The Kaizen Agent
A self-driving car continuously learns by imagination

Sara Mahmoud

May 2019

Abstract

For an agent to autonomously interact in a real world environment, it needs to learn how to behave in the different scenarios that it may face. There are different approaches of modeling an artificial agent with interactive capabilities. One approach is providing the agent with knowledge beforehand. Another approach is to let the agent learn from data and interaction. A well-known techniques of the former approach is supervised learning. In this approach, data is collected, labeled and provided to train the network as pre-defined input and correct output as a training set. This requires data to be available beforehand. In a real world environment however, it is difficult to determine all possible interactions and provide the correct response to each. The agent thus needs to be able to learn by itself from the environment to figure out the best reaction in each situation. To facilitate this, the agent needs to be able to sense the environment, make decisions and react back to the environment. The agent repeats this trying different decisions. To learn from these trials, the agent needs to accumulate old experiences, learn and adjust its knowledge and develop progressively after each interaction. However, in many applications, experiencing various actions in different scenarios is difficult, dangerous or even impossible. The agent therefore needs an experimental environment where it can safely explore the possibilities, learn from experiences and develop new skills.

This research aims to develop a methodology to build an interactive learning agent that can improve its learning performance progressively and perform well in real world environments. The agent follows the Japanese concept *Kaizen* which refers to activities that continuously improve all functions. It means striving for continuous improvement and not radically changing processes. The contribution of this research is first to model and develop this agent so that it can acquire new knowledge based on existing knowledge without negatively affecting the old knowledge and skills. Secondly, this research aims to develop a novel method to systematically generate synthetic scenarios that contributes to its learning performance.

This proposal consists of the background of artificial cognitive systems, a comparison of the theories and approaches in artificial cognitive systems for developing a learning interactive system, and a review of the state of the art in reinforcement learning. Imagination-based learning is discussed and the purposes of imagination are defined. Imagination for creation is used as a scenario generator for practicing new skills without the necessity to try them all in the real world. The research proposal results in the research questions and objectives to be investigated as well as an outline of the methodology.

Contents

List of Figures

List of Tables

1 Introduction

Development and learning are important characteristics for an autonomous agent. An agent that starts with small or no knowledge about the world, needs to improve its skills and knowledge to adapt to this world. Humans for example are born with very limited capabilities. During a life time, humans develop lots of skills. For example, at the first few months of an infant live, an infant learns how to control her hands and fingers with simple movements. Then by the fifth year, the child learns walking, talking and eating. After that, she learns reading and writing. At a later stage, she learns how to run a business. These skills are gradually developed through a life time one after the other. Similarly in artificial agents, it may be difficult to develop an agent with all the required knowledge all at once. If the agent is able to learn and develop progressively, it will not be limited to only the learning at the beginning of the development. The agent would be able to adapt to new learning while using the previous ones. In artificial cognitive systems, a learning interactive agent is a system that perceives the surrounding, takes a decision, and consequently perceives the change in the surrounding world and learns from this experience. In this research, an experience is defined as the complete cycle of perceiving the state of the agent, selecting an action and observing the change in the environment after executing this action.

A definition of learning could be the ability to tune the skill for a specific outcome. An example of an interactive learning agent is a self-driving car. The car safely goes from its current location to a certain location following the traffic rules. The car should be able to recognize objects such as other cars, pedestrians and traffic lights. In addition, it should drive only in the allowed roads, not at the side of the road or the opposite direction. Taking the self-driving car application, some examples of the skills are; object recognition, lane keeping driving and traffic light rules. Tuning a certain skill is learning such as finding out to drive within the road lane without crashing into other cars. In this learning, the agent should keep. In this context, development is defined as building new skills on top of already acquired ones. In the self-driving car example, a development is the ability to learn how to overtake a car on front on top of the lane keeping skill. After that, developing the ability to adjust the speed in different types of roads. Although the speed control is not really coupled with the overtake, it depends on it. The agent needs to increase its speed while being within the limit when performing an overtake task.

One approach of learning is the supervised learning. This approach requires the availability of the training data prior to the training process. The real world is usually characterized by uncertainty. This makes it difficult to collect the data for all the possible scenario. Therefore, it is inappropriate for a real world interactive agent to learn in a supervised approach. Another approach is learning by experiences. In this approach the agent interacts with the environment and learns from the correctness of the experience. In this case, the agent needs to explore the different possibilities and observe the outcome of the interaction, evaluate it and learn from the experience. One of the interactive learning

mechanisms is the reinforcement learning (RL). In RL, an experience consists of observing the current state, taking an action, receiving a reward for the taken action and then observing the new state the agent transited to. The agent learns the optimal policy from the accumulated experiences. A developmental agent needs to be modeled such that it can learn skills progressively.

One of the challenges in learning from a real world environment is that some experiences could be difficult or impossible to try out. In the self-driving car example, it may be difficult or dangerous to explore all the possible scenarios in order to find the optimal decision. For example, crashing into cars or pedestrians is dangerous to experience in the real world. A suggested solution is inspired from human cognition. Humans uses imagination to simulate different alternatives to learn from there experiences. Using similar mechanism allows the artificial agent to experience different scenarios until it learn from the simulated environment without experiencing them in the real world. The definition if imagination in artificial agents is inferred from Hesslow's definition of mental simulation and imagination [10]. In this context, imagination would be reactivating the sensed motors without experience the scenario in the real world, and being able to reactivate the action motor without acting in the environment.

One of the challenges for this approach is generating imagination scenarios that the agent needs to learn from. In the self-driving car example, an imagined scenario could be different number of cars when learning the overtake task. A scenario could be with one target car, which is to be overtaken. Another scenario is having another car in front of the target car. Besides, the difficulty of the scenarios could vary depending on the current capabilities of the agent. An agent that still learns how to navigate may not be able to learn in an environment full of rapidly moving objects. It may start with one slowly moving object, increasing the number of objects gradually.

This proposed research aims to study the method of develop an interactive learning agent that can progressively learn and develop one skill after the other. The contribution of this research is firstly, a learning mechanism for an interactive agent that can develop different skills and build upon each others. Secondly, an imagination mechanism that extracts features and creates new scenarios from those features. Thirdly, feature selection mechanism of what and when to imagine each feature such that it gives the highest learning impact in the interactive agent. This research takes a self-driving car as an application scenario for the interactive agent.

2 Background and Literature

There are different paradigms for modeling an artificial cognitive systems. Each paradigm has strengths and weaknesses. Different approaches can be used to model a cognitive paradigm. This chapter begins with a comparison of the different artificial cognitive-system paradigms. Based on this comparison, a paradigm is nominated for exploring the mechanisms of developing the interactive learning agent. Reinforcement learning is then presented as a nominated learning mechanism for the developing the agent. Next, three approaches to imagination based learning are discussed to present the different purposes of imagination in artificial agents.

2.1 Artificial Cognitive Systems

There are two main paradigms in viewing cognitive systems. The first paradigm, the cognitivism, is based on deduction, where the conclusion is reached by following set of rules that leads to a final outcome. The second paradigm, the emergent, builds its conclusion in an inductive way. In this approach, the agent analyses the data to find a common rule that generalizes the pattern in the data. For detailed comparison refer to [48].

2.1.1 Cognitivist paradigm

The main idea in cognitivist approach is that cognitism is logical inference based on a knowledge base and a set of rules; cognitivism is based on symbolic information processing. In other words, in the cognitivism, the world objects are modeled in terms of explicit physical symbols and processing based on these representations [27]. The physical world objects correspond to model symbols used by the agent to represent the information about the world. The associations between symbols forms rules used by the agent to infer information and behavior. Using this approach, the agent is thought to be able to reason to infer more knowledge about the world in a logical deduction process, which allows the agent to adopt the actions that leads to the intended goal. The knowledge-based rules determine how the agent behaves in certain situations.

