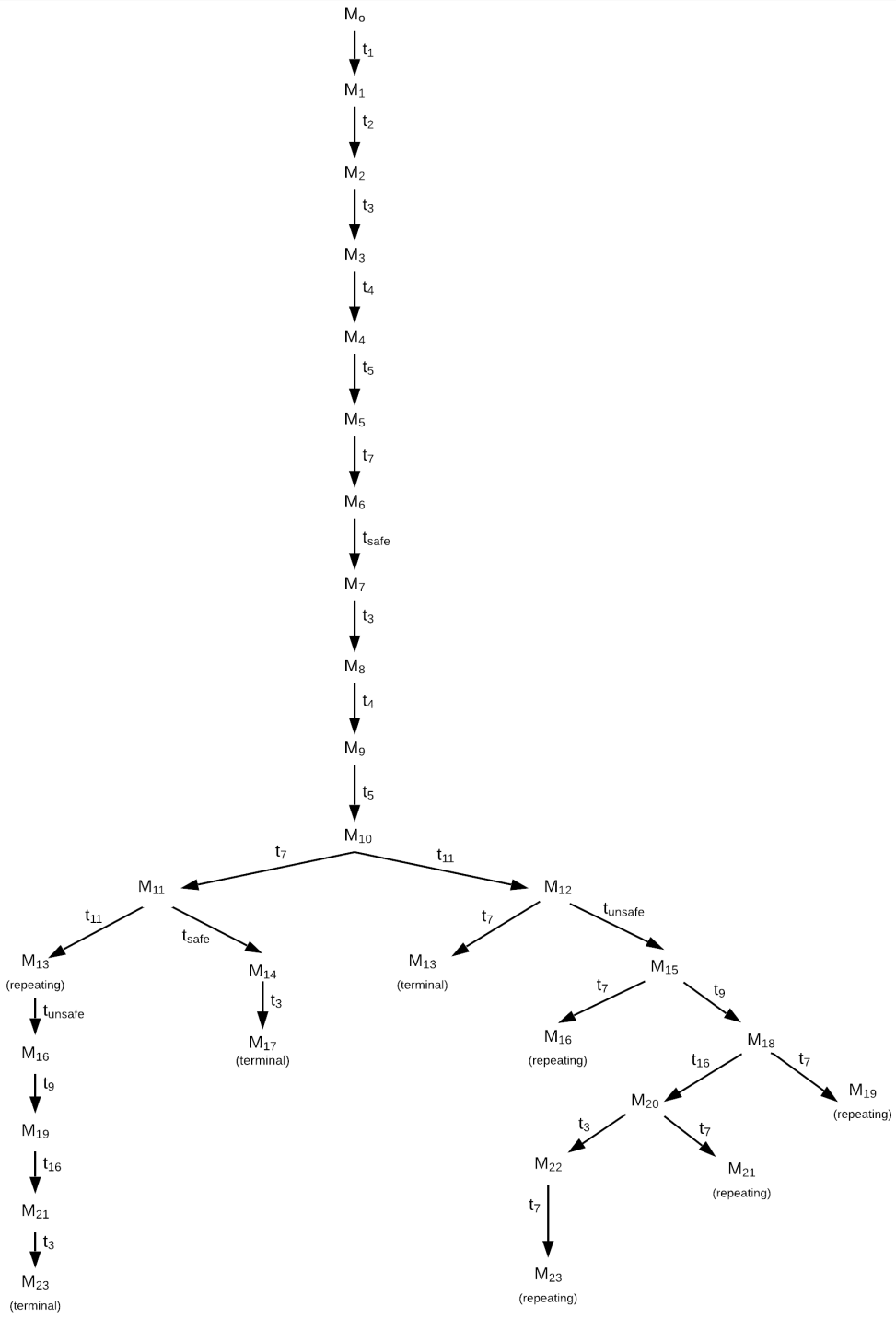

Fig. 4.9.: Coverability Tree from PN Toolbox for Case 3

Coverability Tree - Text Mode

From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To	From	Fired	To
M0	t1	M1	**M1**	**t2**	**M2**	M2	t3	M3	**M3**	**t4**	**M4**	M4	t5	M5	**M5**	**t7**	**M6**
M6	tsafe	M7	**M7**	**t3**	**M8**	M8	t4	M9	**M9**	**t5**	**M10**	M10	t7	M11	**M10**	**t11**	**M12**
M11	t11	M13	**M11**	**tsafe**	**M14**	M12	t7	M13	**M12**	**tunsafe**	**M15**	M13	tunsafe	M16	**M14**	**t3**	**M17**
M15	t7	M16	**M15**	**t9**	**M18**	M16	t9	M19	**M18**	**t7**	**M19**	M18	t16	M20	**M19**	**t16**	**M21**
M20	t3	M22	**M20**	**t7**	**M21**	M21	t3	**M23**	**M22**	**t7**	**M23**						

M[p1,p2,p3,p4,p5,p6,p7,p8,pwref,p10,p11,p12,p13,c,p14,c,p16,pright,pleft,pldw,plka,psensor,pblinker,pvref,paref,pdata]

M0 = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,1,0]	M1 = [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,1,0]	M2 = [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,1,0]
M3 = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,1,1]	M4 = [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,0,1,0]	M5 = [0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,0,1,0]
M6 = [0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,1,0]	M7 = [0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,1,0]	M8 = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,1,1]
M9 = [0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0]	M10 = [0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0]	M11 = [0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0]
M12 = [0,0,0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0]	M13 = [0,0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]	M14 = [0,0,1,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1,0]
M15 = [0,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0]	M16 = [0,0,0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]	M17 = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1,1]
M18 = [0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0]	M19 = [0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]	M20 = [0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]
M21 = [0,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]	M22 = [0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1]	M23 = [0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]

Fig. 4.10.: Reachable places from PN Toolbox for Case 3

```
Reachable places:
     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     1     0     0     0     0     0     0     0     0     0     1     0
     0     0     0     0     1     0     0     0     0     1     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     0     0     0     1     0     1     0     1     0     0     0     0
