Online State Elimination in Accelerated reinforcement Learning

Safreni Candra Sari, Kuspriyanto, Ary Setijadi Prihatmanto,
and Widyawardana Adiprawita

School of Electrical Engineering and Informatics, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung, 40132, Indonesia
safrenicsari@yahoo.com, kuspriyanto@lskk.ee.itb.ac.id, asetijadi@lskk.ee.itb.ac.id,
wadiprawita@stei.itb.ac.id.

Abstract: Most successes in accelerating RL incorporated internal knowledge or human intervention into the learning system such as reward shaping, transfer learning, parameter tuning, and even heuristics. These approaches could be no longer solutions to RL acceleration when internal knowledge is not available. Since the learning convergence is determined by the size of the state space where the larger the state space the slower learning might become, reducing the state space by eliminating the insignificant ones can lead to faster learning. In this paper a novel algorithm called Online State Elimination in Accelerated Reinforcement Learning (OSE-ARL) is introduced. This algorithm accelerates the RL learning performance by distinguishing insignificant states from the significant one and then eliminating them from the state space in early learning episodes. Applying OSE-ARL in grid world robot navigation shows 1.46 times faster in achieving learning convergence. This algorithm is generally applicable for other robotic task challenges or general robotics learning with large scale state space.

Keywords: Reinforcement Learning, robot learning, Reinforcement Learning, Accelerated Reinforcement Learning, Soccer robotics.

1. Introduction

Robot learning is a research field at the intersection of machine learning and robotics. It studies techniques allowing a robot to acquire novel skills or adapt to its environment through learning algorithms. A remarkable variety of problems in robotics may be naturally phrased as problems of Reinforcement Learning[1]. Reinforcement learning enables a robot to autonomously discover an optimal behavior through trial-and-error interactions with its environment. RL is a well-known technique for the solution of problems where agents need to act with success in an unknown environment, learning through trial and error [2].Unfortunately, convergence of any RL algorithm requires extensive exploration of the state-action space, which can be very time consuming. Therefore acceleration of learning processes is one of important issues in reinforcement learning[3, 4].

Most successes in accelerating RL incorporated internal knowledge or human intervention into the learning system, such as reward shaping, transfer learning, parameter tuning, and even heuristics. These approaches could be no longer solutions to RL acceleration where internal knowledge is not available. This paper proposed a novel approach in improving the RL learning performance called by accelerating the speed of the learning convergence without involving heuristics or any internal knowledge.

Since the learning convergence is determined by the size of the state space where the larger the state space the slower learning might become, reducing the state space by eliminating the insignificant states can lead to faster learning. In this research a novel method called Online State Elimination in Accelerated Reinforcement Learning (OSE-ARL) is introduced. This algorithm distinguishes insignificant states from the significant one from early learning episode, which reducing the state space during the learning process.

Received: February 28th, 2014. Accepted: December 9th, 2014

The remainder of this paper is organized as follows. Section II briefly reviews RL approaches and describes the Q(λ)and SARSA(λ)algorithm, while Section III presents a

review of some existing approaches to speed up RL. Next, Section IV shows how the learning speed can be improved by eliminating some insignificant states from the state space during the learning process. Then, section V details the mapping grid world robot navigation into RL and the experiments performed in the domain, and analyses the results obtained. Finally, section VI presents conclusions and future directions.

2. Reinforcement Learning

Reinforcement Learning [3] is a theoretically-grounded machine learning method designed to allow an autonomous agent to maximize its long-term reward via repeated experimentation in, and interaction with, its environment. Under certain conditions, Reinforcement Learning is guaranteed to enable the agent to converge to an optimal control policy, and has been empirically demonstrated to do so in a series of relatively simple test bed domains[5].The common approach in RL is to model the process of learning a task as a Markov Decision Process (MDP). The MDP is defined as the 4-tuple $\langle S, A, Pr, R\rangle$, where$S$ is a set of states and A is a set of actions. The state transition probability density function $Pr: S \times A \times S \to [0,1]$ defines the probability density over S for the next states$s_{t+1} \in S$ after executing action $a_t \in A$ in state $s_t \in S$. The reward function $R: S \times A \times S \to \mathbb{R}$ defines the reward of a state transition as $r_{t+1} = R(s_t, a_t, s_{t+1})$. A control policy (or simply policy) $\pi: S \times A \to [0,1]$ defines the action selection probability density for all actions in all states. An MDP has the Markov property, which means that transitions only depend on the current state-action pair and on neither past state-action pairs nor on information excluded from$s$. This implies that s must contain all relevant state information on the agents and the environment.

The agent's goal is to maximize, at each time-step *k*, the expected discounted return R:

$$R_t = E\{\sum_{j=0}^{\infty} \gamma^j r_{t+j+1}\} \quad (1)$$

where $\gamma \in [0,1)$ is the discount factor, and the expectation is taken over the probabilistic state transitions. The quantity R_t compactly represents the reward accumulated by the agent in the long run. The discount factorγ can be regarded as encoding increasing uncertainty about rewards that will be received in the future, or as a means to bind the sum that otherwise might grow infinitely.