One of the cognitivist-approach techniques is the state space search [45]. In order to reach a certain goal, the system finds the sequence of transitions that may lead to the final goal. In search techniques, the system organizes the state space in a way that allows the agent to find the path of states sequence to the goal efficiently in minimum time within graphs and trees representation of the search space [14]. Here, problem-solving proceeds across states of the search space. The agent matches the current situation with one of the states in the search space, then searches for the next state that brings it closer to the goal. A sequence of states is subsequently constructed to show how to reach the required goal. According to this deliberated sequence, the agent performs the action that transfers it to the next state. The search process can use several algorithms,

such as brute force algorithms [46], best-first, depth-first search algorithms [34] and greedy attribute selection [4].

Cognitivist approaches have for example been used in computer vision processing [26], where the agent processes an image to extract objects from the perceived image. Each object has a model representation and set of rules used by the agent to infer and predict the action. This gives the agent the ability to recognize and track surrounding-objects and determine the suitable reaction to interaction with real-world objects.

An example of a cognitivist system used for self-driving cars, is a simulation-based system [33], where the car agent gets the input as events generated by the simulator. The agent also retrieves road-directions from a text-file defining the next direction in a ruled-based system. The car uses the input values of road directions such as driving right, left or straight starting from the current location until it reaches the targeted destination. The input specifies also the location of traffic signals and crossing lines. The car specifies whether the traffic-light is red in order to stop or green in order to drive. The predetermined rules control the behavior of the car when reaching traffic lights and the crossing lines.

Strength of the cognitivist paradigm

The strength of cognitivist approaches comes from the ability to explicitly infer the knowledge from a given set of rules and a pre-defined model of objects representation. The system is able to recognize, model and operate physical objects [26]. The agent is able to infer logical interpretations for each symbol to figure out the consequence of doing an action and what would be the next state, which expands the agent's knowledge base following this reasoning process. Another strength of the cognitivist approach is that once an agent is implemented and tested, it can be deployed into the application domain. There are no resource or time requirements for training purposes. It also does not require training data sets to build knowledge representation.

In addition, the system's knowledge representation is human interpreted, hence the human would be able to determine the agent's capabilities and limitations. This allows the designer to validate the agent's capabilities in different contexts and be aware of what the agent can or cannot do. Knowing this in prior provides guidance about where the agent could or not be deployed.

In the case of self-driving cars modelled as a cognitivist system, the car is able to recognize and represent objects, such as other cars, trucks, bicycles, pedestrians and traffic-lights. The car agent is able to infer the action of each object and determine how to avoid obstacles that are already defined as solid objects. Because the cognitivist approach is a rule-based approach, designing self-driving cars in this approach gives the car the ability to infer traffic rules. The car is supplied with the rules used for driving on the road.

In addition, the designer is aware of the objects that the car agent recognizes. This information defines where and how the car can drive. For example, the designer would be able to define that the car drives in rural areas and not in crowded cities. Or whether it perceive trains and trams and can avoid their collisions.

Weaknesses of the cognitivist paradigm

Although cognitivism may sound as have solved the entire problem with autonomous driving, several shortcomings of the cognitivist approach to building artificial agents in general have been identified in the literature. As the representation models and their inference rules are mostly pre-defined by the designer, the agent will experience difficulties to continuously learn from the environment by itself. All the required knowledge and rules should be injected into the agent's memory at design-time therefore a self-driving car would not be able to perceive undefined objects. The agent may perceive these new objects inaccurately and act incorrectly, which would lead to hazardous outcomes. Thus, a possibly serious problem is that the agent's scope of knowledge is restricted to that of the designer's perspective rather than the agent's perspective [31]. For example, if the designer did not model people with wheelchairs, the car agent may not be able to recognize them as humans, therefore, the car agent's action is unexpected.

Moreover, adding object representations and rules after deployment needs re-designing the agent to recognize these new objects with new symbols and processing operations using these symbols. This is costly because every re-design instance requires re-implementing and re-testing the agent and re-evaluating the new extended modeling with respect to previous perceptions and actions. Therefore, more autonomous approaches are desirable for interactive systems.

2.1.2 Emergent paradigm

The second approach is the emergent paradigm, which is characterized by self-organized representations. In this approach, the knowledge and rules are not explicitly provided to the agent at design-time or development-time. Instead, the agent discovers and builds knowledge about the world by exploration and interaction with the environment. The agent performs this autonomously while experiencing surroundings, to learn and memorize knowledge. Hence, the agent is able to sense the world and enact actions to interact with the environment. There are three approaches under the emergent paradigm. First the connectionism, this was inspired from the low-level abstraction model of how neural networks mechanism processes information in human mind. Second, an enactivism, this approach is highly inspired by the human ability of self-development. It focuses on the ability of an autonomous agent to construct its own representation of the world in real-time by behaving autonomously while embedded in an environment. It senses the world, perceives it, and interacts with it to develop the required knowledge and to lean from the experience. Then the agent builds an ability to retrieve this knowledge when needed.

An emergent system is based on self-organization that enables an agent to learn from the environment by perception and interaction [48]. In this way, the agent can learn how to react in situations without prior explicit knowledge of the world and rules. Hence, the agent is not biased or limited by the design-time knowledge. This creates the opportunity for the agent to develop the required knowledge about the environment and perceive the knowledge from its perspective.

Connectionist approach

The connectionist or the artificial neural networks approach is based on a neuroscience inspiration in human brain, where a network of cells work together to perform a cognition [37].

The network consists of an input layer with a set of different input units, hidden layers with one or more layers of computational cells and the final layer is the output layer that determines the behavior of the agent. The network is constructed in a topology that represents how cells are connected and how weights that represents strength between cells. Cells are connected through different weighted links, these weights are responsible for calculating the final output of the network. The traditional neural network is trained as a supervised learning approach. The network is given a data set of inputs and the real output. According to the network weights, the network output tries to fit the expected output. The weights are then adjusted according to the differences between the network output and the expected output. The process is repeated with different data sets until the network output matches the expected output as much as possible.

The field of neural networks goes further into deep-learning where multilayer networks are used for complex tasks. The network learns the representation of the data within multiple levels of abstraction. These networks are widely used in image, audio, text and video recognition [16]. The connections between layers are more sophisticated than those found in traditional neural networks.

Most of pattern recognition techniques use a supervised learning approach. The network is given the input of an image for example, and the expected output or action and train the network to perform the same, such as image recognition approaches in Convolutional Neural Networks [15]. Another type of deep learning is Recurrent Neural Networks [7] where the sequence of an input are not independent but this sequence and time affect the learning process. In RNN the network looks at the pattern of the sequence in time and how they are related to each other.

In addition, deep learning can also be used for reinforcement learning. The network does not get the correct answer as feedback as data sets during the training phase. The training system defines what is right and what is wrong in a given situation according to its interaction in this situation. The environment gives the network the input and then the network responds with an action. The environment sends back a positive signal as a reward when the agent performs correctly or a negative signal as a failure. The difference between what the network output and the environment feedback is used as an error. The network is trained to minimize this error by repeatedly observe the state, take action and observe the reward. By doing this interaction, the RL agent creates its own data set of states and action online while interacting with the environment instead of fully provide it the full data set beforehand as in traditional supervised learning. More details is discussed in the next section.

For self-driving cars as interactive systems, supervised learning has been used for end-to-end training of a driving agent. In ALVINN (Autonomous Land Vehicle In a Neural Network) [35], a three layers neural network was used for

autonomous car-learning. The network gets camera images with laser sensors as inputs and produces the vehicle direction control. The network was trained and tested in a simulation environment. Another related work was training a Convolutional neural network for real car driving by NVIDIA [2]. Real driving-data was collected by expert drivers. The data contained both image inputs from several cameras along with the corresponding driving actions of the steering wheel. A deep neural network was trained using this data and then the network was deployed to a real car control environment. The car performance showed that the trained network in the car was able to drive correctly real roads.