     0     0     0     0     0     1     1     0     0     0     1     1     1     1     1     1     1     1     1
     0     0     0     0     0     1     1     0     0     0     1     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     1     0     0     0     0     1     0     1     0     1     1     1     1
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     1     1     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     1     1     1     1     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     2     2     2     2     1     1     1     1     1     0     0     0     0     0     0     0     0     0     0
     2     2     2     2     2     2     1     1     1     1     1     0     1     0     1     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     1     1     1     1     1     1     1     1     1     1     1     1     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     1     0     0     0     0     0     0     0     0     0     1


 firing sequence 1
t01 -->t02 -->t03 -->t04 -->t05 -->t07 -->t019 -->t03 -->t04 -->t05 -->t07 -->t07 -->t09 -->t016 -->t03 -->

 firing sequence 2
t011 -->t018 -->

 firing sequence 3
t018 -->
```

Fig. 4.11.: Reachability Tree from MATLAB for Case 3

4.2.4 Case 4

The last test scenario for this LKSS model is when the system activates and there is unintentional lane departure detected. The driver is not able to correct it in time, so this case allows the warning to be sent and the system takes over after a certain period of time. This scenario makes use of the entire HPN model. Firstly, the final discrete version of the model from Fig. 3.4 was simulated using the PN toolbox. For variation, the lateral position of the vehicle is out of the safe range and the lane departure is detected from the left side of the vehicle. Similarly, the assumptions for turn signal, cameras and sensors are maintained from the third case. This means that there can be as many of tokens in p_{sensor} and p_{turn} to support the assumption. There is one token in p_1 and p_{13}, two tokens in p_{left} and three for p_{lane}. The initial marking is,

$$M_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 3 & 3 & 0 & 0 & 0 \end{bmatrix}^T$$

Although this simulation was successful, the toolbox generated half a thousand reachable places. It is a huge feat to create the coverability tree from the tool. Hence, the program was used to generate the 95 reachable places for the HPN model for the second testing. The initial marking for this case is,

$$M_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 3 & 3 & 0 & 0 & 0 \end{bmatrix}^T$$