The value function $V^{\pi}(s)$gives the expected return of the following policy π from state s:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \,|s_t = s\} \quad (2)$$

where $E_{\pi}\{\cdot\}$ denotes the expected value given that the agent follows policy π. The task of the agent is, therefore, to maximize its long-term performance, while only receiving feedback about its immediate, one-step performance. One way it can achieve this is by computing an optimal action-value function. The action-value function or Q-function $Q(s,a)$ gives the estimated return of choosing action a in state s and following the control policy afterwards:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\} = E_{\pi}\{\sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \,|s_t = s, a_t = a\} \quad (3)$$

A policy that is better than or equal to all other policies with respect to R for all $s \in S$ is an optimal policy, denoted π^*. All optimal policies share the same optimal value function $V^*(s)$ and optimal action-value function$Q^*(s,a)$.

Temporal Difference (TD) learning methods have the goal to estimate $V^{\pi}(s)$ or $Q^{\pi}(s,a)$. TD methods estimate the (action-) value function at time step k, $Q_k(s,a)$, by bootstrapping

from an initial estimate, using information from single state transitions. Because TD methods learn from single observed state transitions, they do not need a model. They work on-line, for both episodic tasks and infinite horizon tasks. The following recursive reformulation of $Q^{\pi}(s,a)$ (reformulation$V^{\pi}(s)$is analogous) shows the relation between $Q^{\pi}(s_k,a_k)$ and $Q^{\pi}(s_{k+1},a_{k+1})$,

$$Q^{\pi}(s,a) = E_{\pi}\{r_{t+1} + Q^{\pi}(s_{t+1},a_{t+1}) | s_t = s, a_t = a\} \quad (4)$$

This formulation can be used to derive the TD error $\delta_{TD,t+1}$of the transition, which gives the difference between the current estimate $Q_t^{\pi}(s_t,a_t)$ and the estimatebased on r_{t+1} and $Q_t^{\pi}(s_{t+1},a_{t+1})$:

$$\delta_{TD,t+1} = r_{t+1}\gamma Q_t^{\pi}(s_{t+1},a_{t+1}) - Q_t^{\pi}(s_t,a_t) \quad (5)$$

The TD error is used to update the estimate of $Q_t^{\pi}(s_t,a_t)$. For discrete state-action spaces$Q$can be updated as follows:

$$Q_{t+1}^{\pi}(s_t,a_t) = Q_t^{\pi}(s_t,a_t) + \alpha\delta_{TD,t+1} \quad (6)$$

in which $\alpha \in (0,1]$ is the learning rate or step size.

In TD control, the policy is directly derived from $Q(s,a)$. An important policy is the greedy policy, which selects $a_{t,greedy}$, the action with the highest estimated return:

$$a_{t,greedy} = arg\max_{a'} Q^{\pi}(s_t,a') \quad (7)$$

while greedy actions exploit the knowledge gained and currently stored in $Q(s,a)$, new knowledge can be gained from selecting exploratory, non-greedy actions. A widely used action selection policy that includes exploratory actions is the ϵ-greedy policy $\pi_{\epsilon-greedy}(s_t,a_t)$which is defined such that a random action is selected with probability ϵ(uniformly sampled from A) and $a_{t,greedy}$otherwise:

$$\pi_{\epsilon-greedy}(s_t,a_t) = \begin{cases} 1-\epsilon+\frac{\epsilon}{|A(s)|}, & if a_t = a_{t,greedy} \\ \frac{\epsilon}{|A(s)|}, & if a_t \neq a_{t,greedy} \end{cases} \quad (8)$$

with $\epsilon \in [0,1]$the exploration rate and $|A(s)|$the number of actions in A within state s. For a goodtrade-off between exploration and exploitation, the value for $\epsilon$is typically chosenfrom the range [0.01,0.20] [6].

Popular on-line TD control algorithms are Q-learning and SARSA. SARSA is an on-policy algorithm, estimating the value function for the policy being followed. Q-learning is an on-policy algorithm under which $Q(s,a)$ converges to the optimal value function $Q^*(s,a)$ belonging to the optimal policy π^*, independently of the policy actually followed during learning. The TD-errors for these algorithms are computed as follows:

$$\delta_{TD_{SARSA},t+1} = r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t) \quad (9)$$

$$\delta_{TD_Q,t+1} = r_{t+1}\gamma + \max_{a'} Q_k(s_{t+1},a_{t+1}) - Q_t(s_t,a_t) \quad (10)$$

To speed up convergence, SARSA and Q-learning can be combined with eligibility traces[3], thereby forming SARSA(λ) and Q(λ), respectively. With eligibility traces, the TD error is not only used to update $Q_t(s,a)$ for $s = s_t,\ a = a_t$, but also for state-action pairs that were visited earlier in the episode. In this process, more recently visited (s,a)-pairs receive a

stronger update than pairs visited longer ago. For discrete state-action spaces, $Q(s,a)$ is updated, $\forall s \in S,\ \forall a \in A$, as follows:

$$Q_{t+1}^{\pi}(s,a) = Q_t^{\pi}(s,a) + \alpha\delta_{TD,t+1}e_{t+1}(s,a) \quad (11)$$

with

$$e_{t+1}(s,a) = \begin{cases} \gamma\lambda e_t(s,a) + 1, & if\, s = s_t \text{ and } a = a_t \\ \gamma\lambda e_k(s,a) & \text{otherwise} \end{cases} \quad (12)$$

where $e_t(s,a)$ contains the eligibility of a state-action pair at time step t with $e_0(s,a) = 1$, and λ the (eligibility) trace discounting factor. For Q(λ), the eligibility of preceding states is only valid as long as the greedy policy is followed.