A self-driving car application with a connectionist approach could be modeled as a deep learning neural network in a reinforcement learning. The network gets image data inputs and is trained in trial and error environments until it learns to drive correctly. The car needs to be situated in different conditions and learn to generalize learning situations so that it will not be limited to the given situations. Also, being able to learn by interaction allows the car to learn things that cannot be hard-coded or inferred by rules and can only be leant by experience such as driving in different weather conditions, when the road is icy and slippery or dry and stable, as well as if the weather is foggy and vision is unclear. In such driving environments, the car goes through unpredicted situations that the car needs to be able to deal with.

Enactive approach

The enactivism is a branch of the emergent systems paradigm, which means that it follows its self-organizing processes, while the agent interacts with the environment. However, the enactive approach goes further by highly emphasizing the embodiment of the agent where the shape and structure of the agent's body affects its perception and action [32].

Vernon [48] distinguished between five main key elements in designing an enactive system; autonomous, embodiment, emergence, experience and sense-making. First, autonomy relates to the ability of the agent to act and interact in an environment without being controlled by another agent. Larger degree of autonomy would allow a cognitive system to explore the environment and learn by experience. Embodiment makes the body engaged in the process of learning. It identifies how the agent is situated in the environment, what capabilities it has to interact with the world and how the agent views this world. Then, "emergence" occurs when the agent interacts with the environment and new situations emerges during this interaction. This creates the experience for the agent to acquire knowledge and memorize it. Finally, sense making identifies how the agent perceives the world and how it makes sense of the surrounding world.

Strength of the emergent paradigm

The emergence system is based on the self-organization which make a system able to learn from the environment by perception and interaction. The agent can learn how to react in situations without feeding it with the required knowledge and rules directly. Hence, the agent is not biased or limited to the knowledge of the designer. This opens the opportunity for the agent to develop the required knowledge about the environment and perceive the knowledge from

its perspective and based on embodiment that it has.

A self-driving car application with connectionist approach could be modeled as a deep learning neural network in reinforcement learning. The network gets image data input and be trained in trial and error environment until it learns to drive correctly. The car needs to be situated in different conditions and learn to generalize the learning so that it will not be limited to the given situations. Also, being able to learn by interaction allows the car to learn things that are not hard coded or inferred by rules and can be learned by experience such as driving in different weather conditions when the road is icy and slippery or dry and stable. As well as if the weather is foggy and vision is unclear. In such driving environment, the situation has different probabilities that the car needs to explore and learn.

Additionally, the emergence paradigm considers real world factors such as the agent being autonomous and embodied which is a crucial factor in self-driving cars because the body of the car and its sense making, plays big role in the car's learning. For example, the self-driving car's perception is highly affected by what kind of sensors it has.

Weaknesses of the emergent paradigm

On the other hand, the emergent paradigm training is a crucial step in developing an agent. The emergent system learns by trial and error which makes it dangerous to be trained in real roads where humans and other cars are driving. Besides, crashing the car for the purpose of training is not reasonable. Therefore, another system is needed such as a simulator to let the self-driving car live the driving experience. Because the agent needs high amount of training to be able to learn different situations in the situated environment, the simulator needs to be able to provide the self-driving car with these situations and offer a real world alike environment which is another challenging software artifact because the simulation should be as close as possible to the real situations that the agent would face. Otherwise, the car may not be trained properly and it may also react incorrectly in new situations that the car have not experienced before. Training a complicated driving agent network needs long time and high resources for training that may go for days or months of training.

Another difficulty is that the network represents the knowledge in distributed model which together froms the representation of all the knowledge. The way that the agent makes decisions in, is not a human interpreted which makes it difficult for human to know what has the agent learned and what it has not. This may become a danger if the designer assumes that the car learned to act in a situation but in fact in hadn't.

One of the drawbacks in connectionist is the overfitting problem which is the situation when the network learns the data set in the learning environment very well but it is not able to generalize the learned data [18]. In other words, the network is very tuned to the data set that it has trained on and is not able to act properly when it receives unfamiliar input or input with noise. A self-driving car that has overfitting problem may get the same input image of a turn road with some noise in the image and it may not be able to make the turn correctly.

2.2 Reinforcement Learning (RL)

As mentioned in the previous subsection, the cognitive paradigm that allows the agent to learn by interaction with the environment is the emergent paradigm. The approach in emergent paradigm that learns from previous experiences and sense-making is the enactive approach. One method of developing an enactive system is reinforcement learning.

Reinforcement learning (RL) is learning what to do as an enactive system. How to act in particular situation. The history of RL goes back to the psychological behavior perspective [13]. Human learns from doing sequence of actions and observe the consequence of these actions. Humans tend to repeat actions that bring positive feeling and avoid actions that bring negative consequences [12]. RL is based on trial and error interaction in an environment to find the highest positive consequences.

In machine learning, pleasure and pain are quantified into numbers as reward signals. The goal of the agent is to discover the actions resulting in the highest value which is the highest accumulated rewards over a span of time. The agent first explores the environment by trying different actions in different situations and then it observes the consequence states and rewards, as shown in Figure 1. When the agent executes an action in a given state, the agent transfers into a new state and receives a reward for taking this action. However, since in many cases high rewards are delayed and not immediately obtained, the agent needs to explore sequence of actions until it attains the high *accumulated* rewards. RL requires successive trials and errors besides searching for delayed rewards to learn the optimal actions in different states. Central to the RL is the reward function because it determines how good the action in the given state is [42].

Unlike supervised learning, described in Section 2.1.2, that is based on statistical pattern recognition of given samples, RL evaluates the action it takes and then builds its learning accordingly. RL can be integrated with supervised learning to gain both the rewards and the pattern of experiences that it has obtained from interacting with the environment where time is an important factor in learning.

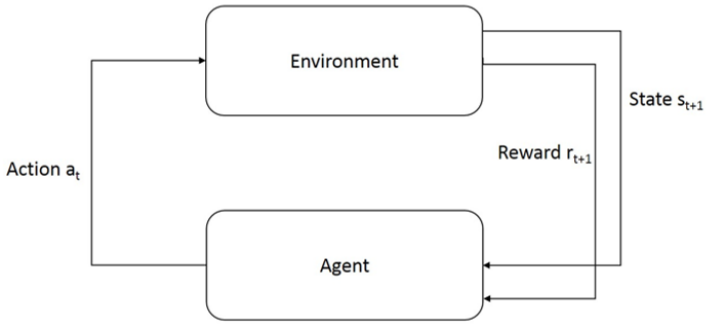

Figure 1: Reinforcement learning conceptual model

2.2.1 RL components

The following subsection is based on Sutton and Barto reference unless stated else [42]. RL is defined by the following sub elements:

A state s: represents the conditions that the agent can be in. The state in an environment is the information of the agent and the environment available for the agent to make a decision. A Markov state that holds all the needed information, is what the agent needs to take a Markov Decision.

The Markov property: A state that holds all the relevant information is called to have the Markov property. The Markov state summarizes everything important so that the agent doesn't need to remember the previous states. The current Markov state is all what the agent needs to make a decision. In other words, how the agent reached this situation doesn't matter in the Markov property. What only matters is the current state. If any old information is required, the Markov state should reflect this information in the current state without the need to memorize the sequence of states and actions that led to this state [36].

A process that satisfies the Markov property is called a Markov Decision Process (MDP). MDP is defined by: states of the environment s, set of available actions a, transition probability (the probability of moving from s_i to s_j when taking action a, the reward function R when taking action a and environment goes from s_i to s_j, reduced over time by the discount factor γ. An action a: is the set of all possible decisions that the agent can take while being in a state s.