All the arc weights were considered to be integers to reduce the reachable places, found in Figs. 4.12 to 4.15. The program has generated a total of ten firing sequences because multiple transitions are enabled at a certain time during the simulation. These results are easier to follow due to the reduced markings. This scenario is connected with the third case which has proven the model to behave as expected. With this in mind, we can direct our analysis towards the last reachable places found on Fig. 4.15. The last columns of the matrix are the possible final markings produced by the simulation. Looking at the nineteenth row, which indicates the marking evolution of p_{19} or p_{lka}, there are three columns with a token in this place. When the time elapsed, indicated by p_{14} or the fourteenth row, has received two tokens, the succeeding marking is denoted by the token in p_{lka}. This shows that the system takes over when the driver was not able to steer the vehicle back to the center of the lane after the warning has been sent. However, the number of markings is still large to create a reachability tree. It is a demanding task that requires a grueling and lengthy period of analysis. Also, the interaction of the discrete and continuous parts are not verified in this simulation because the arc weights are similar to a petri net that is only discrete. The solution is to test the model using SimHPN. As mentioned in the previous section, SimHPN has a simple GUI that requires inputs similar to the MATLAB program. A *.mat* file was created for efficient loading of the HPN model.

The GUI automatically displays the file read as seen on the bottom area of Fig. 4.16. From the same figure, the file describing the HPN includes the input and output incident matrices, the initial marking, the firing rate (*lambda*) and the types of transition. The firing rates vary per transition, depending upon the priority of firing. A firing rate of 1 depicts that the transition will fire in a second. If the rate is

