6-26-2015

Predictive Warning System for a Class of Shared-Control Vehicular CPS via Dynamic Information Flow Tracking.

Rafael Figueroa

Rafael Figueroa

Electrical and Computer Engineering

This thesis is approved, and it is acceptable in quality and form for publication: *Approved by the Thesis Committee:*

Dr. Rafael Fierro

Chair

Dr. Asal Naseri

Member

Dr. Thomas Caudell

Member

Predictive Warning System for a Class of Shared-Control Vehicular CPS via Dynamic Information Flow Tracking

by

Rafael Figueroa

B.S., Mechanical Engineer, University of Carabobo, 2003

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Electrical Engineering

The University of New Mexico

Albuquerque, New Mexico

May, 2105

Dedication

Mare and Adriana.

Acknowledgments

I would like to thank:
Dr. Rafael Fierro, my academic advisor, for his support and guidance.
Dr. Crandall, for introducing me to the subject of DIFT.
Dr. Thomas Caudell and Dr. Asal Naseri for their advice and for being part of my committee.
Dr. Meeko Oishi, for her important input on the first version of the warning system dynamic model.
Paul Groves and Coralis Nunes, my study partners.
MARHES Lab partners.

This work was supported by the NSF grant CNS 1017602

Predictive Warning System for a Class of Shared-Control Vehicular CPS via Dynamic Information Flow Tracking

by

Rafael Figueroa

B.S., Mechanical Engineer, University of Carabobo, 2003

M.S., Electrical Engineering, University of New Mexico, 2015

Abstract

Manned vehicles are maturing into robotics agents under shared control. Vehicle operators will be confronted with understanding the expected response from the robotic agent to their actions. I propose a warning system framework to continuously track human inputs, identify possible conflicts and provide contextual warning information to the operator to help avoid accidents. I consider a robotic agent which nominal operation can be encoded using the hybrid automata framework with human inputs. Trajectory prediction methods are used to identify possible conflicts, coded as the avoid set of the system. Using Dynamic Information Flow Tracking (DIFT), human inputs and their effect over time are accumulated and classified in a continuous range between spurious or legitimate inputs.

Contents

List of Figures

List of Tables

Glossary

$term$	explantion
$\mathbf{H}$	hybrid automata
$\mathbf{Q}$	hybrid automata mode
$\mathbf{F}$	dynamics function set
$\mathbf{G}$	guard set
$\mathbf{R}$	reset map set
$\boldsymbol{x}$	system state
$\boldsymbol{u}$	human agent input
$\boldsymbol{u_n}$	nominal human agent input
i	state index
j	input index
e	hybrid automata mode index
k	discrete time index
f	nominal input distribution

f_n	nominal input distribution
λ	normalized *legitimate* tag function
ς	normalized *spurious* tag function
$\mathbf{A}$	avoid set
v	warning level activation potential
r	self-feedback gain
w	constant *spurious* tag gain
γ	conflict level
m	constant conflict level gain
$\varphi(\cdot)$	warning level activation function
t_a	maneuverability time
t_c	time to conflict
t_f	trajectory tracking time horizon
y	warning level
t_0	simulation start
t_{ss}	*spurious* inputs start
t_{se}	*spurious* inputs end
t_{cs}	future conflict present start
t_{ce}	future conflict present end

Chapter 1

Introduction

1.1 Motivation

A significant challenge on the design of warning systems is attempting to identify the intentions of the human agent and how this could affect a robotic agent under shared control. The warning system resides with the robotic agent or vehicle and has very limited data regarding what the human agent is experiencing. This leads to creating warning signals specific to the expected accidents or conflicts, *e.g.* a dangerous bank angle on a commercial airplane. However, as vehicles become more complex, increasing the number of alarms for specific situations becomes overwhelming for all but specialized human agents.

To move beyond specific alarms, a warning system would need a basic grasp of the human agent actions within the system operational plans and current conditions. For certain classes of shared-control vehicles this information is partially available and can be incorporated into the warning system. With this information, a warning system would be able to provide a contextual warning to human agents and possibly prevent accidents.

1.2 Overview

In this work, DIFT (Dynamic Information Flow Tracking)[1] has been adapted to track the actions of the human agent. A key feature of DIFT is to be able to classify information, in our case continuous or discrete inputs, into spurious or legitimate. I consider a robotic agent which task or behavior can be modeled as a hybrid automaton, where data is available regarding the nominal actions for each continuous evolution mode. Using this model, the human agent inputs will be compared with the nominal inputs to classify the current actions with a probability of being either spurious or legitimate.

There are two major event categories for which alarms are raised:

- There is a current or expected conflict.
- Human actions are potentially errors or intentional attacks (spurious inputs).

However, these events are not mutually exclusive. The warning system can be tuned to be more sensitive to inputs that are tagged as spurious. Under this methodology, a warning alarm would be raised much faster for the scenario with a combination of spurious actions and a predicted conflict. Although this would be the worst case scenario from the warning level point of view, it would be the scenario where the alarm would be most effective at avoiding accidents.

A common technique for conflict detection is to project the current or expected actions into the future and detect possible conflicts. Trajectory prediction methods alone could result in many false warnings in a system that allows for maneuvers from the human agent, *e.g.* a quick aggressive action meant for a time much shorter than the trajectory prediction time horizon. I propose to attenuate this disadvantage by dynamically tracking the evolution over time of the relationship between actions and possible conflicts.

The proposed warning system framework will function as a dynamic system, accumulating warning levels associated with these the spuriousness and conflict detection categories and providing a global warning level. This key characteristic will help to differentiate between a short term deviation from the nominal operation to a persistent erratic or reckless behavior. The system's alarms and context based presentation to the human agent can then be designed to be triggered by the warning level.

In addition to providing additional information regarding the human agent actions, the proposed method would allow for the identification of the input or set of inputs contributing to the trajectory in conflict. An advantage to tracking each input separately is that a warning level can be assigned to each one. This allows for customization of alarm levels at different conditions, a key feature needed for large or complex systems.

By encoding the nominal behavior of the robotic agent as hybrid automaton, the warning system designer will be able to use existing techniques from the hybrid automata framework to analyze the system. Conflict conditions will also be encoded into the hybrid automata framework as the avoid set of the system. These powerful analysis techniques help to design warning systems adapted to the robotic agent dynamics.

Finally, the system implementation is done using ROS or Robot Operating System in order to allow for direct real-world usage. A modular design approach is used. The objective is to create a framework to be adapted for particular robots or vehicles. A case study is presented with a differential drive robot and a simple task. The task and possible conflicts are encoded in a hybrid automaton and a basic warning system model is proposed and simulated.

1.3 Related Work

The modeling of robotic agents dynamics is established as an application of hybrid automata theory, or more in general hybrid system, model [2]. In particular, the techniques developed to compute the reachability of the avoid set [3] would help answer questions such as: can we guarantee that an alarm will be raised one second before reaching the avoid set? (conflict condition). This provides a wealth of publications available to properly design a hybrid automaton model and analyze their expected system response.

In this work, the hybrid automaton model is used specifically to model the nominal operation of the robot. Related work in encoding the robotic agent behavior based control system in the hybrid automata framework [4] is available. Similar to the approach used in this work, each mode in a hybrid automaton represents a specific robot behavior.

The research of shared control vehicles is prominent in commercial air transportation systems. A relevant topic is the design of the user interface to provide information to the aircraft pilot.The work on the verification of such systems [5] was influential on this thesis; the authors implemented a procedural automaton to analyze the system, where each mode represents a step in the aircraft landing process. Warning systems are designed taking into consideration the human agent response. The objective is that the human would perform corrective actions in the event of a conflict, however the information might not be available or the vehicle interface might not be able to show it in the proper context.

In air transportation systems, the pilot intentions has been studied using probabilistic trajectory prediction [6]. In addition, many short-term trajectory prediction methods [7] have been studied. Due to the modular implementation used in this thesis, the trajectory prediction method can be replaced without affecting the rest

of the system.

DIFT [1] has been implemented in hardware [8] and software [9], as a mechanism to prevent malicious attacks in computer security. The key mechanism of tagging input information as either legitimate or spurious and tracking the evolution of these tags is present in these implementations. In particular, [10] proposes a decoupling of DIFT with a dedicated coprocessor. Their motivation is based on simplecity and economy; in this thesis, the decoupling of DIFT with the main process is advantageous as a basis for adaptation on existing vehicles. Ideally, the warning system and associated information flow tracking should be done in parallel with the process with minimum effect.

Although research on conflict prediction and avoidance general frameworks is abundant, the study of warning system frameworks tend to focus on more specific solutions. In [11], a warning system is proposed which uses evolving neural networks algorithms. Their approach included a conflict prediction method and a contextual graphical user interface providing sensor information regarding the conflict (a collision) and the current warning level.

Similar work on warning systems involving prediction methods includes a practical example in [12]. Their focus is on the trajectory prediction of both their vehicles and pedestrians to detect possible future collisions between them. The warning system implemented and shown in [13] includes two different alarm levels, a light and sound at first and a higher level of alarm that actually breaks the vehicle automatic. This is inline with the warning system design approach proposed in this thesis in regards to conflict detection, however it does not include the difference between spurious and legitimate inputs.

Chapter 2

Preliminaries

2.1 Hybrid Automata

The following hybrid automata framework [14] will be used to describe the nominal behavior of the robotic agent. The hybrid automaton **H** is a collection which includes the set of continuous states, the set of discrete states or modes **Q** and for each mode: the set of functions **F**, domain, edges, guard conditions **G** and reset map. For a mode index e, the dynamics of the nonlinear agent can be described as:

$$\boldsymbol{x}(k+1) = F_{[e]}(\boldsymbol{x}(k), \boldsymbol{u}(k)) \tag{2.1}$$

Where $\boldsymbol{x}$ is the state of the system and $\boldsymbol{u}(k)$ is the human agent input. Using a nominal input $\boldsymbol{u_n}(k)$, written in terms of the state and nominal robotic agent operation, results in the hybrid automata discrete-time dynamics model:

$$\boldsymbol{x}(k+1) = F_{n[e]}(\boldsymbol{x}(k)), \tag{2.2}$$

To refer to specific member of a set, *e.g.* $q_{[e]} \in \mathbf{Q}$, the following indexes will be used:

i is the state index,
j is the input index,
e is the hybrid automata mode index,
k is the discrete time index.