A policy $\pi(s)$: is the decision making mechanism of selecting an action. The policy maps between the perceived state and the action to be selected. The policy defines how the agent acts in a particular state. The policy can be in a different forms. It can be stochastic where the agent randomly selects an action. In can also be a look up table that maps each state into an action. A function or a computational mechanism is also used for policy. The policy determines the behavior of the agent in the environment. During the training, the goal of the agent is to find the optimal policy π^*. The optimal policy is the strategy that returns the highest value. This is done by successively evaluate its current policy and improve it toward the optimal policy. Following the optimal policy is following the highest value at each state.

A reward function $R(s, a, s')$: the reward function determines the immediate reward that the agent gets for taking the selected action in the given state. At each step, the agent gets a reward for the action it takes and calculates the value accordingly. The reward function shapes what the agent needs to learn. Shaping the reward function is a challenging task in RL [42].

A value function $V(s)$: represents the accumulated rewards over time while going from one state to another, starting from the current state. It is important for the agent to consider the long run value. A state may yield low value in the short term but followed by high accumulated reward in the long run. Conversely, a state may yield a high reward in the short term and conceals low rewards afterward. The agent should seek the highest value not just the highest reward. When the agent is initialized in the environment the values of

each state are not known directly but are calculated and estimated during the training process. Because closer values are more valuable than away rewards, the value function diminishes the later values by a discount factor of $\gamma \in [0, 1)$. The expected rewards $V^{\pi}(s)$ of being at state s is the accumulated rewards of each state is goes into when following the policy π as shown in equation (1).

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s\right\} \tag{1}$$

The discount factor γ determines how far the agent should consider. Lower $\gamma \approx 0$ causes selecting actions that leads to close immediate rewards. While higher $\gamma \approx 1$, results in taking actions that considers the far future rewards.

This equation can be formed in terms of relation between values over time, which is called the Bellman equation, equation (2)

$$V^{\pi}(s) = \max_{a} E\{r_{t+1} + \gamma V^{\pi}(s_{s+1}) | s_t = s, a_t = a\} \tag{2}$$

$$V^{\pi} = \max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right] \tag{3}$$

The Bellman equation can be formulated as the sum of the probabilities for all the possible rewards the agent can get when moving from state s to all the possible states s' when taking action a [42], equation(3).

An Action-value function Q^{π}: the value of taking action a at state s is defined as the action-value when following policy π.

$$Q^{\pi}(s, a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \tag{4}$$

$$Q^{\pi}(s, a) = E\left\{r_{t+1} + \gamma \max_{a'} Q^{\pi}(s_{s+1}, a') | s_t = s, a_t = a\right\} \tag{5}$$

$$Q^{\pi} = \max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right] \tag{6}$$

Both the V^{π} and the Q^{π} are estimated from interacting with the environment while training.

A model: is the transition function that determines the next state when taking a particular action at a particular state. The model mimics the rules of the environment.

An episode: is the sequence of state, action, reward and next state from the initial state until a termination state. A terminal state is when the episode ends. It could either ends when the agent reaches the goal and achieves the task or when it falls into a critical failure where no return point is possible. RL may sometimes not have a bounded episode but a continuous run that has no termination state. In this case, the agent keeps running until it is manually stopped.

2.2.2 Methods for solving the RL

To solve a reinforcement learning problem, the agent should be able to find the optimal policy that returns the maximum rewards over the long run. By finding the optimal policy, the agent is able to select actions from the initial state till the end goal through the states that returns the highest value in each action. Worth reminding is that the highest value at each action is not the highest reward at each action but the sequence that results in highest accumulated rewards. It is not a straight forward problem to solve the Bellman equation to find the optimal policy because it is a non-linear function. Also, there could be more than one optimal policy for one problem.

There are different methods for solving the reinforcement learning to find the optimal policy, such as dynamic programming, Monte Carlo method, and temporal-difference learning.

Dynamic Programming (DP):

In DP, the problem is broken into sub problems and then each problem is solved recursively. RL and the Bellman equation are based on the recursion concept. DP is a model based technique, which means that the agent knows the transition function and the reward function in advance. The agent uses the the Bellman equation (2,5) recursively to solve the RL problem. In Bellman equation each value is the current reward plus the value of the next state. This iteration of value calculation continues until the episode ends.

To use DP to solve RL problems, perfect knowledge of the model of the environment is required, which means that the transition from state s to s' if taking the action a is known. If there are different possibilities of transitions from one state to multiple other states taking an action a, the probability of these transitions should be known to the agent in advance. The value of a state is the maximum of the reward and the summation of the discounted probabilities of the values for each possible state the agent has a probability to transit to.

$$V(S) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s') \right) \tag{7}$$

To solve the RL problem in DP method, we recursively iterate through the states to find the optimal policy. There are two main algorithms to solve the DP:

Value Iteration:

The agent iterates through the state to calculate and store the value for each state. Then, according to these values, the agent can get the optimal value by selecting the action that takes it into this state. This process iterates through all possible states and all possible actions in each state until the value converges to an optimal value [3].

Policy Iteration:

Instead of calculating the values for each action in each state and then evaluate the optimal policy from these values, the policy iteration calculates the

policy immediately and evaluates the policy without the need of calculating the value [43].

Monte Carlo method (MC):

One drawback of the Dynamic programming is that it is a model based technique. The agent needs a complete knowledge about the model of the world and the transition function and the probability of these transitions beforehand. It uses this knowledge to solve the optimal policy. If the model of the world is not given, the dynamic programming will not solve the optimal policy [42].

When the transition function and the reward function are unknown for the agent in advance, a model free technique is more suitable to find the optimal policy. In this approach, the agent needs to know its current state, the available actions and the experienced rewards that it gained while exploring the environment.

Monte Carlo methods is a model free technique that uses random sampling to create a conception of the real model [23]. The agent don't know the transition function or the reward function beforehand. It samples the environment to compute the transition function and the reward function by taking random actions in different states and observe the transition and reward in real time. MC doesn't need a prior knowledge about the dynamic of the environment to find the optimal policy. In this method, knowledge is accumulated online or during exploring the environment until the episode is terminated. It is an important assumption that the task is episodic. Which means that the agent should reach an end state of the episode, whether it successfully reaches a goal or it falls into failure state [50].

The agent accumulates the sequence of the visited states and action pairs and returns (G) accumulated discounted rewards from the first state up to the terminal state. At the end of each episode, the policy is updated as the value of each visited state is calculated by averaging the values computed at each state [42].

The MC method is composed of two main phases; the policy evaluation and the control. First, in policy evaluation, the agent follows a given policy to be evaluated. Starting from an initial state, the agent takes actions according to the given policy until it reaches a terminal state. Each sample is called a visited state, at which the agent observes the return value for this state and append it in a return list. A return list is a list for all states and their return values. The value of the current state $V(s)$ is the average of the the return list [42].

In most cases, knowing the state value $V(s)$ is not enough. The policy needs to be evaluated based on what the agent does at this state. Alternatively, the agent needs to know only the state-action value which is the quality value $Q(s, a)$ that defines the quality of taking action a at state s. In this case, the return value is not calculated for the state but for the selected action at that state.

Secondly, learning phase, where the agent improves and optimizes the policy to maximize the expected reward at any state. There are two types of policy optimization; on-line or off-line. On-line policy is improving the same policy that has been evaluated immediately at the end of the episode. Off-line policy, allows the agent to improve one policy while evaluating another policy. This

is done by using the policy greedily to generate the return values and find the difference from the expected values to improve the policy. However, relying only on the policy to evaluate and improve the policy would prevent the agent from exploring states and actions that the policy doesn't favor. This leads to an exploration and exploitation trade-off problem.