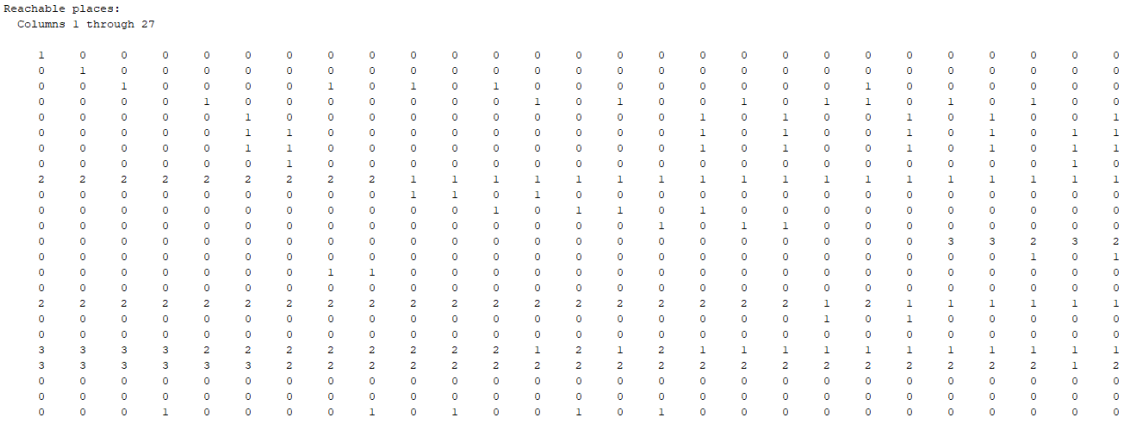

Fig. 4.12.: First set of reachable places from MATLAB for Case 4

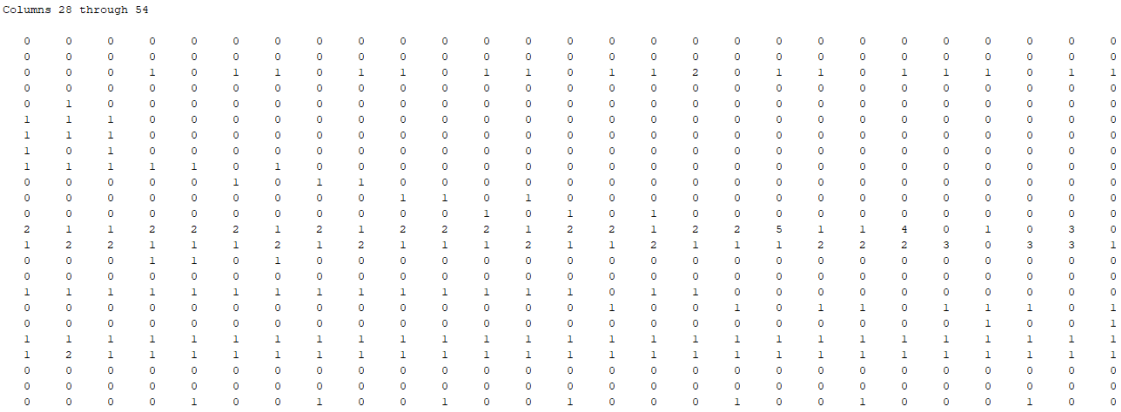

Fig. 4.13.: Second set of reachable places from MATLAB for Case 4

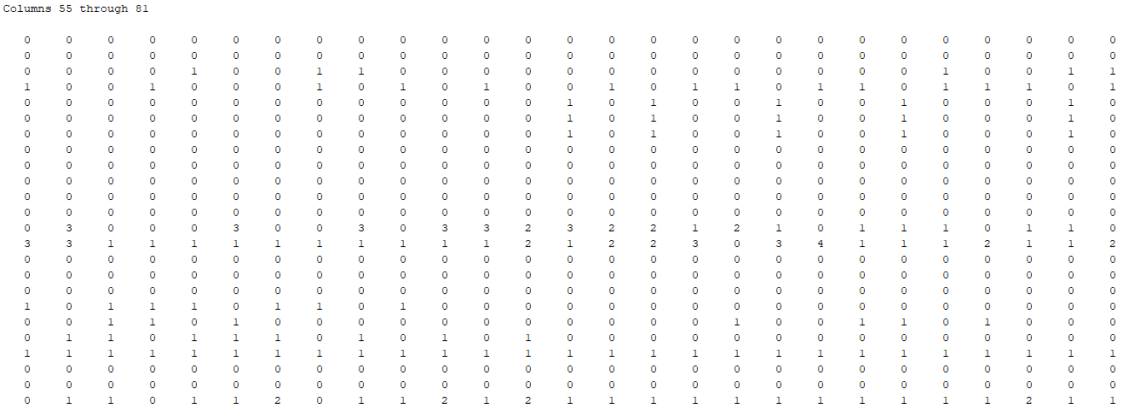

Fig. 4.14.: Third set of reachable places from MATLAB for Case 4

```
Columns 82 through 95

     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     1     0     0     1     0     2     1     0
     0     1     0     1     0     0     0     0     0     0     0     0     0     0
     1     0     1     0     0     1     1     0     1     0     0     0     0     0
     1     0     1     0     1     1     1     1     1     1     1     0     0     0
     1     0     1     0     1     1     1     1     1     1     1     0     0     0
     0     0     0     0     1     0     0     1     0     1     1     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     1     0     0     0     0     0     0     0     0     0     0     0     0     0
     1     2     2     0     2     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     1     1     1
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     0     1     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     1     1     1     1     0     1     1     0     1     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     2     2     2     2     2     2     2     2     3     2     3     2     3     4
```

Fig. 4.15.: Last set of reachable places from MATLAB for Case 4

less than 1, e.g. 0.2, then the transition will fire in a fifth of a second. The smaller the rate, the higher the priority of firing. There are 18 deterministic discrete transitions, labeled as q, and one continuous transition, c. By clicking the button *Simulate*, we see the marking evolution for all the places of the model. We can refine the results by choosing to display the places of importance, p^c_{13}, p^c_{14}, p^d_{15}, p^d_{18} and p^d_{19}. The first two continuous places indicate the time set and time elapsed while the remaining discrete places are the vehicle safe status, lane departure warning and lane keeping assist respectively. Fig. 4.17 shows the simulation result containing the marking evolution of the major events. When the p^d_{18} receives one token, the lane departure warning is sent. The place p^c_{13} begins with 3 tokens and counts down quickly by sending tokens to p^c_{14}. As soon as two seconds has elapsed, lane keeping assist takes over. This action

is reflected through the token in p_{19}^{d} or p_{lka}. After some time, the vehicle is safe once again. Hence, the result verifies that when a similar scenario happens, the model operates with the expected behavior.