Thus, for Q(λ), $e$is also reset after an exploratory action. Choosing a value for λ can be done in the same way as for using a characteristic time scale for the eligibility of the agent's actions.

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s,a
Repeat (for each episode):
  Initialize s,a
  Repeat (for each step of episode):
    Take action a, observer r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s,a:
    $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    $s' \leftarrow s; a' \leftarrow a$
    until s is terminal

Figure 1. Tabular SARSA(λ).

SARSA is an on-policy TD control method, which the first step is to learn an action-value function rather than a state-value function. In particular, for an on-policy method we must estimate $Q^{\pi}(s,a)$for the current behavior policy π and for all states $s$and actions a. When we consider transitions from state-action pair to state-action pair, and learn the value of state-action pairs, formally these cases are identical: they are both Markov chains with a reward process. The theorems assuring the convergence of state values under TD(0) also apply to the corresponding algorithm for action values:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \quad (13)$$

This update is done after every transition from a non terminal state s_t. If $s_{t+1}$is terminal, then $Q(s_{t+1},a_{t+1})$is defined as zero. This rule uses every element of the quintuple of events,$(s_t,a_t,r_{t+1},s_{t+1},a_{t+1})$, that make up a transition from one state-action pair to the next. This quintuple gives rise to the name SARSA for the algorithm. The general form of the complete eligibility trace version of SARSA or SARSA(λ) is given in Figure 1.

One of the most important breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as *Q-learning* [7]. Its simplest form, *1-step Q-learning* is defined by

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)] \quad (14)$$

In this case, the learned action-value function, Q, directly approximates Q^*, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated. The complete algorithm in pseudo code is given in figure 2.

Initialize $Q(s,a)$ arbitrarily and $e(s,a)=0$, for all s,a
Repeat (for each episode):
  Initialize s,a
  Repeat (for each step of episode):
    Take action a, observer r,s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $a^* \leftarrow arg\max_b Q(s',b)$(if$a'$ ties for the max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r+\gamma Q(s',a^*)-Q(s,a)$
    $e(s,a) \leftarrow e(s,a)+1$
    For all s,a:
    $Q(s,a) \leftarrow Q(s,a)+\alpha\delta e(s,a)$
    If $a'=a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
      , else $e(s,a) \leftarrow 0$
    $s' \leftarrow s; a' \leftarrow a$
    until s is terminal

Figure 2. Tabular version of Watkins's Q(λ) algorithm.

3. Approach on Accelerated Reinforcement Learning

The quality of the learning itself is measured based on eventual convergence to optimal, speed of convergence to optimality and regret [4]. Although many algorithms come with a provable guarantee of asymptotic convergence to optimal behavior [7], an agent that quickly reaches a plateau at 99% of optimality may, in many applications, be preferable to an agent that has a guarantee of eventual optimality but a sluggish early learning rate. Therefore the speed of convergence to near-optimality is more practical to be measured. The speed of convergences to the near optimality with high dimension environment is often big issues in RL. One effort that can be applied to accelerate RL is to find a new algorithm that reduces the state space by carefully eliminating some unimportant states while learning. If not careful enough then the potentially important state might also be eliminated, and the learning process will fail.

Several methods have been proposed to speed up RL. One of them is incorporate the prior knowledge into RL. Mataric[8] used implicit domain knowledge to design the reinforcement/reward function in situated domains based on utilizing heterogeneous reward functions and goal specific progress estimator. Laud and DeJong [9] formulated an explanation of the potential of reward shaping to accelerate reinforcement learning with a reward-based analysis. Konidaris and Barto[10]introduced the use of learned shaping rewards in RL tasks, where an agent uses prior experience on a sequence of tasks to learn a portable predictor that estimates intermediate rewards, resulting in accelerated learning in later tasks that are related but distinct. Matignon, Laurent et al. [11]accelerate goal-directed RL by modifying the reward function using a binary reward function (for discrete state space) and continuous reward function (for continuous state space) and implementing Gaussian goal biased function as the initial values of *Q(s)*. Ma, Xu et al. [12] applied a state-chain sequential feedback Q-learning algorithm for path planning of autonomous mobile robots in unknown static environments, where the state chain is built during the searching process.

Another approach in accelerating the RL is by applying transfer learning in RL. The core idea of transfer is that experience gained in learning to perform one task can help improve learning performance in a related, but different, task[13]. Drummond [14] used transfer

learning from the related tasks, which generate a partitioning of the state space which is then used to index and compose functions stored in a case base to form a close approximation to the solution of the new task. Taylor and Stone [15]introduced behavior transfer, a novel approach to speeding up traditional RL. Celiberto, Matsuura et al. [2]applied transfer learning from one agent to another agent by means of the heuristic function speeds up the convergence of the algorithm. Case-based is used to transfer the learning, and it makes TL-HAQL algorithm. Peters and Schaal[16]reduced the problem of learning with immediate rewards to a reward-weighted regression problem with an adaptive, integrated reward transformation for faster convergence. Takano, Takase et al. [17] accelerated the learning process by implementing the effective transfer learning method, which merges a selected source policy to the target policy without negative transfers. Norouzzadeh, Busoniu et al. [18]used two transfer criteria in measuring agent's performance (by the distance between its current solution and the optimal one and by the empirical return obtained) to decide when to transfer learning from an easier task to a more difficult one so that the total learning time is reduces.