Temporal-Difference learning (TD):

TD learning is another model free method. It is a novel approach is RL because it combines features from both MC and DP [42]. It is similar to MC in sampling the world for incomplete knowledge of the world. MC calculates the state value (or the state-action pair) and updates the policy by the end of the episode. However, TD learning doesn't wait until the end of the episode to get the collected rewards to improve the policy. This means that TD learning can be used for both episodic and continuous problems. The TD learning updates the state value (or state-action pair) at each step based on the difference between the original and the estimated state (or state-action pair) value. The original value is the observed reward plus the discounted value of the next state. The difference between the existing value and the estimated value is the error of the estimation. The state (or state-action pair) value is then adjusted to minimize this error, equation (8), where α is the learning rate.

$$V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{8}$$

Sarsa: On-Policy TD Control

Sarsa is a TD learning method that uses the state-action pair value for the policy iteration. It calculates and updates the state-action pair value based on the current policy that the agent is following, eqn (9). In other words, the value of the next state is the value received after taking the action based on the current policy.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \tag{9}$$

Q-Learning: Off-Policy TD control:

Q-Learning is a breakthrough method that has gained a lot of interests in RL recently. The state action value is independent of the policy.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_a Q(s',a') - Q(s,a)] \tag{10}$$

In Q-Learning, the next state value is the maximum state action value independent of which action yields this value. It has been proven that this allows faster convergence [42]. More details are presented in subsection 2.2.4.

2.2.3 Exploration and exploitation dilemma

A model free technique needs to explore the environment by selecting random actions at different states to observe the transition into a new state and the reward it gets from this transition. After it creates a conceptual model of transition and rewards from these experiences, it can rely on the gained experiences

to select sequence of actions that returns the highest value. It is difficult to state when it has finished exploration and that the estimated transition and reward functions are close to the real world functions.

This raises the trade-off of exploration and exploitation time. In other words, the agent needs a stochastic policy to explore new states while it also needs to utilize the gained experience to progress. It is important to find a balance between the exploration and exploitation. In model free technique, this balance is attained by the exploration rate ϵ. If it is high, $\epsilon \approx 1$, the agent has higher probability to explore by following a stochastic policy. If the exploration rate is low $\epsilon \approx 0$, the agent has higher probability to exploit by using its deterministic policy. At the beginning of the training, the value of the exploration rate $\epsilon = 1$ and during the training it decreases gradually. By the end of the training, the exploration rate ϵ is small enough to follow a deterministic policy while keeping a small probability for exploration to ensure that it doesn't stuck in a local maxima.

2.2.4 Deep reinforcement learning

RL is mainly used in discrete state space where states and transitions between these states are clearly defined [49]. However, it has started to flourish in continuous state space as well. For continuous state space, discretizing the space into tabular states and actions makes it inefficient because of the enormous number of states. Instead, a function approximation such as neural network or deep learning is required for such continuous spaces.

Q-Learning is a breakthrough method that has gained a lot of interests in RL recently. In this method, the calculation of the state-action function, equation (11), is independent of the policy. In other words, the agent is able to calculate the value of the next state without calculating the policy. The policy (π) in DQL is selecting the action that outputs the maximum Q value, equation (12).

$$Q^{\pi}(s,a) = E\left\{r_{t+1} + \gamma \max_{a'} Q^{\pi}(s_{s+1}, a') | s_t = s, a_t = a\right\} \tag{11}$$

$$a_t = argmax_a Q(s_t, a, \theta) \tag{12}$$

Where r_t is the received reward, s_{t+1} is the next state, a' is the action that would be taken in the next state and θ is the network hyper-parameter.

For continuous state space, discretizing the space into tabular states and actions makes it inefficient because of the enormous number of states. Instead, a function approximation such as deep learning is required for such continuous spaces. Using deep learning as function approximation of the state-action function is called Deep Q-Learning (DQL).

The technology of deep learning has shown great breakthroughs in pattern recognition, classification and regression in different fields such as speech [6] and vision [15] with a massive amount of data. Deep networks are trained by giving both the raw data as an input and the correct outputs as learning samples.

The network updates its weights to minimize the loss on a stochastic gradient descent [39].

Interactive learning is used to approximate the value function for the given state at each step and allows for interactive learning minimizing the loss. The loss is the mean square error between the predicted $Q(s_i, a_i)$ value and the actual value y_i, equation (13). The goal is to reduce the network loss for the weights (Θ). The network loss $L(\Theta)$ is computed as in equation (14).

$$y_i = \begin{cases} r_t, & \text{terminal state} \\ r_t + \gamma max_{a'} Q(s_{t+1}, a, \theta_{i-1}), & \text{non terminal state} \end{cases} \tag{13}$$

$$L_i(\theta_i) = \mathbb{E}_{s,a}\left[(y_i - Q(s,a))^2\right] \tag{14}$$

Where r_t is the observed reward of the current step.

When applying deep learning to RL, several issues should be taken into considerations. First, the network requires multiple iterations to update the gradient decent for the given sample. Second, the similarities between states in close time steps when learning on-policy for the current sample at each step, leads to a high correlation in the data sample. In deep learning, the training data set needs to have variability in samples to break this correlation. For example, if the sequence of samples focus on action a_1, the network will update its weights to fit action a_1. Introducing action a_2 afterwards increases the variance of the update significantly and leads to catastrophic learning [47]. Third, although RL has shown remarkable results in environments that it was trained on, the agents lacks the ability to generalize its knowledge. This is because of the limited number of simulation scenarios it is trained on.

2.3 Imagination Based Learning

Inspired from the human imagination [10][11], an agent can imaginary practice old experiences to improve the performance in when facing the situation in the real world. The concept of artificial agent imagination is not totally new. Imagination has been used in artificial agents as reactivating sensory and action motors internally without having an external stimuli [44]. In this work, imagination is the virtual environment where the agent generates various scenarios to experiment and learn new skills.

We categorize imagination into three purposes; training, planning and creation. These purposes are seen as complementary to each others and they are used to improve the overall learning process.

Mahadevan [19], argues that imagination science is the future of AI. Data science is based on statistical calculation for finding correlations in the given data. Machine learning of image recognition and object recognition does not go much beyond labeling items. He claims that imagination science goes beyond data science because it allows producing novel samples.

2.3.1 Imagination for training

In this approach, the agent first experiences the world by perceiving states and perform actions. During its training, the agent also imagines performing the same actions at the same states that have already been experienced. By repeating the experience and the same states, the agent learning of these states is increased because the agent doesn't only experience it once in the real world but several of times in imagination.

The Dyna architecture [41] in one of the earliest proposed word for this approach. In this architecture, the agent accumulates the previous experiences of the state, action, reward and next state information over time. The accumulated experience is then used as imagination resource to predict the future value function while taking the same action. The predicted value is used to update the state-action tabular for discrete states, or train a neural network for the continuous states. In this imagination, the same experience is imagined multiple times to speed up the training.

Mnih et. al. [24, 25] used the buffer technique which is mainly inspired from the Dyna architecture, for training an agent to play Atari games. They showed that a system is able to over-perform humans in games where the agent is able to get all the information needed to take an action or when the reward is not delayed from the action. This learning mechanism was used to train the famous AlphaGo [40] that over-performed the world Champion in the game Go. After this, RL has rapidly grown in continuous world environments. Lillicrap et al. [17] designed a reinforcement model with deep learning. The model is a deep network of convolution layers followed by filter layers and an output layer. Although, the model can learn different games with the same configurations and hyper-parameters, the model can't transfer its skills from one game to another.

For games, such as Montezuma's Revenge, where the rewards are very sparse and delayed for long time, a reactive agent does not learn the game. Such games require complex planning to know which action to take earlier that would benefits later.

2.3.2 Imagination for planning

Inspired by the Dyna architecture, Baldassarre [1] proposed a planning mechanism for reinforcement learning. Instead of using planning to speed up the training, Baldassarre uses the planning to implement trackable planning. In his presented work, the agent is able to reach a goal that it has not reached before, if it knows the goal state and the model of the world. To even speed up the training, a bidirectional planning is proposed. As the forward planning predicts the sequence of states and actions starting from the current state up to the goal, the backward planning predicts from the goal to the current state. This approach was used in a dynamic programming approach were the agent knows the model of the world and can predict the probability of performing the action. These planning and predictions are limited to environments that the agent has a pre-knowledge of the world model.