2.2 Dynamic Information Flow Tracking

Dynamic Information Flow Tracking or DIFT, was created originally as a hardware security mechanism to prevent attackers from taking control or damaging the program in a computing system.

DIFT works by dynamically tracking the sources of data coming into the memory. The sources are divided between legitimate sources and spurious sources. One cited possible spurious source, for example, would be data taken from the communication ports. When data from spurious sources is written into the memory a tag is assigned to that memory location. DIFT then tracks the use of this spurious data and assigns a dependency spurious tag to all the memory locations affected by it. The tracking is dynamic due that if legitimate data is stored in a memory location previously considered spurious, the tag of that memory location would change, however any spurious dependencies created would remain until changed.

Once a memory location has been tagged as spurious, the computer hardware system is configured to warn of specific uses of this data. For example, if the spurious data is used as a program pointer.

In particular, a warning is raised whenever a memory location tagged as spurious is used as a program instruction. Using a variety of attacks, such as a memory

overflow, an attacker could write a program to memory disguised as legitimate data. Once this program is ran by the hardware, without DIFT or a similar mechanism, it could take control of the computing system and never return control to the original program.

In this work, an analogy between a computer program and the hybrid automata framework is drawn. Computer programs are ran by reading an executing instructions from the program memory in order until a jump is found to another block of instructions. The orderly execution of a j block of instructions is made analogous to the continuous evolution of the hybrid automaton dynamic system for mode j. The decision or conditional jumps between blocks of instructions in a program are made analogous to the system of edges and guards in a hybrid automaton.

2.3 Artificial Neural Networks

Artificial neural networks, inspired from biological neural networks, can be used as a modeling system. Each computing node in the network is called a neuron. The edges between nodes are directional and weighted. In general, teaching or training a neural network refers to the adjustment of the network connection weights to achieve a desired model behavior.

In particular, a dynamically driven recurrent neural network is used in this work. Dynamically driven recurrent neural networks have feedback or recurrent connections. A State-Space model, similar to the models used in feedback systems in control theory, is available for this architecture. Artificial neural networks are good candidates to find a model based on available data. The network can be trained, using optimization techniques like gradient descent, to minimize the error between a desired response and the actual system response.

2.4 ROS: Robot Operating System

ROS or Robot Operating System allows for usage of advanced algorithms and tools related to robotics. It is written entirely as open-source software.

ROS is not a proper operating system, it must be installed on an existing operating system. Currently, the Linux Ubuntu distribution is the only one officially supported. Other experimental installations exist for OS X, UbuntuARM and others. ROS does not provide real-time control, however it can be interfaced with on board real-time control computers as in the case of the Atlas Robot [15]. Each task and element of the robot can be written as a different process and ROS takes care of their coordination and execution.

Each computational process in ROS is called a node. There is a special node called "Master" which coordinates the connection and execution between all other nodes. The nodes are connected using topics, which are named directed edges between the output of a node and the input of other nodes. The combination of nodes and topics form a computational artificial network.

The information on topics are shared using either ROS standardized messages or custom made messages. Common standardized robotic messages include the robot pose or sensor readings. A node is said to publish a topic when it makes it available for any other nodes to read. Nodes can read any published topic from other nodes by subscribing to those topics.

Libraries are available to use existing sensor and actuator drivers and create an API (Application Programming Interface) to fit their input/output mapping into a ROS formatting. In addition, many other open-source software have been interfaced with ROS. This allows a ROS developer to use advanced algorithms without fully understanding their inner mechanisms.

Chapter 3

Methodology

3.1 Warning System Design

3.1.1 Spurious Tag

The nominal operation of the vehicle under shared-control is modeled using the hybrid automata framework. Where every discrete event in a hybrid automaton represents a mode of operation, *e.g.* plane landing or cruising could be broken into 2 or more modes.

The human agent provides inputs to maneuver the vehicle. The likely distribution of the human nominal inputs is considered available data. The current inputs are compared with the nominal inputs: similar inputs are considered *legitimate* and dissimilar inputs are considered to be *spurious*.

For example, for a car cruising on the highway the steering wheel is expected to have small and slow deviations from the center, a sharp turn during this particular mode would signal possible conflicts.

The nominal input distribution for each input $u_{[j]}$, for each discrete event $q_{[e]}$, will be represented in general by a distribution function similar to a probability density function. The nominal input distributions could be then approximated based on previous inputs with adequate operation or known probability density functions such as Gaussian or normal distributions. The maximum(s) of the nominal input distribution would represent the nominal input(s):

$$u_{n[j,e]} = \arg\max_{u_{[j,e]}} f_{n[j,e]}(u_{[j,e]}), \tag{3.1}$$

where $f_{n[j,e]}$ is the nominal input distribution for input index j and hybrid automata mode index e. An input equal to the nominal input is assigned the maximum likelihood of being a *legitimate* input. In order to compare the *legitimate* or *spurious* qualities of various inputs, the function is normalized so that the value "1" always represents *legitimate* and the value "0" represents *spurious*. I define the normalized *legitimate* tag function $\lambda_{[j,e]}$ at time k as:

$$\lambda_{[j,e]}(k) = \frac{f_{n[j,e]}(u_{c[j]}(k))}{f_{n[j,e]}(u_{n[j,e]})}, \tag{3.2}$$

which maps each current input $u_{c[j]}$ to a *legitimate* tag value. The normalized *spurious* tag function $\varsigma_{[j,e]}$ is then defined as the complement of the normalized *legitimate* tag:

$$\varsigma_{[j,e]}(k) = 1 - \lambda_{[j,e]}(k). \tag{3.3}$$

3.1.2 Mode Tracker

Using the hybrid automata framework, the current mode index e is continuously tracked. Starting at mode $q_{[e]}$ as defined in the hybrid automaton *Init* set, the guards conditions are verified. When a guard condition is activated, the mode changes according to the particular model of the vehicle. The nominal input distributions, and thus the *legitimate* and *spurious* tag functions, will change accordingly to the new index e. Which means that inputs that were considered *legitimate* in the previous mode, might become *spurious* in the new mode and *vice versa*. For example, a plane that continues to ascend after reaching the cruising altitude and is expected to change to cruising mode might indicate a conflict.

3.1.3 Trajectory Prediction

Due that actions are taken by a human agent with information not available to the system, a prediction of the real input pattern is not considered. A short-term trajectory prediction model is implemented holding the current inputs as constants and observing the evolution of the possible conflicts at each time step. The prediction will be done for a fixed time horizon t_p which is chosen according to the dynamics of the robot agent and expected tasks. A balance should be achieved when designing this parameter, a larger time horizon would allow for more time for corrective actions but would also increase the prediction error when the actions are not held constant.

With the current inputs $\boldsymbol{u_c}(k)$ held constant, the dynamic model is simulated in time for the fixed time horizon t_p. During the simulation it is verified that the robot agent does not enter any conflict condition. The conflict conditions will be in terms of the avoid set $\boldsymbol{x}(k) \in \mathbf{A}$ [3] for this system. If the system reaches the avoid set, the estimated time for conflict t_c is calculated.

3.1.4 Warning Level

The warning system is modeled as a dynamic system. An output $y_{[j]}(k)$ is defined for each input $u_{[j]}(k)$. The output of the warning system represents the warning level associated with that input and is normalized such that $y \in [0, 1]$. Using this heuristic, alarms can be designed to be raised at different levels per input *e.g.* a low level alarm when the warning level reaches 0.1 and a high level alarm when it reaches 0.9.

The warning level is calculated based on three factors: previous warning level, the *spurious* tag of the current input and how imminent is a conflict in the finite horizon trajectory prediction. The presence of a conflict heavily influences the system to rise the warning level for all inputs, while the *spurious* tag of the current input only raises the level associated with that input. With this information, a user interfacing alarm system would be able to provide contextual information to the human agent. Not covered in this work, a separate conflict response system could also be designed to actuate the robotic agent to avoid the conflict if necessary.

The dynamic system model used for the warning level coincides with the model for a single artificial neuron with negative self-feedback [17]. This is a useful model due that accounts for the accumulation of the inputs history, given a more comprehensive view on the human agent intentions. In addition, provides for a mechanism to reduce the warning level as the robot agent is restored to safer conditions, *i.e.* the input *spurious* tag has been reduced and/or the trajectory prediction conflict has been resolved. The following are the equations describing the dynamic system:

$$v_{[j]}(k) = -\varphi_{[j]}(v_{[j]}(k-1))r_{[j]} + \varsigma_{[j]}(k)w_{[j]} + \gamma_{[j]}(k)m_{[j]}, \tag{3.4}$$

$$y[j] = \varphi_{[j]}(v_{[j]}(k)), \tag{3.5}$$

where:

$v_{[j]}(k)$	$\in \mathbb{R}$	is the warning level activation potential,
$r_{[j]}$	$\in \mathbb{R}$	is the negative self-feedback gain,
$\varsigma_{[j,e]}(k)$	$\in \mathbb{R}$	is the input spuriousness tag function,
$w_{[j]}$	$\in \mathbb{R}$	is a constant *spurious* tag gain,
$\gamma_{[j]}(k)$	$\in \mathbb{R}$	is the conflict level,
$m_{[j]}$	$\in \mathbb{R}$	is a constant conflict level gain,
$\varphi_{[j]}(\cdot)$		is the warning level activation function,
$y_{[j]}(k)$	$\in \mathbb{R}$	is the warning level,

The three constant gains $r_{[j]}$, $w_{[j]}$ and $m_{[j]}$ are design parameters of the warning system for each input j. In addition to system modeling techniques, these gains can be seen as weights in a single layer artificial neural network and could be learned if training data is available.

Warning Level Activation Potential

Without any conflicts or *spurious* tags, Equation (3.4) reduces to:

$$v_{[j]}(k) = -\varphi_{[j]}(v_{[j]}(k-1))r_{[j]},$$

which will drive the warning level activation potential $v_{[j]}(k)$ to zero exponentially. This will allow the system to lower the warning levels of a system once the conflict has been resolved. This is a key component of using a dynamic system, the time to forget will directly impact the allowable maneuverability of the vehicle outside of the nominal operation.

Conflict Detection

The conflict level function will be defined as:

$$\gamma_{[j]}(k) = \frac{t_a}{t_c}, \tag{3.6}$$

where:

t_a maneuverability time,
t_c time to conflict,
t_f trajectory tracking time horizon,

The maneuverability time t_a depends on the robot maneuverability characteristics. It must be chosen to be smaller than the simulation time horizon t_c with some safety factor. This will allow the human agent to maneuver the robot away from the conflict once the warning system provides an alert.