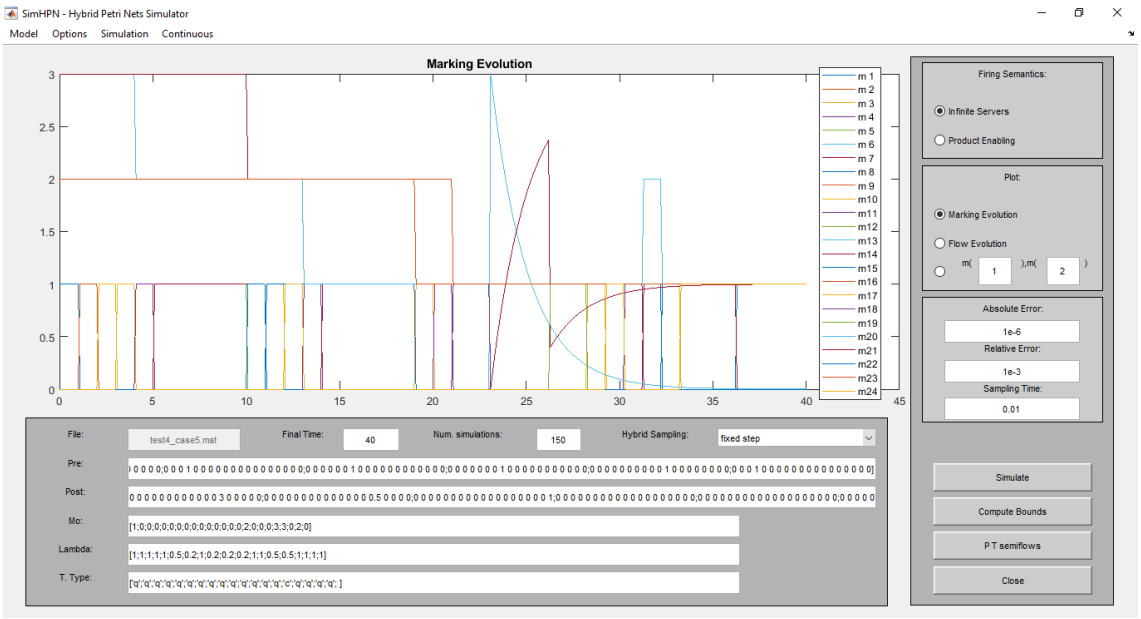

Fig. 4.16.: SimHPN GUI set-up for Case 4

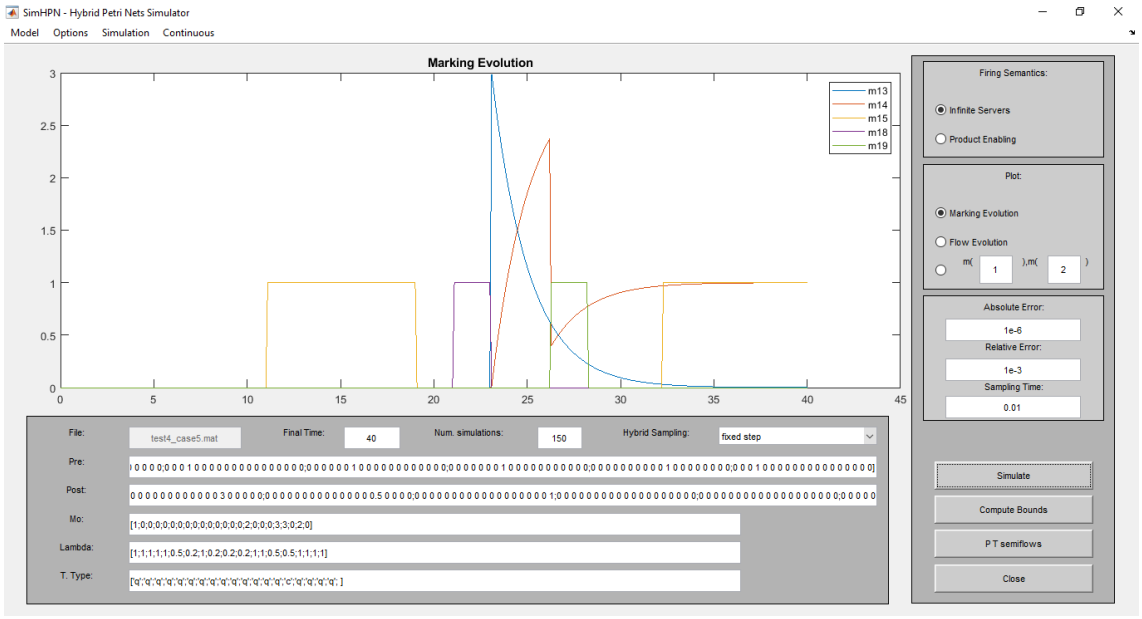

Fig. 4.17.: Simulation result of major events for Case 4

5. CONCLUSION

5.1 Summary

This thesis is an effort to combine lane departure warning and lane keeping assist functionalities into one system using a Hybrid Petri net. Also, it is an attempt to simulate the behaviour of the system based on different scenarios. Chapter 1 introduced the concept of Petri nets and how they are used for graphical representation of dynamic systems. In addition, related work to the modeling approach such as road traffic, adaptive cruise control and automated parallel parking. The same chapter described the individual operation of the two systems of interest.

Chapter 2 provided the background on discrete, continuous and hybrid Petri nets. Examples were presented for all the types mentioned to simplify theoretical notations and dynamics. The detailed process of modeling was discussed in Chapter 3. Initially, a flowchart was created to identify the high-level events in the system. Then, the events were translated into places and transition that made up the initial version of the discrete Petri net model of the system. Several versions were created after identifying drawbacks emerged one after the other. Once the drawbacks were resolved, a final version discrete petri net was integrated with a simple continuous petri net to form the hybrid petri net model. Four scenarios were made and the expected behaviour was discussed in detail.

The final Petri net models were tested using MATLAB in Chapter 4. Firstly, the final discrete Petri net model was tested by created an algorithm on MATLAB to determine the reachable states of the system for all the cases. Then, an embedded tool, called PN Toolbox, was used to generate the coverability tree to support and verify the results from the algorithm. The chapter concludes with simulating hybrid Petri net model using SimHPN, another embedded tool, and justified the expected

behaviour. Overall, this thesis has aimed to demonstrate the practical applications of the modeling approach. In particular, hybrid Petri nets are a very useful tool for modeling and analysis for the study of complex dynamic systems, specifically road and vehicle safety systems. Further control algorithms can be added to improve such systems to reduce road accidents and promote safer driving experience. Potentially, this type of approach can be used to assess most control systems. Troubleshooting and debugging algorithms and simulation can be made simpler through this graphical representation.