Another planning system was an Imagination-based planner by DeepMind [29] where they used the model to construct a plan. The work briefly describes how the agent's controller chooses either to imagine or to take and action in the real world.

2.3.3 Imagination for recreation

RL has shown remarkable results in environments that it was trained on. However, with limited number of simulation scenarios, the agents lacks the ability to generalize its knowledge. In the imagination for recreation approach, the agent uses sensed data of the world to create new unseen scenarios. One method for recreation using Generative Adversarial Network (GAN) [5] that learns to generate new unseen data from a given data. This technique was used by Santana [38] collected real world video data and trained a neural network to generate similar video scenarios and then run the RL driving agent in it. Initial results generated very similar image simulation compared to the real data.

Similarly, Ha and Schmidhuber [9] illustrate unsupervised learning for car agent in OpenAI simulation environment. First, the car explores the environment to learn the model of the world. Next, it learns to generate new episode simulations in a sequence of images similar to the explored ones with slight differences that the car has not experienced before. They used the OpenAI Atari tool as the simulation environment, where a car agent attempts to learn to drive in these new scenarios in a RL mechanism. The work showed that the agent performed higher when trained in the generated unseen environment and then injected back to the real environment. However only relying on similar images is insufficient for generating new simulation scenarios. It is difficult to associate a reward function to a video generated scenario. Besides, this technique lacks the availability of the physical dimension of the environment. Generating simulated world using similar images does not imply embedding the physics of this world in it

2.3.4 Learning for real world deployment

Gu et.al. [8] describes a robot arm trained and tested in a simulation to open a door with a door handle. The research increased the speed of learning by collecting learning data from multiple robots simultaneously and store the data in the replay buffer. This technique showed higher efficiency in learning in shorter time. In addition, the replay technique contributes significantly in reducing the training time in real time simulation in a configuration that the agent can't rely on the computation speed used in deep learning. The work presents a promising direction in applying reinforcement learning for real physical life.

3 Research Problem

This chapter is organized as following; Section 3.1 describes the problem and the scope of this research. Based on the problem description Section 3.2 lists the research questions that this research aims to solve.

3.1 Problem Description

An interactive agent needs to experience different situations in the environment to learn how to act in most of the scenarios it faces. Most current learning mechanisms requires the availability of all the training data or the environment scenarios beforehand to train the agent all at once. However, in real world applications it is difficult or impossible to get all the tasks beforehand. The agent needs to be able to progressively learn one skill after the other whenever new situations and challenges arises. Besides, even when the agent encounters the situation in the real world, it may be difficult to try different experiences in the real world, for example, when the self-driving car encounters a sudden break from the car in front. The agent needs an environment to experiment new knowledge and train in a safe environment. An imagination approach can be used where the agent generates different scenarios to practice the different situations. The scenario generation requires the agent to be able to re-create situations from previously seen ones. The artificial agent may have different ways of re-creation (Section 2.3.3) .

This research focuses on increasing the learning performance of interactive artificial agents.A particular focus of the future work will be the generation of training data for RL by means of imagination mechanisms. The imagined scenarios are used to generate the required data for the agent to learn from. In order to learn from these scenarios, the agent should first be able to continuously develop and build upon old data. Secondly, the scenarios should be build upon a ground foundation and compatible with the learning progress. Thirdly, the learning process should be applicable for the real world scenario.

3.2 Research Questions

The research questions are divided into a primary research question and a secondary question.

RQ1. The primary question of this research is **How an agent that interacts with an environment can incrementally build knowledge and skills from imaginary scenarios?** To answer this question the following objectives need to be achieved:

Obj.1. What are the artificial cognitive approaches for training an interactive learning agent?

This question surveys the different artificial cognitive system approaches and models the suitable approach for training an interactive system. The goal of this question is to adopt the most appropriate

approach for training the artificial cognitive system in an interactive environment.

Obj.2. To what extent do the imagination techniques affect the learning?

This question studies the feasibility of the minimum effect of imagination on training compared to no imagination. The aim of this question is to validate the concept of learning by imagination in artificial cognitive systems.

Obj.3. How can an agent continuously learn new skills

This question tries to find out a learning mechanism that can progressively develop and learn new skills. This question aims to formulate a mechanism to allow the agent continuously improve and build upon previous knowledge when the training data is not available all at once.

Obj.4. How to generate synthetic training data for the agent in a systematic mechanism.

This question studies the different mechanism of generating imaginary scenarios such that the generated episode is systematically and automatically designed.

Obj.5. How can features be extracted for scenario generation:

1. Based on pre-given features.
2. Based on all features extracted from the collected data in the real world.
3. Based on selected features extracted from the collected data in the real world.

This question aims to define the features for creating the imagination episode. This is divided into three approaches. The first approach is using features determined manually by the programmer according to a pre-defined goal.

The second approach automatically extracts features from the collected data. In this approach, all extracted features will have equal importance and priority.

In the third approach, some features will be selected as higher priority than others. The progress of the learning agent determines the selection criteria.

RQ2. How is it convert the extracted features from the real world data sensory data into the imaginary generated features?

This question extends objective **Obj.5.** This question defines the representation of the sensory and motor data from the real world. The aim of this question is to have a bidirectional mapping between the real world and the imaginary world. This is a secondary question that is added for research improvement if possible.

3.3 Limitations

This work takes self-driving cars as an application area. The simulation used for imagination OpenDS open source. The data collection, imagination generation and learning mechanism are implemented for a driving car. For limited access to real world deployment into real car, an RC car is used for data collection and evaluation deployment. The RC car runs on a pre-defined and controlled road environment.

4 Research Method

This chapter describes the research method used to answer the research questions presented in the previous chapter.

4.1 Research Strategy

The research strategy for this research is Design Science [28]. This strategy aims in building new more effective IT artifacts that replaces the exciting technology or overcomes the limitations of the available tools. The design science strategy is suitable for this research because the focus of the research is to study and develop a method for an artificial interactive agent that continuously learn from imagined environments. The design science strategy facilitates the process of studying and building such an IT artifact.

4.1.1 Research output

The output of the research is a model for an imagination based progressive learning mechanism. A model is an abstract representation of how things are and the relationship between entities [22]. The model of the progressive imaginary learning describes the entities involved in the learning through imaginary scenarios. The model illustrates the relation between different components. The expected results and outcomes of this research is the following:

1. A survey of different cognitive system approaches for learning new behaviours, This will consist of a detailed literature review of previous work that compares and analyzes the strengths and weaknesses of each approach and theory.

2. A mechanism for modeling an incremental learning approach were the agent is able to append new learning to previous ones. This is an implementation for a proof of concept for the model. It includes the design and development to construct a theory for a progressive learning agent.

3. An encoder/ decoder model that maps between the real world environment and the simulation representation. This question deals with the data representation of collected sensory data from the real world that consists of both symbolic and non-symbolic data and semantically convert it into the simulation symbolic representation. The agent should be able to match the knowledge it gains from the simulation training and perceive the real world data even if they don't directly correspond.

4.1.2 Research activities

In designing and creating the main research activities are build and evaluation [20]. Build is constructing the artifact to proof that it can be constructed. Then evaluate is to determine the performance of this artifact.

Building and evaluating a research artifact in Design Science Research Methodology (DSRM) includes six activities as described in Figure 2 [30]. First, identifying the problem and the motivation behind the work. In this activity, the researcher justifies why the research is important and what is the benefit outcome of it.