When the conflict is found exactly at the time horizon t_f, meaning $t_f = t_c$, the conflict level γ will be $\frac{t_a}{t_f}$.

This conflict condition of $t_c = t_a$ will add the constant gain $m_{[j]}$ to the warning level activation potential even if the inputs are not spurious. Without any *spurious* tags and a high level when a conflict is present $\gamma_{[j]}(k) = 1$, (Equation 3.4) reduces to:

$$v_{[j]}(k) = -\varphi_{[j]}(v_{[j]}(k-1))r_{[j]} + m_{[j]},$$

which will drive the warning level activation potential $v_{[j]}(k)$ to $m_{[j]}$ exponentially.

If the conflict worsens so that $t_c \to 0$, the conflict level function which will have hyperbolic growth and will raise alarms regardless of all other parameters in the system.

Chapter 4

Framework Tools

The warning system framework was implemented using ROS [18] an open-source Robot Operating System. The code for software implementation is available with an open-source license at [19]. In addition, an independent hybrid automata simulator was written in the Python programming language for this work. It is also available at [19] as "hasimpy", or Hybrid Automata Simulator in Python.

4.1 Hybrid Automata Simulator

An independent simulator was built to be a key piece of this framework. The hybrid automaton model is created using the same structure described in Section 2.1. The size and complexity of the automaton model only limited by the simulating machine and the programming language.

The characteristics of each system mode, such as the guards and edges, can be easily encoded as Python functions and added to an mode object. After giving an initial state and mode, the system will provide the projected path. The simulator can take a control law depending on the mode, state and/or time.

4.2 ROS Implementation

At a basic level, ROS uses Nodes to represent processes and Topics to represent topics being communicated though Nodes. The Nodes/Topics representation for the framework was made to match the different methods defined in Chapter 3. Figure 4.1 shows a graph with the connections between Nodes (Ovals) and other Nodes, using Topics (Rectangles) directly from ROS. Figure 4.2 shows a more legible diagram of the same implementation.

By using ROS the warning system framework can be implemented directly to a large variety of robots compatible or made compatible with the ROS API (Application Programming Interface). Another possibility is to gather the state and input information of the vehicle and implement the warning system in parallel to the vehicle operation. This would be useful for implementation on existing robots that are not compatible with the ROS API, like cargo trucks or heavy machinery vehicles.

Nodes or processes are connected as shown in Figure 4.1. The specific robot topic names shown in this graph correspond to the Turtlebot, a differential drive robot. All robot topics names in ROS are suffixed with the robot name. This allows for the identification of the topic in the event of multiple robots. Specific robot elements in a robot can also have their own topic names unique suffix, for example to separate the pose of the right arm from the pose of the left arm.

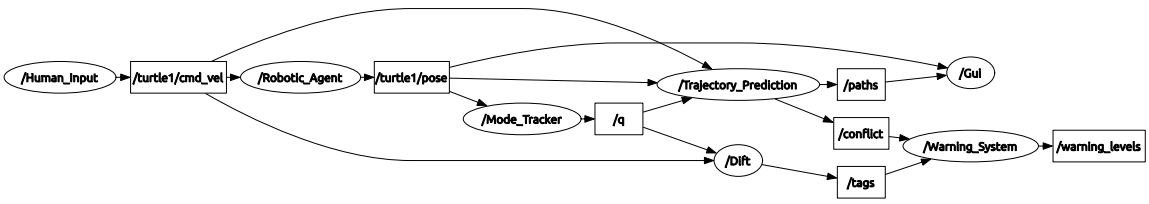

Figure 4.1: ROS Nodes and Topics for Warning System Implementation

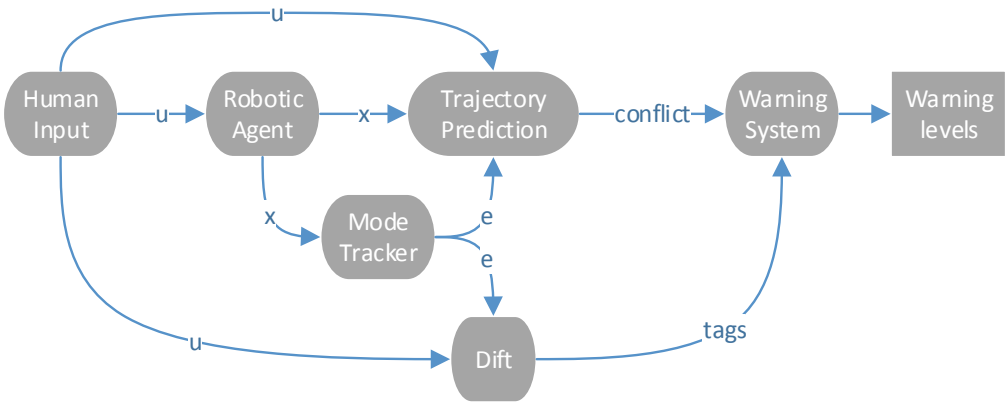

Figure 4.2: System Diagram for Warning System Implementation

4.2.1 Human Input

This node takes the signals from all human controls, like the steering wheel or joystick, and encodes it into a standard ROS message. The message is called a "Twist" which contains the linear and angular velocities for all 3 axis.

The message can be published to any actuator or to a group of actuators. In the case of a group of actuators, the message will be sent directly to the robot driver which would process the message and send appropriate activation signals to all its motors. This encapsulates the complexity of the robot construction details for the robot user.

Publishes: Actuation command (Twist message at $/turtle1/cmd\_vel$)

4.2.2 Robotic Agent

This node handles the robot driver. It will be able to publish topics with information from its sensors and internal processes. As a minimum, it will be subscribed to a command topic, where other node will be able to move the robot with a twist message.

One significant advantage of this system is the ability to change the robot or an element in the robot and have the same code work. In addition, ROS provides its own simulator "Gazebo", which was used during the Darpa Robotic Challenge [20]. For several robots, a robot model can be installed in the simulator and the virtual robot can be interacted using ROS nodes and topics. This makes the warning system ROS implementation neutral to using the Gazebo simulator virtual robot or a real robot. For this thesis, a small simulator specific to the differential drive type of robot was developed. The simulator was developed entirely to be compatible with ROS, subscribing and publishing to the same topics as the real robot does.

Subscribes: Actuation command (Twist message at $/turtle1/cmd\_vel$)

Publishes: Robot pose (PoseStamped message at $/turtle1/pose$)

4.2.3 Mode tracker

The mode tracker is a version of the hybrid automata simulator hasimpy described in Section 4.1. Instead of simulating the robot dynamics or kinematics, it just tracks the real robot behavior to determine when a change in mode occurs, *i.e.,* when any of the guards in the current mode are activated. The mode tracker is initialized at system start. It will subscribe to any state or condition stated in the guards in order to make a decision.

Subscribes: Robot pose (PoseStamped message at $/turtle1/pose$)

Publishes: Mode index (UInt16 or Unsigned 16-bit Integer message at $/q$)

4.2.4 Trajectory Prediction

Starting from the current mode index provided by the mode tracker, this node interfaces with hasimpy to provide information regarding the current trajectory prediction. The system uses the current actuation command provided by the user for the simulation time horizon. During the simulation, it automatically registers any change in mode and verifies if the robot reaches a conflict (encoded as the avoid set) for that mode.

It uses a standard ROS navigation message called "Path" to publish the projected path to the graphical user interface. The information from the path is not used by the warning system directly, it is instead contextual information that could be shown to the user.

The current conflict level, as defined in Subsection 3.1.4, is published using a custom message type called "Conflict" which contains information regarding the first conflict found and the time to reach the conflict.

Subscribes: Robot pose (PoseStamped message at $/turtle1/pose$)

Subscribes: Mode index (UInt16 or Unsigned 16-bit Integer message at $/q$)

Subscribes: Actuation command (Twist message at $/turtle1/cmd\_vel$)

Publishes: Conflict level (custom Conflict message at $/conflict$)

Publishes: Navigation path (Path message at $/paths$)

4.2.5 Dift

This nodes compares the current human agent input with the nominal inputs expected for the current mode index. For each input, a spuriousness tag is assigned following Subsection 3.1.1.

A vector containing all the input spuriousness tags is published in a custom message called "Tags".

Subscribes: Mode index (UInt16 or Unsigned 16-bit Integer message at $/q$)

Subscribes: Actuation command (Twist message at $/turtle1/cmd\_vel$)

Publishes: Spuriousness tags (custom Tags message at $/tags$)

4.2.6 Gui

A basic GUI or graphical user interface was created for this system. The interface was created using the Qt Framework, in particular the Python library PyQtGraph [21], a Python graphics library.

Subscribes: Navigation path (Path message at $/paths$)

Subscribes: Robot pose (PoseStamped message at $/turtle1/pose$)

Figure 4.3 shows a screen shot of the basic GUI developed. The GUI is divided in two areas: top and bottom. In the top, the two white circles with the red needle indicate the current input level.

In addition, it shows green zones to help the human agent know the nominal values for the current mode. The pie chart type indicators next to them indicate the current:

- *Spurious* Tag in yellow.
- Conflict level in red.
- Warning level in orange.

The robot is represented as a grey arrow head and the GUI is designed to show the current path in the 2D plane predicted by the trajectory prediction model.