More recent proposal in accelerating RL is to include heuristics in RL algorithms. Gao and Toni [19] incorporate heuristic, represented by arguments in value-based argumentation into RL by using Heuristically Accelerated RL techniques in RoboCup Soccer Keepaway-Takeaway game. Celiberto, Matsuura et al. 2010 [2] applied transfer learning from one agent to another agent by means of the heuristic function speeds up the convergence of the algorithm. Case Based (CB) is used to transfer the learning, and it makes RL algorithm faster. Terashima, Takano et al. [20]used the prior information on the problem utilizing options as prior information. In order to increase the learning speed even with wrong options, methods for option correction by forgetting the policy and extending initiation sets. Bianchi et al. [21] presented a novel class of algorithms, called Heuristically-Accelerated Multi-agent Reinforcement Learning (HAMRL), which allows the use of heuristics to speed up well-known multi-agent reinforcement learning algorithms. Such HAMRL algorithms are characterized by a heuristic function, which suggests the selection of particular actions over others.

Other approaches were also proposed. Senda, Mano et al.[22]reduced state space by modelling the state space by 3D space coordinates where then the space model is simplified by converting 3D coordinates to 2D coordinates under a certain terms. Grounds and Kudenko[23] investigated the use of parallelization in RL, with the goal of learning optimal policies for single-agent RL problems more quickly by using parallel hardware. Braga and Araújo[24]influenced zone algorithm, an improvement over the topological RL agent (TRLA) strategy, that allows reducing the number of requested interactions, which is based on the topological-preserving characteristic of the mapping between states (or state–action pairs) and value estimates. Kartoun, Stern et al. [25] allowed several learning agents to acquire knowledge from each other. Acquiring knowledge learnt by an agent via collaboration with another agent. Price and Boutilier 2003[26]proposed an implicit imitation that can accelerate reinforcement learning dramatically in certain cases, roughly by observing a mentor, a reinforcement learning agent can extract information about its own capabilities in, and the relative value of, unvisited parts of the state space. Potapov and Ali [27] tuned the learning steps, discount and exploration degree parameters to influence the convergence rate. McGovern, Sutton et al. [28]used built in policies or macro-actions as a form of domain knowledge that can improve the speed and scaling of reinforcement learning algorithms.

The algorithm proposed in this paper is aimed to improve the RL learning performance by accelerating the speed of the learning convergence without involving heuristics or any learning domain prior knowledge. Since the learning convergence is determined by the size of the state space, where the larger the state space the slower learning might become, reducing the state space can lead to faster learning. Instead of heuristics or any learning domain prior knowledge, this proposed method identifies some potential consistent local minima states to be considered as insignificant states and is considered to be eliminated from the state space. This method reduces the state space, decreasing the computation order and hence accelerating the convergence speed.

4. Online State Elimination to Accelerate Reinforcement Learning

The complexity of an algorithm is often expressed using big O notation. Big O notation is useful when analyzing algorithms for efficiency. In RL, if a good task representation or suitable initialization is chosen, the worst-case complexity of reaching a goal state has a tight bound of $O(n^3)$ action executions for Q-learning and $O(n^2)$ action executions for value-iteration [29], where n stands for number of states in the state space. If the agent has initial knowledge of the topology of the state space or the state space has additional properties, the $O(n^3)$ bound can be decreased further. In our case, where prior knowledge is not available, initial knowledge is not incorporated in the new algorithm.

Since the worst case complexity depends totally on number of states, it's very clear that n has very dominant factor in determining the convergence speed, where reducing n will lead to decreasing the computation needed to reach learning convergence. When robot learns to master a new skill, it learns to determine which states considered important to support its performance. Robot learns to classify which states are significant, and which states are less significant. By updating its *Q (s,a)* values every iteration, agent update its policies by choosing the highest Q value as its decision factor, which means it starts to ignore smaller Q value (which indicates less significant states). This condition forces the agent to rarely visit these less significant states until agent succeeded in maximizing its rewards.

Almost all RL algorithms are based on estimating value functions--functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The notion of "how good" here is defined in terms of future rewards that can be expected, or, to be precise, in terms of expected return. Of course the rewards the agent can expect to receive in the future depend on what actions it will take. Accordingly, value functions are defined with respect to particular policies. The value of the state space in this case represents the significance factor of the state. High value state represents the high probability that agent will decide in determining its optimum policyπ^*.A high value state means significant states that have to be maintained in the state space because it provides solution to the agent. On the other hand the states that have less value become less interesting for the agents. The probability to visit these states is towards 0 in 100% exploitation cases. When the insignificant factor of these states can be measured, the states which have high insignificancy can be considered to be eliminated from the state space leaving it reduced.

In order to determine the insignificance of a state, in this paper a new tuple ι is proposed.
Definition 1: The insignificance function$\iota: S \longrightarrow \mathbb{R}$is a function that returns a value indicates the insignificance rate of a state $s \in S$. This insignificance function represents an insignificance rate of a state that derived from value *V(s)*, where the lowest value *V* of the neighborhood state is considered to be potentially insignificant. This function indicates which state $s \in S$ is insignificant enough that it can be ignored and should be eliminated from the state space, since they don't provide solutions to the agent.

Definition 2: If the domain$X$ is a metric space then$f$is said to have a local (or relative) maximum point at the point x^* if there exists some $\varepsilon > 0$ such that $f(x^*) \geq f(x)$ for all x in X within distance ε of x^*. Similarly, the function has a local minimum point at x^* if $f(x^*) \geq f(x)$ for all x in X within distance ε of x^*.