5.2 Future Work

This thesis has opened several areas of future work to further improve the system's functionality. At the beginning, the assumptions before modeling included a straight and even road with a straight-line driving condition. New variables for uneven and curved roads can be incorporated for real-life driving. With the fast-paced progress on technology, the concept of vehicle-to-vehicle communication can be adapted to the model. This will help replace redundant functionalities and reduce the size of the model. Another direction to expand the work is in relation to the current sensors used. Sensor measurements such as steering angle, speed and lateral position can be converted to continuous places and transitions to improve the decision-making process of the system. This will introduce more detailed scenarios to improve the simulations.

In relation to improving the model itself, a control strategy can be implemented. This aims to resolve conflicts between multiple enabled transitions. Also, it may be a new Petri net controller or adding decision transitions in the existing model. Lastly, the model can be converted to timed hybrid Petri nets or probabilistic Petri nets. These types would account for timed firing of transitions and the probabilities of events. Not only will these recommendations improve the functionality, but also they are geared towards system optimization.

REFERENCES

REFERENCES

[1] H. Takahashi and K. Kuroda, "Intelligent vehicle control considering driver's visual perception," *Proceedings 199 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems*, 1999.

[2] W. Reports, "Advanced driver assistance systems market 2018 global analysis, opportunities and forecast to 2023," (Last Date Accessed: 4 February 2019). [Online]. Available: https://bit.ly/2IotGEA

[3] Z.-L. L. Li-Gui Zhang and Y.-Z. Chen, "Hybrid petri net modeling of traffic flow and signal control," *2008 International Conference on Machine Learning and Cybernetics*, vol. 4, pp. 2304–2308, 2008.

[4] O. Yaqub, "Modeling, analysis, and simulation of two connected intersections using discrete and hybrid petri nets," Master's thesis, Purdue University, Indianapolis IN, 2012.