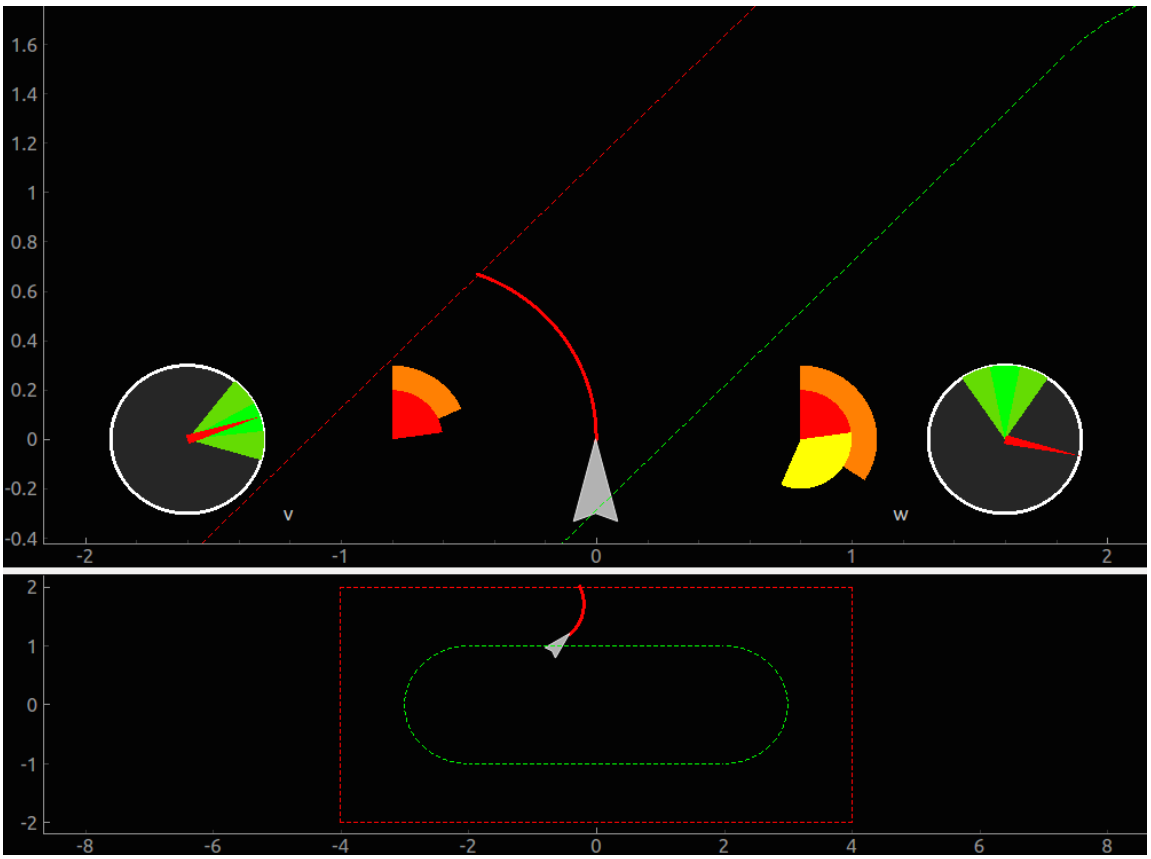

Figure 4.3: Case Study GUI

Chapter 5

Case Study

For this case study, open loop test inputs with a desired response will be used to the chose the parameters of basic a warning system using the framework described in this thesis.

A solution of the system dynamics and parameter design is provided using Control Theory in Section 5.6 and Recurrent Neural Networks in Section 5.7.

A demonstration of the system in action is available at [23].

5.1 Robotic and Human Agent

The robotic agent used was a Turtlebot, a ROS compatible differential drive robot. A human agent can drive the robot using remote control and a graphical user interface (Topic "$/Gui$") on a separate computer.

The inputs for this robot, by its design, are the forward velocity ν and angular velocity ω. Based on this, the trajectory prediction process will be using a kinematic model to predict the Turtlebot behavior.

5.2 Hybrid Automaton Model

A single hybrid automaton will be used for the case study. The hybrid automaton model used will be the following:

$$\mathbf{x} = [x,\ y,\ \theta] \tag{5.1}$$

$$\mathbf{u} = [u_{[0]},\ u_{[1]}] \ = \ [\nu,\ \omega] \tag{5.2}$$

$$\mathbf{Q} = \{q_{[0]},\ q_{[1]}\} = \{Straight,\ Curve\} \tag{5.3}$$

$$\mathbf{F} = \{F_{turtle}(u), F_{turtle}(u)\} \tag{5.4}$$

$$\mathbf{G} = \{x >= 2\ or\ x <= -2,\ x < 2\ and\ x > -2\} \tag{5.5}$$

$$\mathbf{A} = \{\mathbf{A_1}, \mathbf{A_2}\} \tag{5.6}$$

$$\mathbf{A_1} = \mathbf{A_2} = x > 4\ or\ x < -4\ or\ y > 3\ or\ y < -3 \tag{5.7}$$

where:

x, y	are the position coordinates,
θ	is the orientation angle,
ν	is the linear velocity,
ω	is the angular velocity.

The equation describing the continuous-time kinematics in this robot is:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \nu\cos(\theta) \\ \nu\sin(\theta) \\ \omega \end{bmatrix} \tag{5.8}$$

Equation 5.8 will be used for the trajectory prediction process, which can use a continuous-time or discrete-time equation.

5.3 Nominal Input Distributions

The system has 2 inputs and 2 modes, which results in 4 nominal input distributions or one nominal input distribution per input, per mode. For this case study, Gaussian distributions are used.

$$f_{n[j,e]} = \frac{1}{\sigma_{[j,e]}\sqrt{2\pi}} e^{-\frac{(u-\mu_{[j,e]})^2}{2(\sigma_{[j,e]})^2}}. \tag{5.9}$$

The distribution function is normalized using equation 3.2, to obtain the *legitimate* tag function $\lambda_{[j,e]}$ and the *spurious* tag function $\varsigma_{[j,e]}$. The distribution mean will be set to the nominal value for that input and mode, so that $\mu_{[j,e]} = u_{n[j,e]}$. For a semi-circle of radius = 1 and clockwise movement desired, the nominal inputs are shown below, the resulting path "racetrack" is shown in Figure 5.1.

$$u_{n[j=0,e=0]} = 0.2$$
$$u_{n[j=0,e=1]} = 0.2$$
$$u_{n[j=1,e=0]} = 0.0$$
$$u_{n[j=1,e=1]} = -0.2$$

The scaling factor σ for the nominal input distribution functions in Equation 5.3 are:

$$(\sigma_{[j=0,e=0]})^2 = 0.1$$

$$(\sigma_{[j=0,e=1]})^2 = 0.1$$

$$(\sigma_{[j=1,e=0]})^2 = 0.1$$

$$(\sigma_{[j=1,e=1]})^2 = 0.05$$

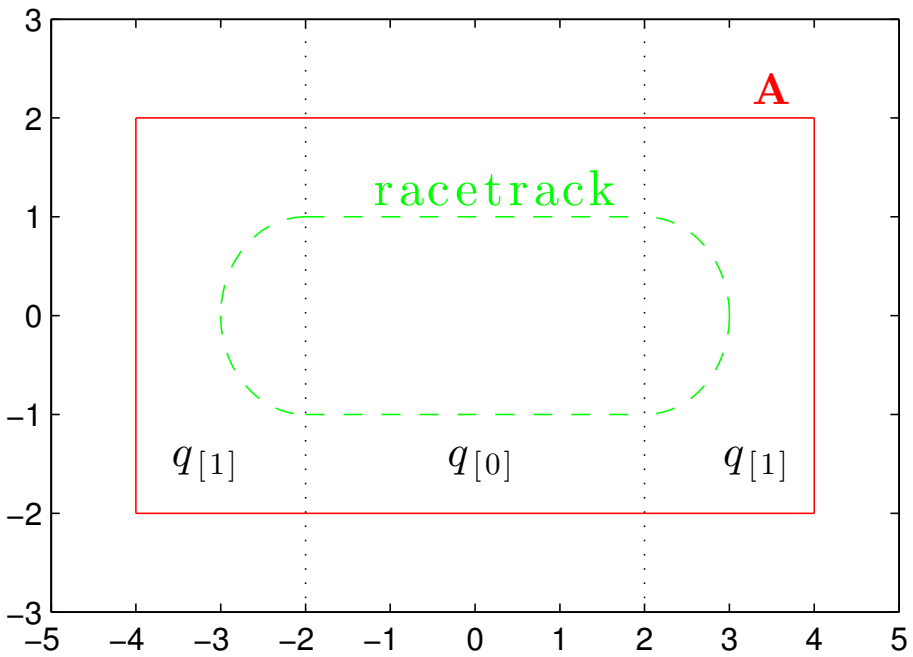

Figure 5.1: Hybrid automaton modes $\{q_{[0]}, q_{[1]}\}$, racetrack example path and avoid set **A**.

5.4 Alarm Thresholds

Using the warning level value, alarm thresholds can be configured to trigger graphical and/or auditory alarms to the human agent. A basic alarm thresholds configuration desired for this case study is shown in Table 5.1

Table 5.1: Alarm Threshold Configuration Scheme

***Spurious* Tag**	**Conflict**	**Warning Level**
None	None	None
Low	None	Low
High	None	Medium
None	Low	Medium
None	High	High
Low	Low	Medium
High	Low	Medium
Low	High	High
High	High	High

Where the following alarm ranges are defined:

$$\begin{aligned} &\text{None} && y < 0.1, \\ &\text{Low} && y >= 0.1 \text{ and } y < 0.3, \\ &\text{Medium} && y >= 0.3 \text{ and } y < 0.7, \\ &\text{High} && y >= 0.7 \end{aligned}$$

Where higher levels of alarm might trigger different responses in the vehicle control panel warnings or activate automated responses for a particular condition.

5.5 Test Inputs

During these test inputs, combinations from Table 5.1 are used. The following sequence of events will occur:

1. Robot uses nominal inputs.
2. *Spurious* inputs and/or future conflicts detected.
3. Robot recuperates to no warning level conditions.

For consistency the following variables will be used to indicate the time associated with these events:

t_0 simulation start,
t_{ss} *spurious* inputs start,
t_{se} *spurious* inputs end,
t_{cs} future conflict present start,
t_{ce} future conflict present end,

For each test input, a desired warning level response will be provided following the characteristics of Table 5.1. In addition to the desired warning level, a ramp up/ramp down is added to the desired values to empathize that a dynamic response is possible from the warning system. This would contrast with performing a static mapping from spurious tag and conflict level to the warning level, which would lose the history of previous warning levels and thus possibly lose context. The ramp up/down time chosen is 1.0 seconds, based on the robot maneuverability characteristics.

The start time of the desired ramp up response will be 1.0 seconds after t_{ss} or t_{cs}, the end time of the desired ramp up time will be t_{ss} or t_{cs}, depending which is

present first. The start time of the desired ramp down response will be t_{se} or t_{ce}, the end time of the desired ramp down time will be 1.0 seconds after t_{se} or t_{ce}, depending which is present last.