Definition 3: A state is called a local minimum state at k^{th} iteration or s_{min}^k when its *V* value is proven to be a local minimum of all *V(s)* function in k^{th} iteration for all $s \in S$.
Since the first learning iteration every state in state space has initial insignificance value $\iota(s) = 0$ for all $s \in S$, as the initial value of the *V(s)*. When agent update its state-action value by harvesting rewards *R(s)* every time it visits a state, insignificance factor $\iota(s)$ is also updated by the following return:

$$\iota_{k+1}(s) = \begin{cases} \iota_k(s) + \mu if sis\ s_{min}^k \\ \iota_k(s) = 0 \qquad otherwise \end{cases} \tag{15}$$

where $\iota_{k+1}(s)$ represents the insignificance valueι of a state$s$ in the $(k+1)^{th}$ iteration, $\iota_k(s)$ the insignificance value of state s in k^{th} iteration, and μ is the insignificance step which represents the increasing potential of a insignificant state. The insignificance of state s is updated every iteration but it will be reset back to 0 when in the next iteration s is no longer a local minimum ($s \neq s_{min}^k$.).

Definition 4: A sub state space $S_{min}^k \subset S$ is a state space at k^{th} iteration, which consists of all $s \in S$and $\iota_k(s) \geq$ I. When $\iota_k(s)$ of a state$s$ larger than a threshold value I, the state will be added to a sub state space $S_{min}^k \subset S$, which is then considered to be eliminated from the state space.

$$S_{k+1} \longleftarrow S_k \cap S_{min}^k \tag{16}$$

In every iteration k^{th}, agent reduce its Q table to only the new state space S_{k+1}, and original action space A. However learning at early stages is essentially random exploration (Bianchi, 2013). Deciding which states is more significant than others in this stage gives very small contributions since every states has it own significance potential. It's very important however to expand I to an exponential function that vary to iteration number k as given in the following equation

$$\mathrm{I}(t) = \mathrm{I}_0 e^{\mathrm{h/k}} \tag{17}$$

where I_0 is a initial value of I and h is a real number. This function gives $\mathrm{I}(t) = \mathrm{I}_0$, when k goes to infinity. This has to be done since in early stage/exploration stage since it's very important to let agent see all possibilities that it can profit from *V(s)*. The complete algorithm in pseudo code is given in Figure 3.

Initialize $Q(s,a)$ arbritarily
Define number of episodes k
Define elimination start episode p
$\iota_s = 0$ (for all $s \in S$)
t=0
Repeat (for each episode*k*):
  Initialize s
Repeat (for each step of episode):
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  Take action a, observer r, s'
  $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
  $s \leftarrow s'$;
until s is terminal
if *k>p*
  $\mathrm{V}(s) = \max_a Q(s,a)$
  k=k+1
  Repeat (for all $s \in S$)
  if V (s) is local minimum
  $\mathrm{I} \longleftarrow \mathrm{I}e^{h/k}$
  $\iota_s = \iota_s + \mu$

  if$\iota_s \geq$ I

$S_\iota \longleftarrow S_\iota \cup \{s\}$
else
$\iota_s = 0$
$S_\iota \longleftarrow S_\iota$
Endif
Endif
$S \longleftarrow S \backslash S_\iota$
until s is terminal
Endif

Figure 3. OSE-ARL Algorithm

5. Mapping Grid world Robot Navigation into Reinforcement Learning

One of the dominant topics in current robotics research is that of autonomous navigation. In the robot navigation problem, the robot need to find an optimal navigable path in a given environment, with certain constraints imposed on the robot, such as a time limit or limited availability of resources. Optimal path here refers to a path between the two points: the source and destination, which has the least path cost, or in other words the most profitable one among all the existing paths.

The environment is a discrete grid-world with randomly located obstacles. There are three robot agents on the grid-world, starting from an arbitrary initial position. The agent, which is nothing but a simple mobile robot, can occupy a single empty tile at a time and is faced with the task of navigating through the map in an autonomous manner. There can only be one agent on one tile at a time. The agent is capable of sensing its immediate environment and moving in 5 directions (action) one tile at a time respectively North, South, West, East and stay put, that makes the action space $A = \{N, S, W, E, SP\}$ available for the agent. The grid-world environment is given in Figure 4.

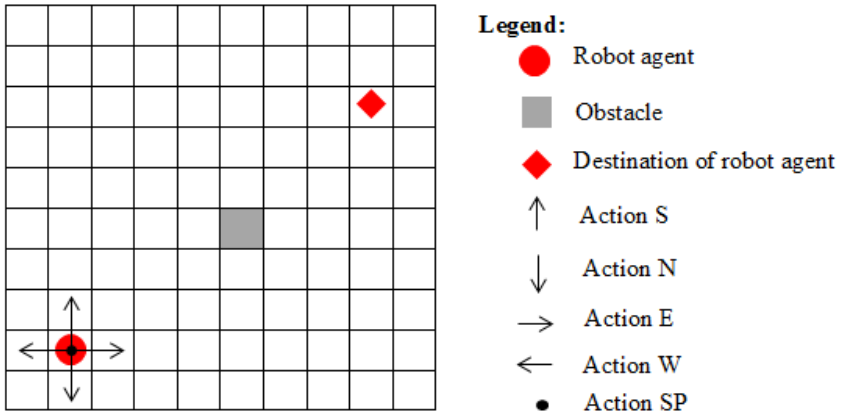


Figure 4. A robot agent on the 10x10 grid world can move to 5 directions resp. North, South, West, East, and stay put, learn to find shortest path to its destination and avoiding obstacles.