A second activity is then needed, which is defining the objective for a solution. According to the problem identification, the researcher identifies the objectives of the research and what is the outcome that the researcher aims to achieve and subdividing it into concrete activities. Next, design and development activity which is building the artifact. The design is the architecture of the artifact components and the development is the functionality implementation that leads to the solution objectives.

After developing the artifact, the researcher needs to demonstrate that the artifact solves the intended problem and it meets its objectives. The demonstration activity proofs that the artifact works, however another activity is required to measure how well the artifact solves the problem. This is achieved in the evaluation activity. The artifact needs to be evaluated in a systematic way in a quantifiable measures of the system. Finally, after the researcher come up with a contribution, the researcher needs to come up with a generalized conclusion and communicate with other researchers in a form of publication.

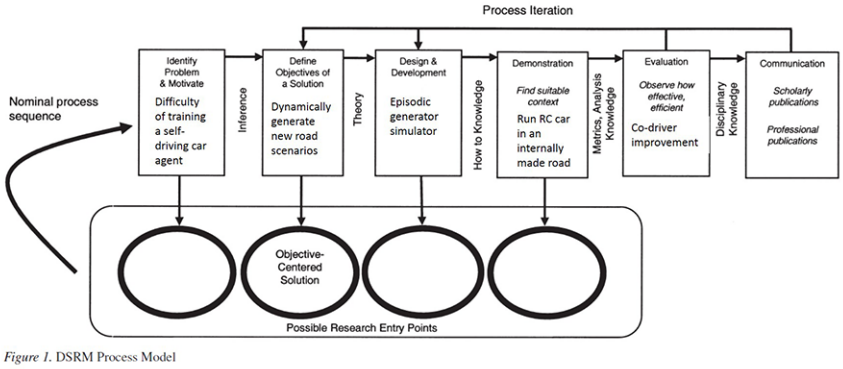

Figure 1. DSRM Process Model

Figure 2: DSRM Process Model

4.1.3 Philosophical paradigm

This research falls under the positivist paradigm not the interpretivism. Although the model is inspired from bio-system of the brain, the main focus of the research is not the social aspect behind it which is what the interpretivism studies. The research aims to build a model of how the world looks like for a self-driving car. This modeling includes laws and rules such as the traffic rules in the road as well as the physics law of how the car performs while driving.

The research doesn't address the social context such as drivers' behaviors or the social impact of self-driving car. In addition, episodes generation of the road are based on the co-driver performance which is measured logically and mathematically.

4.2 Research Design

The research design consists of the data generation and the development of the artifact. This section describes the different sources of the data in this research and then illustrates the system architecture showing the different components.

4.2.1 Data generation

There are different data sources in this research. The sources includes the literature required to understand and study the field and the state of the art. In addition, the digital sources used for generating simulated data such as data collected from the agent. The literature documents include previous work in human cognition in a broad scope, more specifically in human imagination and dreams. The literature then focuses more in artificial cognition, imagination in robotics and machines. Self-driving cars machine learning approaches should also be studied.

Another important source of data is car accident statistics in Europe [51]. It shows the main causes of car accidents, such as weather and speed. The accident statistics will be analyzed to extract the main causes of car accidents to generate similar situations that the co-driver needs to learn. Analyzing these statistics allows the episodic generator to create road scenarios similar to the car accident scenarios so that the co-driver practices these scenarios for training. This data can qualitatively and quantitatively be examined to extract cause of accidents and inspiration for road scenarios to be imagined.

In addition, an important source of data is the driving agent log data. The driving agent logs are the detailed scenario/action data sets in every step. The log data can be collected either from the real world or from during the training phase.

The episodic generator needs this data set to generate new imagination roads. If it is a real world data collection, then it extract the features to generate the desired training roads. While if the collected data is a training log, the road generation is mainly done from analyzing the driving agent's performance, such as measuring the improvements, assessing the performance in different scenarios and then generating the new road with the extracted features. In this case the generated roads are either where the driving agent fails to perform or new complex roads that are harder to drive in.

4.2.2 System architecture

The system architecture consists of four main components: OpenDS, the physical simulation, in which the driving is executed, a middleware connector that

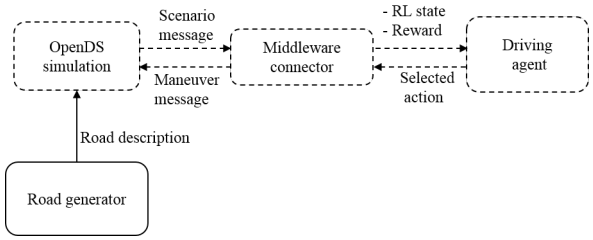

Figure 3: Episodic generator system architecture for self-driving car on OpenDS simulation

(a) 3D rendered Road

(b) Lane keeping parameters

Figure 4: OpenDS road environment

converts the simulation into a RL environment, the road generator that describes the road specifications used in the simulations, and the learning agent (see Fig. 3).

OpenDS Simulation

OpenDS is an open source driving simulator with a multi-threaded physics engine that allows in particular mesh-accurate collision shapes and enables the application of basic forces such as acceleration, friction, torque, gravity and centrifugal forces during simulation. The motivation for using OpenDS is two-fold: first, it provides a more complex simulator including a somewhat more complex physics simulation than usually used in previous experiments on imagination in vehicles [9]. Second, it provides a manner of creating episodic simulations that is viable with respect to complexity, although it is simplified with respect to biological counterparts.

Middleware Connector

The middleware connector is required to actually implement RL since OpenDS is just a simulator. We use this middleware connector to calculate the reward function (optimized for a lane keeping task) at each step. The reward function is crafted based on the road features and task goal to be achieved. The connector

also acts as a communication bridge between OpenDS and the driving agent (using a UDP connection with 20 messages per second).

Road Generator To create new road scenarios, including the imagined road scenarios, a "road generator"is used and currently a basic implementation exists that may be further developed. The generator is primarily used in experimental conditions that use new "imagined" roads as part of their learning. The road generator creates the road scenario used in the simulation. The road scenario (eg. road scenarios of number of lanes) is specified in an XML file that is converted into a road map and a 3D rendered road simulation in OpenDS.

Driving Agent

The driving agent incorporates both the architecture to control the car in the simulation and the means to learn how to control the car. The driving agent in this study is in the form of deep neural networks using Deep-Q-Learning. During training, the driving agent receives a representation of the road through the Middleware Connector and the state of the learning agent. The learning agent then selects the action from set of available actions (turning the steering wheel 0.05rad/s to the left, right, or maintaining its current position) The agent receives a reward that represents how good the chosen action is in the given state. The agent updates the network weights based on the obtained experience and continues with this process until it reaches a terminal state, which is arrived at either when the agent successfully reaches the end of the road, or when it leaves the road prematurely.

Real car The real car is used for collecting data and for evaluating the driving agent performance. In order to generate simulated data, the episodic generator needs extracted features to base the generation on. In order to evaluate the final learning capabilities of the driving agent, the driving agent needs to be tested on a real roads. After the driving agent has learned to drive in the simulation environment, it is deployed in the real car controller to select actions. The real car needs sensors and actuators similar to the simulation input while training. The goal of the real car is to demonstrate that the driving agent performance increased by learning from imagination. For the current research purposes, a simplified radio control (RC) car will be driven in a simple road-like environment to test driving abilities.

Feature Extractor The collected data may consist of both symbolic and non-symbolic data. The road generator uses symbolic data to formalize the road features. In order to generate roads based on the scenarios seen in the real world, a feature extractor is needed to convert non-symbolic data into symbolic data. This is a composed of a network with supervised learning holding the symbols for the relevant objects related to the driving scenario such as other cars, pedestrians, traffic light and road features. For each object, the relevant properties and behaviours are stored in a knowledge graph. Properties refer to static information about the object while the behaviour is the action over time. For example, the pedestrian has the proprieties of current location, speed and type (walking, biking, wheel char, adult or child). It also has the behaviour of crossing or walking by the side. Some objects may have properties but not behaviours such as the properties of the road lane.