During all the test inputs, the forward velocity ν remains constant at the nominal input value. All maneuvers are performed using only the angular velocity input ω with the intention of observing a clear relationship between the change in input variable and warning level.

For each test input, 3 figures are provided: the path taken by the robot, the ω input values, the conflict level and the *spuriousness* tag or dift tag based on Section 5.3.

5.5.1 Arc Test Input: Low *spurious* tag, No conflict level

The robot is driven from the initial conditions:

$$\begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix} = \begin{bmatrix} -1.0 \\ 1.0 \\ 0.0 \end{bmatrix}$$

This test represents a change in mode in the wrong time. The movement after t_{ss} would be the nominal operation in $q_[1]$, however it is considered *spurious* in $q_[0]$. The *spurious* tag is low and constant at $\varsigma = 0.181269$ during the arch section.

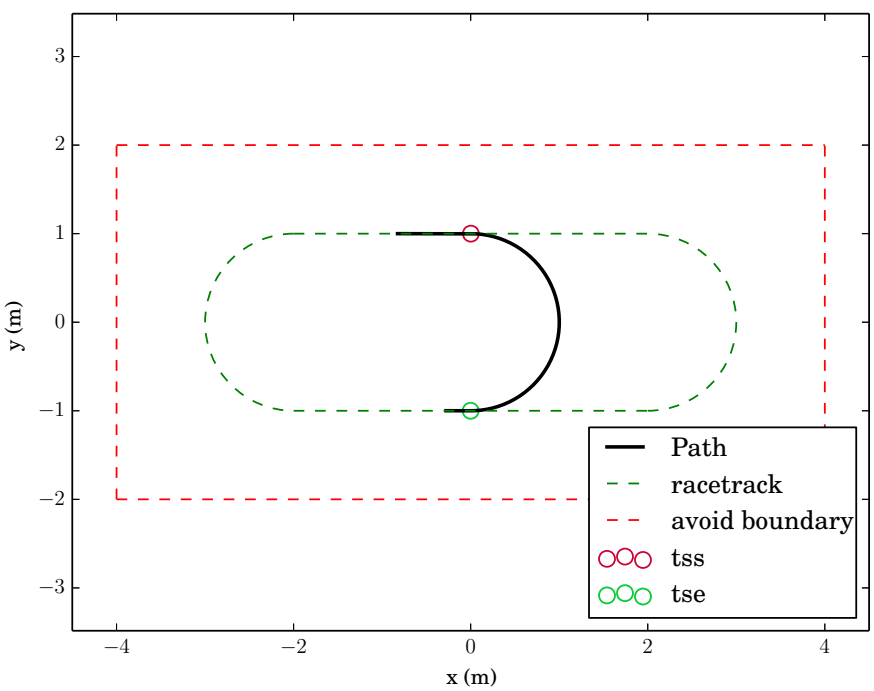

Figure 5.2: Path for Arc Test Input

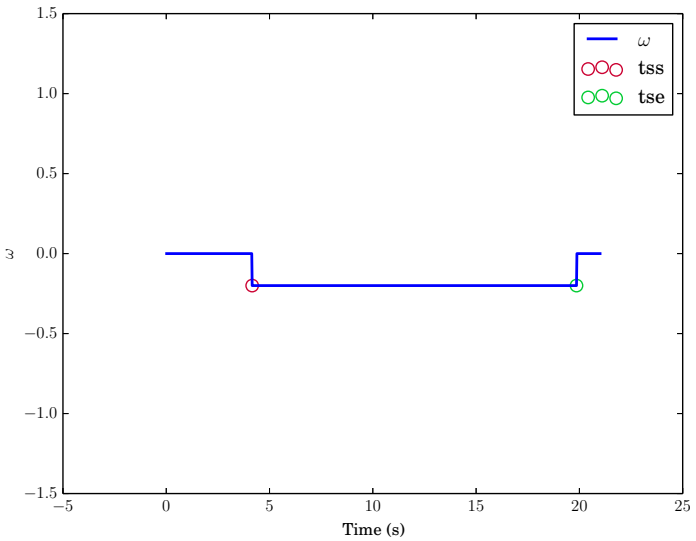

Figure 5.3: Inputs for Arc Test Input

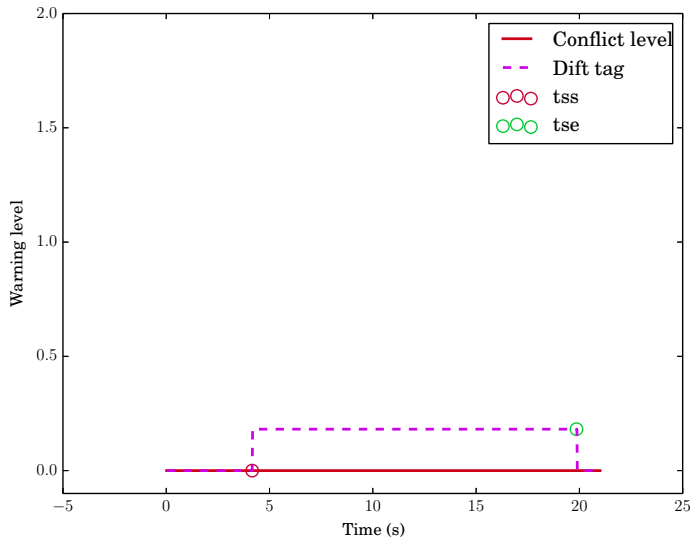

Figure 5.4: Dift tags and Conflict level for Arc Test Input

5.5.2 Diagonal Test Input: No *spurious* tag, High conflict level

The robot is driven from the initial conditions:

$$\begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix} = \begin{bmatrix} -1.0 \\ 1.0 \\ \frac{\pi}{4} \end{bmatrix}$$

During this test input the robotic agent uses nominal inputs for mode $q_{[0]}$, however due that it started in an initial yaw angle θ of 45 °, the robot has a trajectory that will result in a conflict if not corrected. In order to avoid the collision the robot uses *spurious* inputs. However, following the scheme in Table 5.1, the warning level is expected to decrease due that conflicts alone are considered worst cases than *spurious* inputs alone.

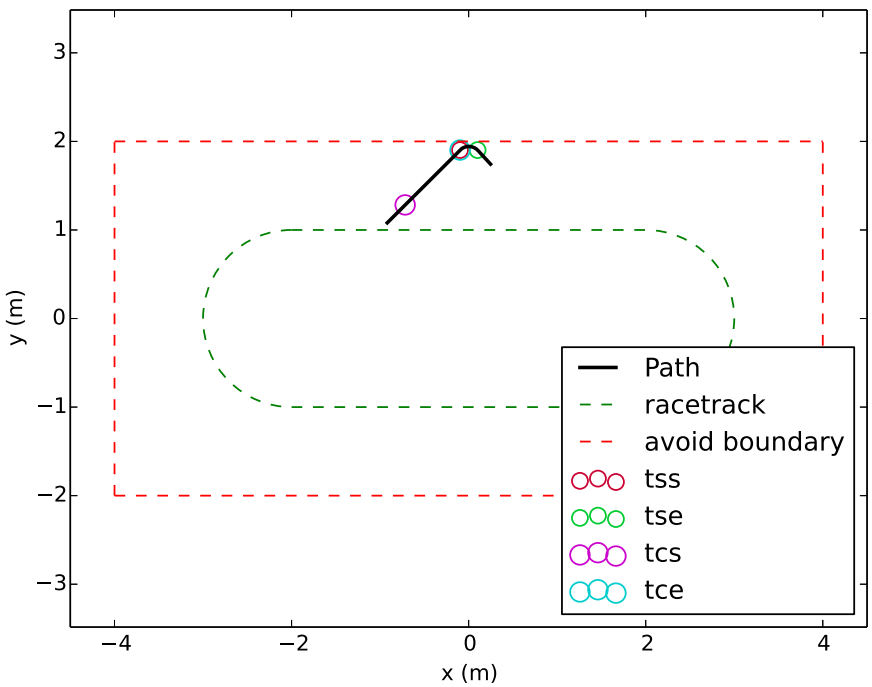

Figure 5.5: Path for Diagonal Test Input

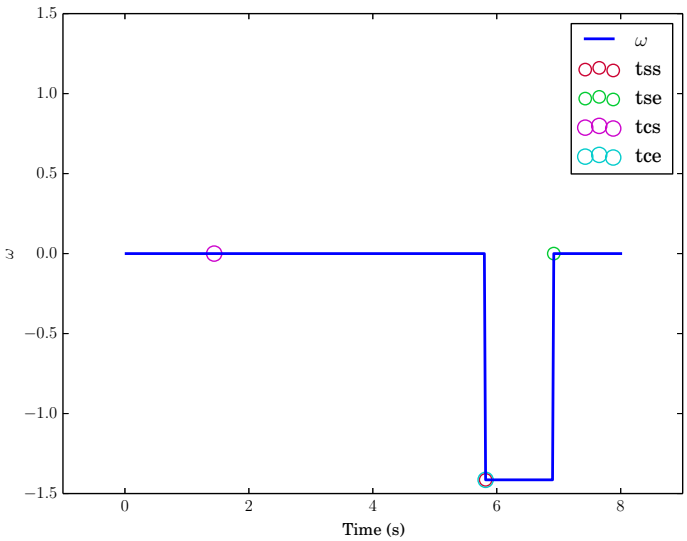

Figure 5.6: Inputs for Diagonal Test Input

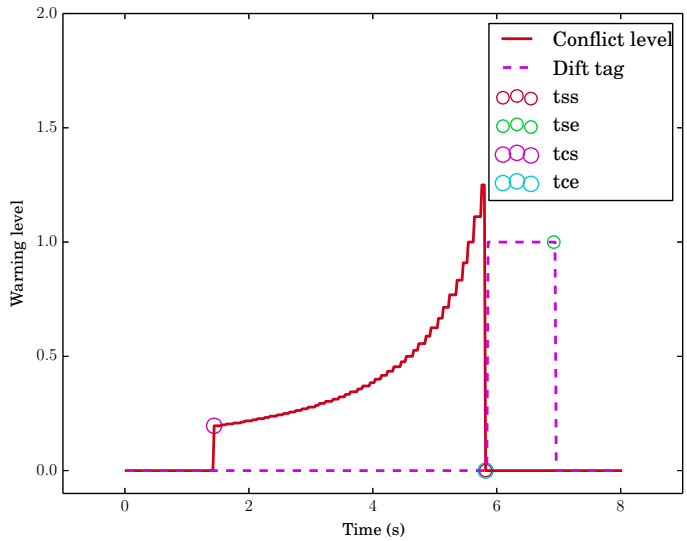

Figure 5.7: Dift tags and Conflict level for Diagonal Test Input

5.5.3 Bump Test Input: Low *spurious* tag, Medium conflict level

The robot is driven from the initial conditions:

$$\begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix} = \begin{bmatrix} -2.0 \\ 1.0 \\ 0.0 \end{bmatrix}$$

This is a more complex input pattern which involves both *spurious* inputs and a expected conflict. This would be, for example, an aggressive maneuver to avoid an obstacle.