The state set Sin this environment is defined as:

$$S = \left\{ \left(p_{agent} \right);\ p_{agent} \in \{(1\ 1), (1\ 2), \ldots, (10\ 10)\} \right\} \quad (18)$$

where $p_{agent}$is the agent position.
The task of the robot agent is to find sequence of actions that have to be performed to achieve the goal state (destination). The reward function $R(s, a, s')$ for all agents are given as follows

$$R(s, \mathrm{a}, s') = \begin{cases} r^{\rightarrow}, & \mathrm{s} \in S^{\rightarrow} \\ r^{+}, & \mathrm{s} \in S^{+} \\ r^{-}, & \mathrm{s} \in S^{-} \end{cases} \quad (19)$$

where $r^{\rightarrow}$, r^{+}, and r^{-}, is resp. the reward when agent takes action to move to one of the 5 directions, when agent achieved the goal state, and when agent bumped the wall or the obstacle.

Every learning episode starts from the same initial position where agent is always in the same grid. The agent performs the learning task until the episode is ended. There are 2 situations that end the episode: when the agent arrived at the destination state, and when maximum trial had been achieved. For action selection the following *ε-greedy* scheme (eq. 8) is used.

6. Experiments and Empirical Result

In the grid world robot navigation problem, an RL agent tried to maximize its returns by finding sequence of optimum policies. The RL agent occupy distinct cells of 10x10 grid (Figure 4) - which the cell's vector coordinate is defined as state, and can take one of 5 actions on each turn: move North (N), South (S), West (W), East.

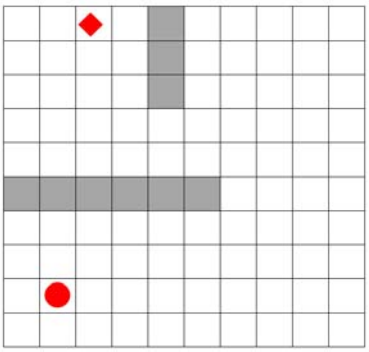
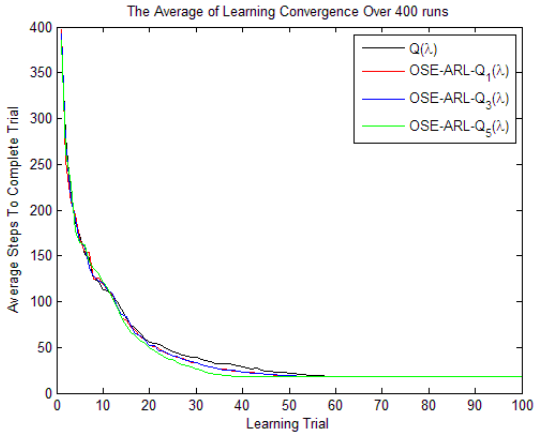


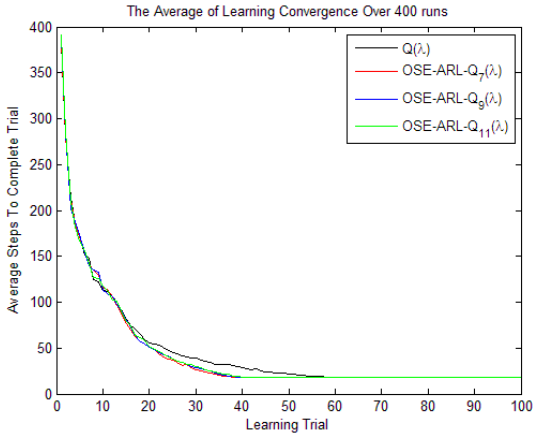
Figure 5. Initial position of robot agents on grid world.

At the beginning of an episode, the agent is positioned in the configuration depicted in Fig 5, while 9 obstacles are placed surrounding the destination or goal state. When the agent takes an action it will take it to the next state. For example when agent is in state (1,1), and take action N, then in the next iteration the state of the agent is (1,2), while taking action S will put the agent on the same state. The system is deterministic, therefore the transition function is $T : S \times A \times S \rightarrow [0, 1]$. Finally when agent arrived at the destination, or has taken 500 iterations step, the episode is ended, and agent started new episode from the same initial position.

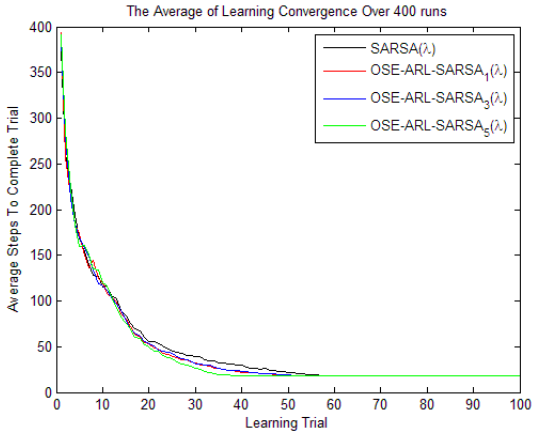
The learning parameters of the agent are setup as follows: learning rate α=0.3, discount factor γ=0.95, eligibility trace decay factor λ=0.5, and initial exploration probability ε=0.5, decaying with the trials count, the $(r^{\rightarrow}, r^{+}, r^{-}) = (-1, 10, -2)$.In the episodic learning the number of trials which indicates how many episodes to allow learning to run at most is set to be 100. The maximum iterations to allow a trial to run at most is set to be 500. The algorithms were implemented in Matlab and executed in desktop, with 4GB of RAM in a Windows 7 OS platform.