[5] S. Yazdanpanah and R. Kourdy, "Presenting a model to car collision avoidance using fluid stochastic petri net," *2009 WRI Global Congress on Intelligent Systems*, vol. 2, pp. 283–287, 2009.

[6] K. Ramesh, "Modeling and simulation of an automated parallel parking system using hybrid petri nets," Master's thesis, Purdue University, Indianapolis IN, 2015.

[7] S.-S. H. W.-C. K. C.-C. H. Pei-Yung Hsiao, Kuo-Chen Hung and Y.-M. Yu, "An embedded lane departure warning system," *2011 IEEE 15th International Symposium on Consumer Electronics (ISCE)*, pp. 162–165, 2011.

[8] E. Salari and D. Ouyang, "Camera-based forward collision and lane departure warning systems using svm," *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1278–1281, 2013.

[9] M. G. D. Cualain and E. Jones, "Multiple-camera lane departure warning system for the automotive environment," *IET Intelligent Transport Systems*, vol. 6, pp. 223–234, 2012.

[10] L. L. Yue Dong, Jintao Xiong and J. Yang, "Robust lane detection and tracking for lane departure warning," *International Conference on Computational Problem-Solving (ICCP)*, pp. 461–464, 2012.

[11] J. G. Jia He, Hui Rong and W. Huang, "A lane detection method for lane departure warning system," *International Conference on Optoelectronics and Image Processing*, pp. 28–31, 2010.

[12] V. Gaikwad and S. Lokhande, "An improved lane departure method for advanced driver assistance system," *International Conference on Computing, Communication and Applications*, pp. 1–5, 2012.

[13] M. W. Xiangjing An and H. He, "A novel approach to provide lane departure warning using only one forward-looking camera," *International Symposium on Collaborative Technologies and Systems*, pp. 356–362, 2006.

[14] T. J. Mei Chen and D. Pomerleau, "Aurora: a vision-based roadway departure warning system," *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, pp. 243–248, 1995.

[15] H. Takahashi, "Various perspectives for driver support systems in japan," *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems*, pp. 1627–1632, 2012.

[16] G. H. Daniel Hoehener and D. D. Vecchio, "Design of a lane departure driver-assist system under safety specifications," *IEEE 55th Conference on Decision and Control (CDC)*, pp. 2468–2474, 2016.

[17] G. L. Abdelhamid Mammeri and A. Boukerche, "Design of lane keeping assist system for autonomous vehicles," *7th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, 2015.

[18] J.-H. W. Jing-Fu Liu and Y.-F. Su, "Development of an interactive lane keeping control system for vehicle," *IEEE Vehicle Power and Propulsion Conference*, pp. 702–706, 2007.

[19] R. David and H. Alla, *Discrete, Continuous and Hybrid Petri Nets*. New York: Springer, 2005.

[20] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. New York: Springer, 2004.

[21] L. R. Jorge Julvez, Manuel Silva and C. Mahulea, "On control of continuous petri nets," (Last Date Accessed: 22 February 2019). [Online]. Available: http://www.diee.unica.it/ seatzu/slides Mahulea.pdf

[22] L. Ghomri and H. Alla, "Modeling and analysis using hybrid petri nets," *Nonlinear Analysis: Hybrid Systems*, pp. 141–153, 2007.

[23] R. David and H. Alla, "On hybrid petri nets," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 11, pp. 9–40, 2001.

[24] N. H. T. S. Administration, "Driver assistance technologies," (Last Date Accessed: 4 February 2019). [Online]. Available: https://www.nhtsa.gov/equipment/driver-assistance-technologies

[25] M.-H. M. Cristian Mahulea and O. Pastravanu, "Learning about petri net toolbox," 2016, (Last Date Accessed: 20 February 2019). [Online]. Available: http://www.pntool.ac.tuiasi.ro/help/index.html

[26] J. Júlvez, C. Mahulea, and C.-R. Vázquez, "Simhpn: A matlab toolbox for simulation, analysis and design with hybrid petri nets," *Nonlinear Analysis: Hybrid Systems*, vol. 6, no. 2, pp. 806 – 817, 2012, (Last Date Accessed: 30 January 2019).