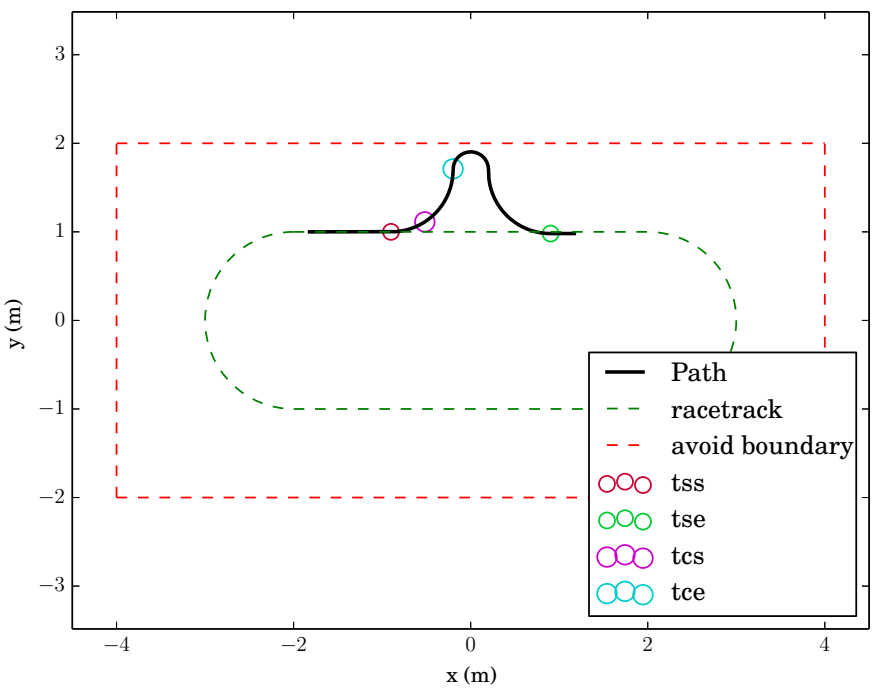

Figure 5.8: Path for Bump Test Input

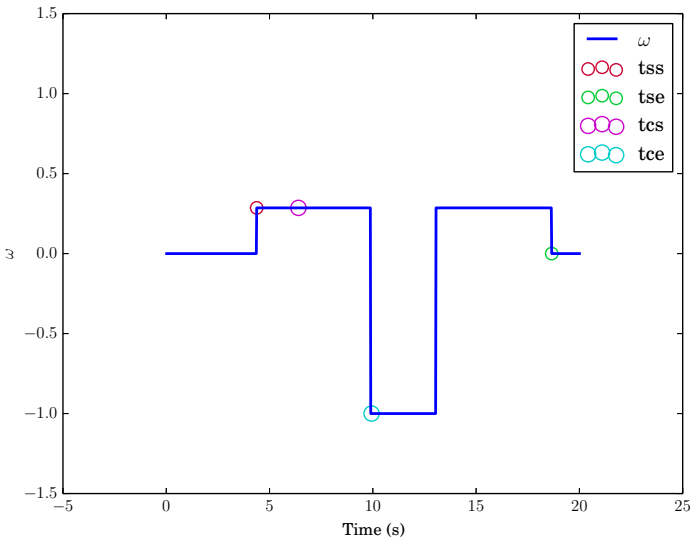

Figure 5.9: Inputs for Bump Test Input

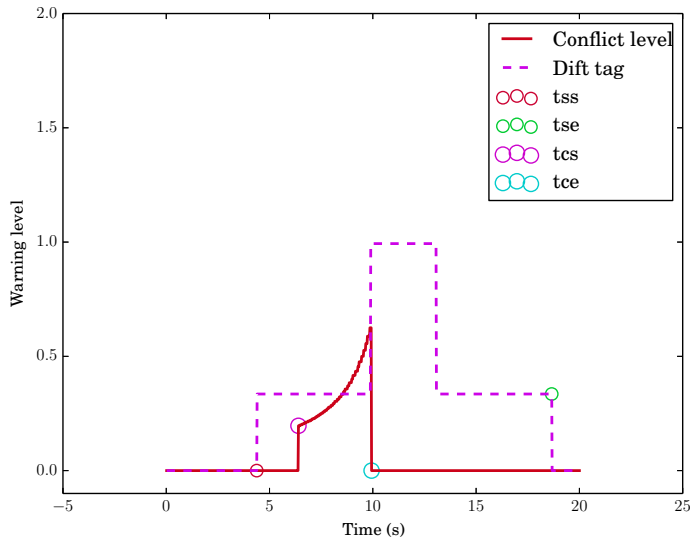

Figure 5.10: Dift tags and Conflict level for Bump Test Input

5.6 Control Theory Method

The warning level activation function $\varphi(\cdot)$ from Subsection 3.1.4 is replaced for a second-order system response and a saturation function. The saturation function will saturate the response to the desired warning level range from 0 to 1. The parameters of the system are designed in the frequency domain due to the availability of closed-form solutions for a general second-order system [22]. The system block diagram for a general second order system is shown Figure 5.11.

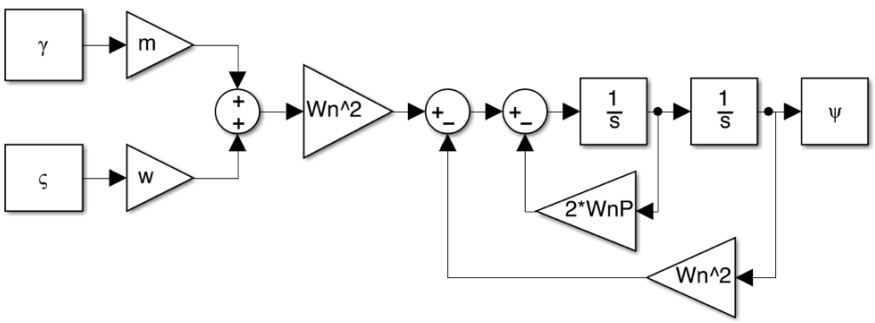

Figure 5.11: General second order system block diagram

In this case, an underdamped second-order system is desired with a damping ratio of 0.8 and a settling time of 1 second corresponding with the desired ramp up/down time. In the frequency domain, this results in Figure 5.12 and the following transfer function:

$$G(s) = \frac{25}{s^2 + 8s + 25} \tag{5.10}$$

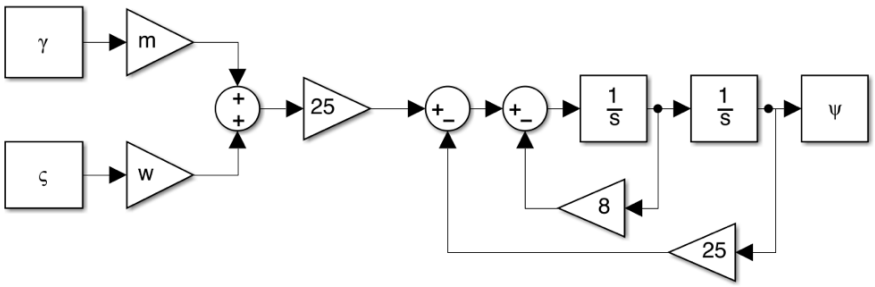

Figure 5.12: Second order system block diagram used

The inputs to the second-order system remain a linear combination of the conflict level γ and the *spurious* tag ς.

The weight or gain m for the conflict level is chosen as 0.9 following Equation 5.1 and Table 5.1. For conflict level of 1, the warning level will converge at 0.9, which is approximately in the middle of the high alarm level range. Similarly, gain w is chosen as 0.5, which is approximately in the middle of the medium alarm level range.

The system was simulated using the 3 test inputs explained in Section 5.5.

The following plots show the warning level desired response and saturated second-order response. The construction of the desired response is based on Table 5.1. Flat values were used for desired warning levels with *spurious* inputs following the *spurious* tags observed in the simulations, adding ramps up and and down as indicated in Section 5.5. For the section of rapid increase in conflict level, a straight line was used as desired response reference.

The warning level provided by the second-order response function converges to the desired values and shows the desired ramp up/down characteristics.