The Average of Learning Convergence Over 400 runs
Q(λ)
OSE-ARL-Q1(λ)
OSE-ARL-Q3(λ)
OSE-ARL-Q5(λ)
Average Steps To Complete Trial
Learning Trial
0
50
100
150
200
250
300
350
400
0
10
20
30
40
50
60
70
80
90
100

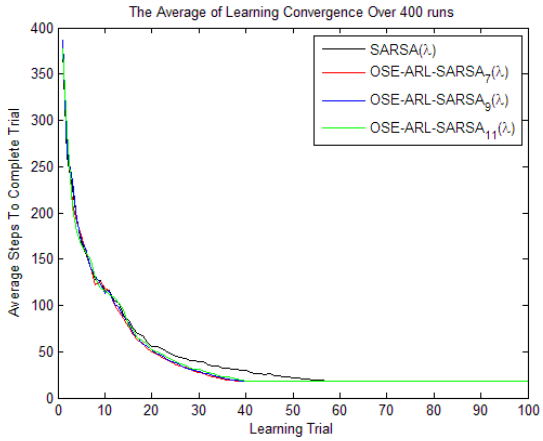
(a)


The Average of Learning Convergence Over 400 runs
Q(λ)
OSE-ARL-Q7(λ)
OSE-ARL-Q9(λ)
OSE-ARL-Q11(λ)
Average Steps To Complete Trial
Learning Trial
0
50
100
150
200
250
300
350
400
0
10
20
30
40
50
60
70
80
90
100

(b)


The Average of Learning Convergence Over 400 runs
SARSA(λ)
OSE-ARL-SARSA1(λ)
OSE-ARL-SARSA3(λ)
OSE-ARL-SARSA5(λ)
Average Steps To Complete Trial
Learning Trial
0
50
100
150
200
250
300
350
400
0
10
20
30
40
50
60
70
80
90
100

(c)



(d)

Figure 6. The learning performance of Q (λ versus OSE-ARL-Q (λ) ((a) and (b)) and SARSA (λ) versus OSE-ARL-SARSA (λ) ((c) and (d)) on 10x10 grid area for robot navigation task. The plot shows the average of the learning convergence over 400 independent runs, where each run consists of runs with 100 trials

For each experiment, a learning session consisted of 400 runs of 100 trials/episodes each. A trial finished whenever agent had arrived at the goal state or when 500 moves/iteration were completed. Each result is presented as a learning curve derived from the average steps to complete trial, which is the number of moves to achieve goal state. The experiment was conducted as follows. Q(λ), SARSA(λ), OSE-ARL-Q(λ) and OSE-ARL-SARSA(λ) are applied respectively to the robot navigation problem with 400 numbers of runs each. The performance of each algorithm is plotted and benchmarked. In figure 6 the Q(λ) is compared to 3 kinds of OSE-ARL-Q(λ), where OSE-ARL-$Q_p(\lambda)$ represents the performance of OSE-ARL-Q(λ) eliminated from episode $\rho \in Z$. In figure 7 the SARSA(λ) is also compared to 3 kinds of OSE-ARL-SARSA(λ).

Figure 6 shows the learning curves of the agent averaged over 400 learning sessions. The horizontal axis represents the numbers of episodes or trials that was done in the experiment, while the vertical axis shows how many steps or iterations needed to complete the task. Figure 6(a) shows the learning performance curve of the Q(λ), and OSE-ARLQ(λ) starting at 1st, 3rd, and 5th episodes. Even though the curve of Q(λ) had reached its convergence after 60th trial, the OSE-ARLQ(λ) was noticeably better from early episodes due to the use of online state elimination method. OSE-$ARLQ_1(\lambda)$ and OSE-$ARLQ_3(\lambda)$ reached respectively its convergence already at 55th trial and 41st trial for OSE-$ARLQ_5(\lambda)$. Similarly in Figure 6(b) OSE-$ARLQ_7(\lambda)$, OSE-$ARLQ_9(\lambda)$ and OSE-$ARLQ_{11}(\lambda)$ reached its convergence at 43rd trial. Figure 6(c) presents the learning curves of the SARSA(λ), and OSE-ARLSARSA(λ) starting at 1st, 3rd, and 5th episodes. The results show that the SARSA(λ) reached its convergence after 60th trial, while OSE-$ARLSARSA_1(\lambda)$ and OSE-$ARLSARSA_3(\lambda)$ reached 53rd, and OSE-$ARLSARSA_5(\lambda)$ after 41 trials. Lastly, Figure 6(d) presents the learning curves of the SARSA(λ), and OSE-ARLSARSA(λ) starting at 7st, 9rd, and 11th episodes. The results show that the OSE-$ARLSARSA_7(\lambda)$, OSE-$ARLSARSA_9(\lambda)$ and OSE-$ARLSARSA_{11}(\lambda)$ achieve its convergence after 42 trials. These results show that by eliminating insignificant states from early learning episodes will speed up the convergence to 1.46 times faster time. The performance analysis of the algorithm is given in table 1.