Table 1: Research Plan: research activities and years divided into quarters

Activity	2017		2018				2019				2020				2021				2022	
	3/4	4/4	1/4	2/4	3/4	4/4	1/4	2/4	3/4	4/4	1/4	2/4	3/4	4/4	1/4	2/4	3/4	4/4	1/4	2/4
Literature review																				
Data Collection																				
Design																				
Development																				
Evaluation																				
Writing																				

5 Research Plan

The research plan consists of the major activities which are literature review, data collection, design and creation, evaluation and writing. An agile approach will be applied for these activities [21]. In other words, the activities are broken down into smaller chunks are carried in smaller group.

Literature review

The plan starts by a large and deep literature review in the field to define the research gap and learn theories, techniques from previous work. A literature survey of the different cognitive systems approaches and the different learning mechanism for interactive agent is conducted This process is carried through out the research to keep up to date with the state of the art of the field [**Obj.1.**].

Data collection

Next, the data collection activity as illustrated in 4.2.1 is used as a source for the design and development. The data collection goes through three phases in the research. At the first phases, related documents are collected and qualitatively analyzed to extract the requirements and features for the first design. The second phases includes the features collection for defining incremental progressive learning agent. The third phases of the data collection includes sensory data collection from the field to extract all the possible features for the road generation [**Obj.5.**].

Design and development

The process of the design and development activities follows the agile approach [21] which is based on a continuous evolving of the developed artifact. A repetitive cycles of design, development and evaluation is carried incrementally on top of the previous artifact.

In this work, the first design is done for the general architecture with simplified version of the components (the simulation, road generation, driving agent). The design includes how these components works as well as how they interface to communicate. The design is then developed. The development includes the implementation and the training process. Besides, the training data is collected (for the data collection phase). For the first phase, the goal is a base and foundation implementation for the research. In this phase the general effect of imagination is measured as a proof of concept [**Obj.2.**] with a pre-defined features [**Obj.5.1.**].

The second phase of the design focuses on the driving agent. The goal of this process is to design and develop a learning agent that can incrementally gain knowledge [**Obj.3.**]. The design includes the driving agent architecture. Which defines how the agent can build incremental knowledge. In this phase, all features are used regardless of its importance [**Obj.5.2.**].

The third phase focuses on the real world data collection, feature extraction and road generator. This phase uses the driving agent from the previous phase to learn from well structured generated scenarios. The design activity in this phase includes how the real world data are represented and stored [**RQ2.**]. It also defines the structure of the symbolic knowledge based were the extracted feature is then stored. The road generator is advanced to generate well structured roads based on critical situations [**Obj.4.** and **Obj.5.3.**].

Evaluation

The evaluation activity is carried out at each phase to imperially measure the improvement added to the learning agent for the designed artifact. At the first phase, the system is evaluated based on the effectiveness of different imagination techniques to no imagination. The second phase is evaluated by how much the driving agent is able to improve over time for the given features and build up knowledge on top of previous knowledge. The third phase is then evaluated on how critical the extracted features are and how these features contribute to the learning.

Writing

The writing activity goes at all phases. This phase includes documenting design, experiments and results. It also includes conference papers, journals and thesis writing.

Publications

This section lists the publications that have been published or submitted from from this research

Published:

Mahmoud, S., & Svensson, H. (2018). 'Self-driving cars learn by imagination. Swecog 2018', *the 14th Swecog conference.* Linköping.

Submitted under review:

Mahmoud, S., Svensson, H., & Thill, S. (2019). "Imagining" variations of a lane keeping task improves the efficiency during learning for an autonomous driving agent. *The 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019).* Macau.

Bibliography

[1] Gianluca Baldassarre. Forward and bidirectional planning based on reinforcement learning and neural networks in a simulated robot. In *Anticipatory behavior in adaptive learning systems*, pages 179–200. Springer, 2003.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, and Jiakai Zhang. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[3] Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

[4] Dayne Freitag. Greedy attribute selection. In *Machine Learning Proceedings 1994: Proceedings of the Eighth International Conference*, page 28. Morgan Kaufmann, 2017.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[7] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[8] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[9] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[10] Germund Hesslow. Conscious thought as simulation of behaviour and perception. *Trends in cognitive sciences*, 6(6):242–247, 2002.

[11] Germund Hesslow. The current status of the simulation theory of cognition. *Brain Research*, 1428:71–79, 2012.

[12] Clay B. Holroyd and Michael GH Coles. The neural basis of human error processing: reinforcement learning, dopamine, and the error-related negativity. *Psychological review*, 109(4):679, 2002.

[13] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[14] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[17] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[18] Charles X. Ling. Overfitting and generalization in learning discrete patterns. *Neurocomputing*, 8(3):341–347, 1995.

[19] Sridhar Mahadevan. Imagination machines: A new challenge for artificial intelligence. *AAAI Conference*, 2018.

[20] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decision support systems*, 15(4):251–266, 1995.

[21] Robert C Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[22] James L. McClelland. The place of modeling in cognitive science. *Topics in Cognitive Science*, 1(1):11–38, 2009.

[23] N. Metropolis. Monte carlo method. *From Cardinals to Chaos: Reflection on the Life and Legacy of Stanislaw Ulam*, page 125, 1989.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, and Georg Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[26] Hans-Hellmut Nagel. Steps toward a cognitive vision system. *AI Magazine*, 25(2):31, 2004.

[27] Allen Newell, John Calman Shaw, and Herbert A Simon. Elements of a theory of human problem solving. *Psychological review*, 65(3):151, 1958.

[28] Briony J. Oates. *Researching Information Systems and Computing*. SAGE. Google-Books-ID: VyYmkaTtRKcC.

[29] Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170*, 2017.

[30] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[31] Rolf Pfeifer. Cognition—perspectives from autonomous agents. *Robotics and Autonomous Systems*, 15(1-2):47–70, 1995.

[32] Rolf Pfeifer and Josh Bongard. *How the body shapes the way we think: a new view of intelligence*. MIT press, 2006.

[33] Shrijit Pillai, Ishtiak Zaman, and Tejas Shah. *Self-Driving Car Simulator*. 2015.

[34] Aske Plaat, Jonathan Schaeffer, Wim Pijls, and Arie de Bruin. Best-first and depth-first minimax search in practice. *arXiv preprint arXiv:1505.01603*, 2015.

[35] Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[36] Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[37] David E Rumelhart, James L McClelland, PDP Research Group, et al. *Parallel distributed processing*, volume 1. MIT press Cambridge, 1988.

[38] Eder Santana and George Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016.

[39] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[40] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, and Marc Lanctot. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[41] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[42] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[43] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[44] Henrik Svensson, Serge Thill, and Tom Ziemke. Dreaming of electric sheep? exploring the functions of dream-like mechanisms in the development of mental imagery simulations. *Adaptive Behavior*, 21(4):222–238, 2013.

[45] Chris Thornton and Benedict Du Boulay. *Artificial intelligence through search*. Springer Science & Business Media, 2012.

[46] Boris A. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984.

[47] JN Tsitsiklis and B Van Roy. An analysis of temporal-difference learning with function approximation technical. Technical report, Report LIDS-P-2322). Laboratory for Information and Decision Systems . . . , 1996.

[48] David Vernon. *Artificial cognitive systems: A primer*. MIT Press, 2014.

[49] Théophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, and Yujia Li. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

[50] Klaus Wunderlich, Peter Smittenaar, and Raymond J. Dolan. Dopamine enhances model-based over model-free choice behavior. *Neuron*, 75(3):418–424, 2012.

[51] Francesco Zambon, Dinesh Sethi, and Francesca Racioppi. *European status report on road safety: towards safer roads and healthier transport choices*. WHO Regional Office for Europe, 2009. OCLC: ocn656802812.