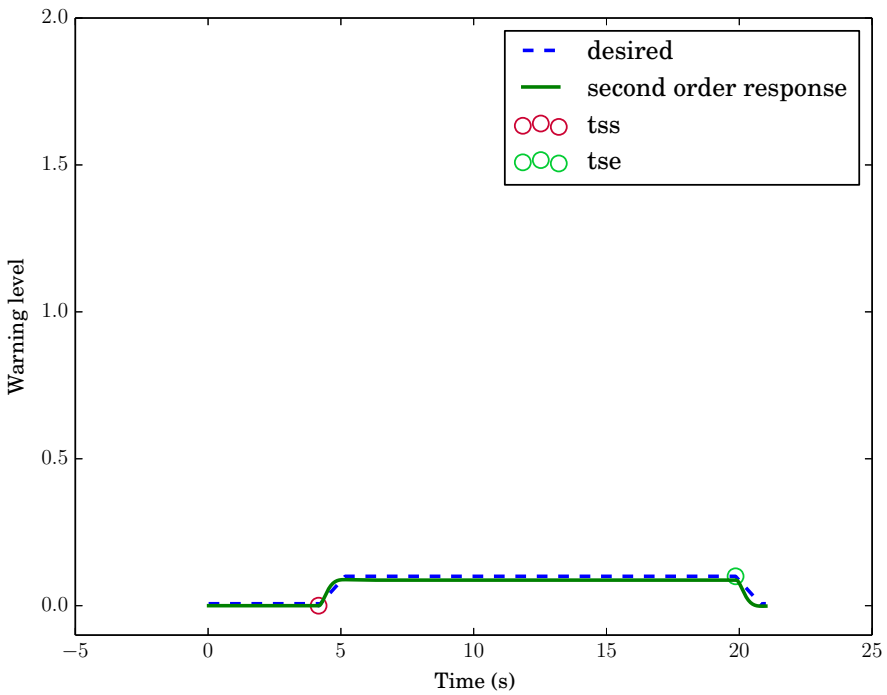

Figure 5.13: Desired vs Learned Warning levels for Arc Test Input

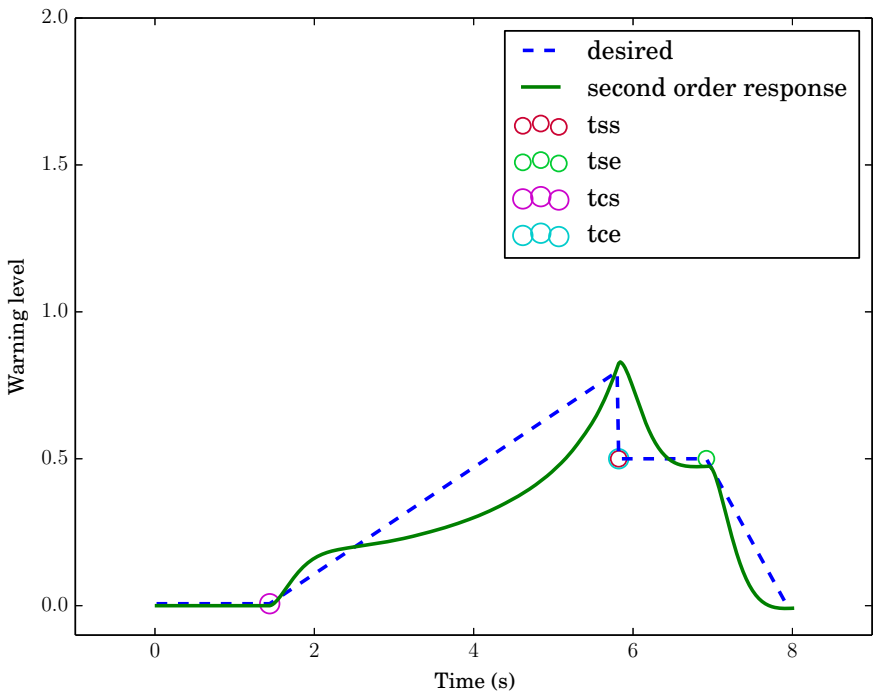

Figure 5.14: Desired vs Learned Warning levels for Diagonal Test Input

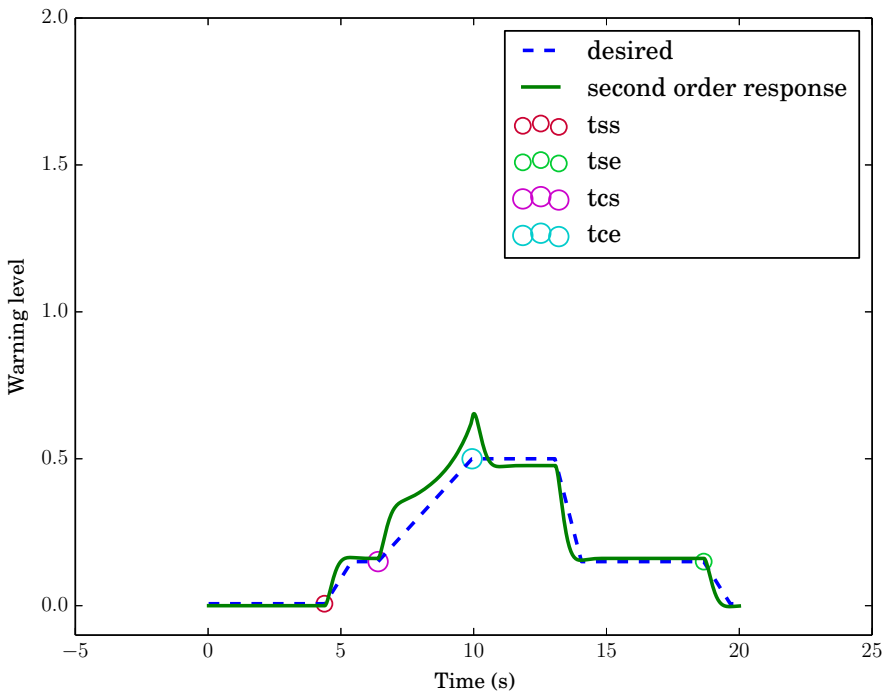

Figure 5.15: Desired vs Learned Warning levels for Bump Test Input

5.7 Recurrent Neural Network Method

A separate method was used to find the a warning level dynamic response using recurrent neural networks. This exploits the coincidence between the dynamic model presented in 3.1 and a recurrent neural network model for a single neuron.

The parameters of the system were calculated using a modified version of the real-time recurrent learning algorithm [17]. The modification stems from having desired data fitting a type of dynamic response from the simulation, instead of an input-output mapping. In addition, 3 simulations were done following the 3 test inputs. The chosen method trained the weights on each test input separately, averaged the resulting weights and then trained the network using all the test inputs. Details of the approach follow below.

The following steps were followed for each test input. The warning level inputs are the same linear combination of the conflict level γ, the *spurious* tag ς and the previous value $y(k-1)$. The desired values for the dynamic system are the same as in Section 5.6. The weights of the linear combination, in addition to a bias input set to 1 used for this algorithm, form the weight vector W. Finding the appropriate value for W so that the warning level response fits the desired response is the objective of this method.

The system must be initialized with a value for W, the initial value used was:

$W_0 = [-0.15,\ 0.1,\ 0.3,\ 0.4]$

The following steps were followed to train the system:

1. Initialize same W_0 to all test inputs.

2. For each test input:

- Present all the test input data with the current weights and record the learning algorithm ΔW.
- Apply a change in W, according to $W_{new} = W_{previous} + \Delta W$, until convergence to W_{final}.
- Calculate the total change from the vector W_0, $\Delta W_{total} = W_0$ - W_{final}.

3. Calculate the mean of all the ΔW_{total}
4. Re-initialize same $W_{mean} = W_0 + \Delta W_{mean}$ to all test inputs.
5. Present the input data from all the test inputs sequentially and add the learning algorithm ΔW from each one.
6. Apply a change in W to all the test inputs, according to $W_{new} = W_{previous} + \Delta W$, until convergence to W_{final}.

The warning level activation function $\varphi_{[j]}(\cdot)$ used corresponds to the following sigmoid function:

$$\varphi_{[j]}(v_{[j]}(k)) = \frac{1}{1 + \exp\left(-\alpha(v_{[j]}(k) - 0.5)\right)}, \tag{5.11}$$

where $\alpha > 0$ is a design parameter chosen so that for $v_{[j]}(k) = 0$, $\varphi_{[j]}(v_{[j]}(k)) \approx 0$; and for $v_{[j]}(k) = 1$, $\varphi_{[j]}(v_{[j]}(k)) \approx 1$. For $\alpha = 10$, the activation function is shown in Figure 5.16

The response of the recurrent neural network system converged the warning level at the desired ranges (low, medium and high) however not at the desired values presented. In addition, the ramp up and ramp down desired characteristic was not present in this method. The learning procedure chosen was not sufficiently robust to try a wide range of initialization weights, values close to the solution were needed for convergence. This limitation can be reduced by designing the simulation experiments, with input and desired outputs, in conjunction with the learning procedure.

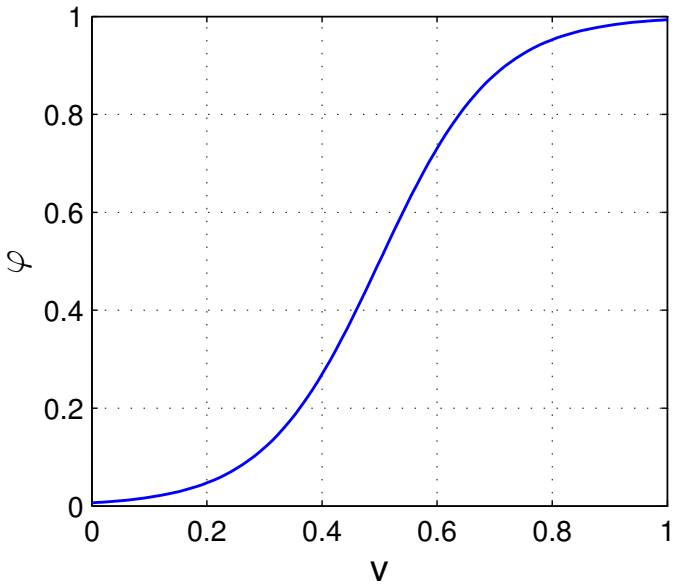

Figure 5.16: Activation Function for $\alpha = 10$

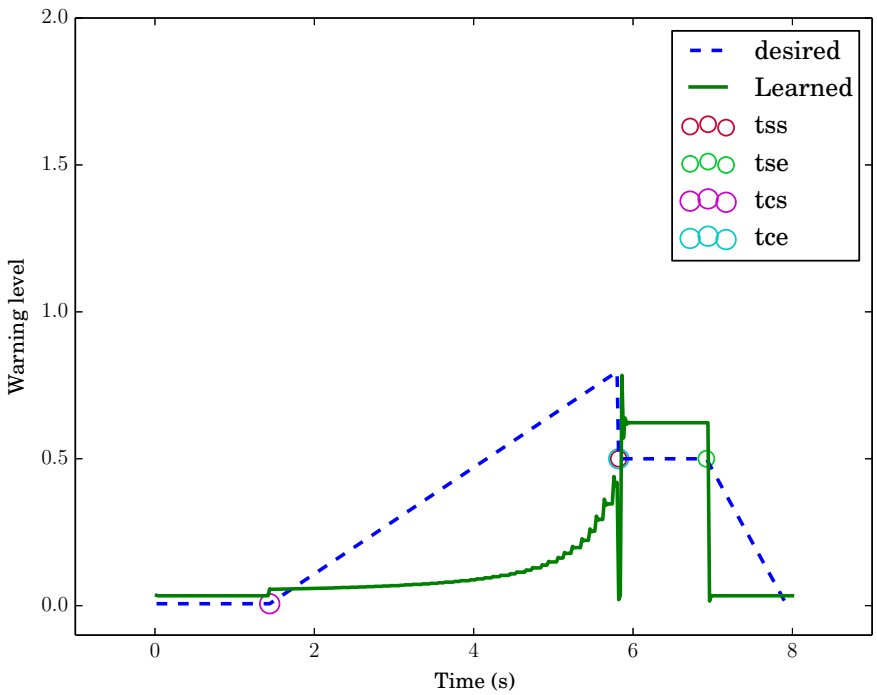

Figure 5.17: Desired vs Learned Warning levels for Diagonal Test Input

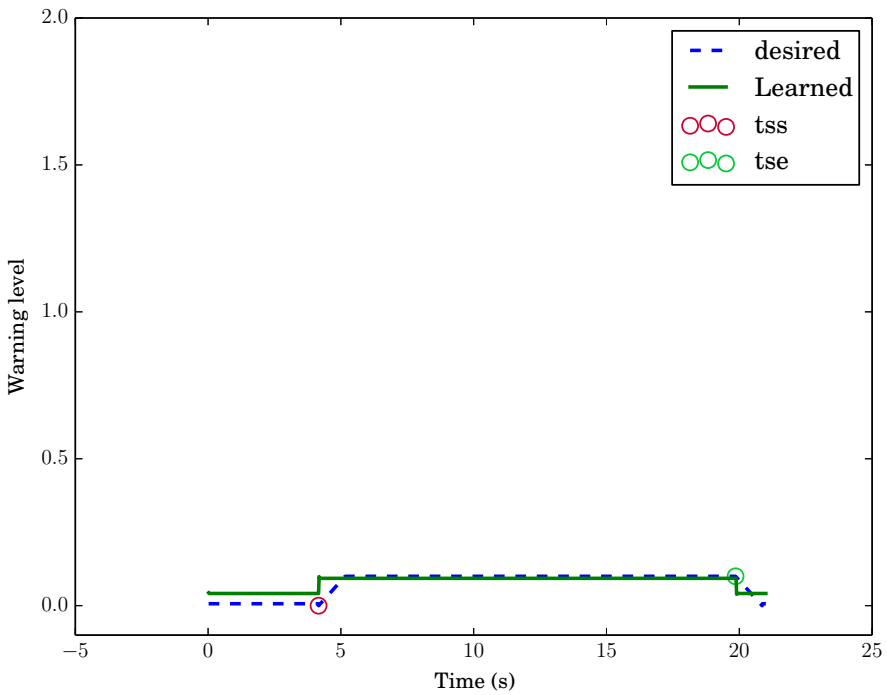

Figure 5.18: Desired vs Learned Warning levels for Arch Test Input

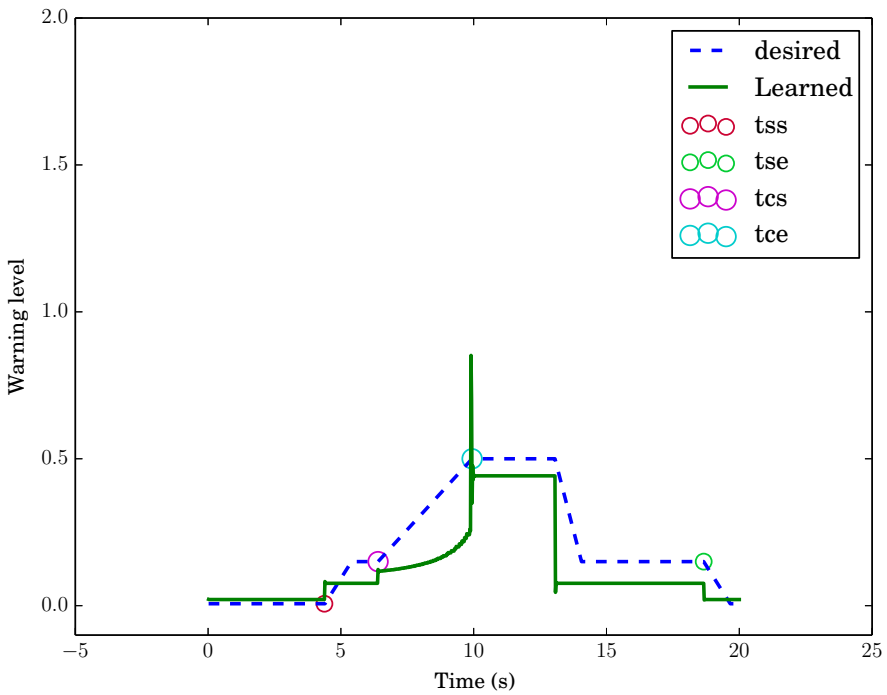

Figure 5.19: Desired vs Learned Warning levels for Bump Test Input

Chapter 6

Conclusions

A real-world warning system framework was developed in this thesis. The provided tools can be used to design and use a warning system on a robotic agent under shared control. In addition, the theoretical framework can be used to further research on a warning system theory for robotics and other vehicles under shared control.

This thesis presented a contribution to forming such theory by applying an analogy to Dynamic Information Flow Tracking methods, which are able to provide an alarm before any conflicts arise. These characteristics should make it attractive to applications such as commercial airplanes warning systems, particularly due to the complexity and number of inputs available to the human agent and the difficulty of recovering from *spurious* inputs.

The framework used an encoding of the nominal operation of the robotic and human agent by applying behavior based hybrid automata. The formulation of all the components of the framework in terms of the hybrid automata theory helps assure that performance analysis of the resulting warning system will be available. Moreover, the implementation using open-source software and robotic software tools, will increase the probability of its adaption and growth.

6.1 Future Work

Performance analysis tools will be of great utility for this framework. Combining techniques developed to compute the reachability of the avoid set and the warning system parameters to find weak points or even optimize its behavior.

A key question to solve will be: Can it be guaranteed for a particular warning system design that a warning will be provided before reaching a conflict? Ideally before the maneuverability time. For example, in the case study presented the orientation of the robot θ was not included in the definition of the avoid set. By not including this variable, it is easy to see that if the robot travels parallel and close to the avoid set, a small turn would result in reaching the avoid set boundary. For more complex robots with dynamics of higher conventionality, a modified computation of the reachability of the avoid set would help future warning system designers.

References

[1] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure program execution via dynamic information flow tracking." in *ASPLOS*, S. Mukherjee and K. S. McKinley, Eds. ACM, pp. 85–96.

[2] P. Antsaklis, J. Stiver, and M. Lemmon, "Hybrid system modeling and autonomous control systems," in *Hybrid Systems*, ser. Lecture Notes in Computer Science, R. Grossman, A. Nerode, A. Ravn, and H. Rischel, Eds. Springer Berlin Heidelberg, 1993, vol. 736, pp. 366–392. [Online]. Available: http://dx.doi.org/10.1007/3-540-57318-6_37

[3] C. J. Tomlin, "Toward reachability-based controller design for hybrid systems in robotics," 2011.

[4] M. Egerstedt, K. Johansson, J. Lygeros, and S. Sastry, "Behavior based robotics using regularized hybrid automata," in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, vol. 4, 1999, pp. 3400–3405 vol.4.

[5] M. Oishi, I. Mitchell, A. Bayen, C. Tomlin, and A. Degani, "Hybrid verification of an interface for an automatic landing," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 2, Dec 2002, pp. 1607–1613 vol.2.

[6] Inseok Hwang and Chze Eng Seah, "Intent-based probabilistic conflict detection for the next generation air transportation system," *Proceedings of the IEEE*, vol. 96, no. 12, pp. 2040–2059, Dec. 2008. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4745649

[7] G. Chatterji, "Short-term trajectory prediction methods," 2014/09/27 1999. [Online]. Available: http://dx.doi.org/10.2514/6.1999-4233

[8] V. Nagarajan, H.-S. Kim, Y. Wu, and R. Gupta, "Dynamic information flow tracking on multicores," in *Proceedings of the Workshop on Interaction*

between Compilers and Computer Architectures, 2008. [Online]. Available: http://homepages.inf.ed.ac.uk/vnagaraj/papers/interact08.pdf

[9] M. Dalton, H. Kannan, and C. Kozyrakis, "Raksha: a flexible information flow architecture for software security," in *ACM SIGARCH Computer Architecture News*, vol. 35. ACM, 2007, pp. 482–493. [Online]. Available: http://dl.acm.org/citation.cfm?id=1250722

[10] H. Kannan, M. Dalton, and C. Kozyrakis, "Decoupling dynamic information flow tracking with a dedicated coprocessor," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, June 2009, pp. 105–114.

[11] N. Kohl, K. Stanley, R. Miikkulainen, M. Samples, and R. Sherony, "Evolving a real-world vehicle warning system," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2006. [Online]. Available: http://nn.cs.utexas.edu/?kohl:gecco06

[12] E. Coelingh, A. Eidehall, and M. Bengtsson, "Collision warning with full auto brake and pedestrian detection - a practical example of automatic emergency braking," in *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, Sept 2010, pp. 155–160.

[13] E. Ackerman, "Volvo tech makes trucks smart enough to not run you over," http://spectrum.ieee.org/cars-that-think/transportation/safety/volvo-tech-makes-trucks-smart-enough-to-not-run-you-over, 2014.

[14] J. Lygeros, S. Sastry, and C. Tomlin, *Hybrid Systems: Foundations, advanced topics and applications*. Springer Verlag, unpublished, 2012.

[15] BostonDynamics, "Atlas," http://archive.darpa.mil/roboticschallengetrialsarchive/files/ATLAS-Datasheet_v15_DARPA.PDF, 2013.

[16] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[17] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[18] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[19] R. Figueroa, "Warning system framework tools," https://github.com/rafafigueroa/wsdift, 2014.

[20] E. Ackerman, "Darpa's virtual robotics challenge: Osrf gets simulator ready," http://spectrum.ieee.org/automaton/robotics/robotics-software/osrf-prepares-for-darpa-virtual-robotics-challenge, 2013.

[21] L. Campagnola, "Pyqtgraph," http://www.pyqtgraph.org/, 2012.

[22] N. Nise, *Control Systems Engineering*. Wiley, 2010. [Online]. Available: http://books.google.com/books?id=WVA8PwAACAAJ

[23] R. Figueroa, "Warning system framework demo," https://www.youtube.com/user/RafaelFigueroaEng, 2014.