Table 1 shows the performance analysis of average (over the 400 learning runs) of the robot agent. From 400 runs, OSE-ARL-$Q_p(\lambda)$ for p=9 and p=11 run 100% successfully, while at p=1 fail in learning happens 64 times out of 400, 8 times for p=3, 5 times for p=5, 4 times for p=7. It can be seen that OSE accelerate the learning speed up to 1.46 faster even though there still is failure in learning prose's when the state elimination is executed from early stages. The failure

happened because some important states were also eliminated, because in early stages, these important states still have low state value. This failure did not happen when agent had chance to explore the states in early episodes, which can be seen on the table that when elimination is started from 9th trial and up, the learning process is 100% successful.

Table1. Performance Analysis Q(λ)versus OSE-ARL-Q(λ) and SARSA(λ) versus OSE-ARL-SARSA(λ)

RL Algorithm	*p*	Successful run (Sr)	Successful percentage (Sp)	Converge after	Acceleration factor	Performance Index (PI)	Sr/PI
Q(λ)		400	100%	60 trials	1,00	2852	14,03
OSE-ARL-$Q_p(\lambda)$	1	336	84%	55 trials	1,09	2673,2	12,57
	3	392	98%	55 trials	1,09	2676,5	14,65
	5	395	98,75%	41 trials	1,46	2507,6	15,75
	7	396	99%	43 trials	1,40	2536,5	15,61
	9	400	100%	43 trials	1,40	2571,1	15,56
	11	400	100%	43 trials	1,40	2590,6	15,44
SARSA(λ)	-	400	100%	60 trials	1,00	2791	14,33
OSE-ARL-$SARSA_p(\lambda)$	1	325	81,25%	53 trials	1,13	2635,8	12,33
	3	392	98%	53 trials	1,13	2667,8	14,69
	5	391	97,75	41 trials	1,46	2543,1	15,37
	7	399	99,75	42 trials	1,43	2523,8	15,81
	9	400	100	42 trials	1,43	2570,1	15,56
	11	400	100	42 trials	1,43	2589,5	15,45

Performance Index of each algorithm is done by measuring the error value of the learning curve,

$$PI = \sum_{i=0}^{j} y_i - y_c \tag{20}$$

where *i* is trial, *j* is number of trials, y_i is the average steps to complete trials at trial *i*, and y_c is the convergence value of the *y*. Finally the learning performance is measured by calculating the ratio of the successful runs (Sr) and Performance Index (PI). To outperform Q(λ), OSE-ARL-$Q_5(\lambda)$ gives the best Performance ratio, while OSE-ARL-$SARSA_7(\lambda)$ gives the best performance ratio in outperforming SARSA(λ). The online state elimination executed in Q(λ) and SARSA(λ) had performed faster convergence speed due to decreasing number of states in the state space.

7. Conclusion and Further Research

RL is the solution of multi robot learning problem, since the robot environment is mostly dynamic and stochastic. However the increasing number state and action space leads to problem in RL since it requires larger memory and computation time which can cause degrading in the learning performance to very poor and even lead to failure in learning.

Since the learning convergence is determined by the size of the state space where the larger the state space the slower learning might become, reducing the state space by eliminating the insignificant states can lead to faster learning. Applying online state elimination in grid world multi-robot navigation had shown significant acceleration factor in multi-agent RL to 1.46 faster convergence speed.

When internal knowledge such as reward shaping, transfer learning, parameter tuning, and even heuristics is no longer applicable to RL problems, online state elimination becomes a promising solution to accelerate RL. The successful learning rate is higher than 98%, due to probability of state importance value. Increasing state importance parameter will increase successful learning rate to 100%.

This algorithm is not only applicable for primitive robot soccer task, but also for other robotic soccer task challenges with large scale state space. This method has also clearly given us a starting point on several promising extension to the existing work and highlighted important new questions. This research has not only resulted in advances in imitation learning, but also has opened up a whole new way of exploring this field that posses an abundant source of ready-to-explore problems for future research.

8. References

[1] Kober, J., J.A. Bagnell, and J. Peters, *Reinforcement learning in robotics: A survey.* The International Journal of Robotics Research, 2013. 32(11): p. 1238-1274.

[2] Celiberto, L.A., et al. *Using transfer learning to speed-up reinforcement learning: a cased-based approach*. in *Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American*. 2010. *IEEE.*

[3] Sutton, R.S. and A.G. Barto, *Reinforcement Learning: An Introduction*1998: MIT Press.

[4] Kaelbling, L.P., M.L. Littman, and A.W. Moore, *Reinforcement learning: a survey.* J. Artif. Int. Res., 1996. 4(1): p. 237-285.

[5] Stone, P., R.S. Sutton, and G. Kuhlmann, *Reinforcement learning for RoboCup-soccer keepaway.* Adaptive Behavior, 2005. 13: p. 2005.

[6] Schuitema, E., *Reinforcement Learning on Autonomous Humanoid Robots*, 2012.

[7] Watkins, C.J.C.H. and P. Dayan, *Q-Learning.* Machine Learning, 1992. 8(3-4): p. 279--292.

[8] Mataric, M.J. *Reward Functions for Accelerated Learning*. in *ICML*. 1994.

[9] Laud, A. and G. DeJong. *The influence of reward on the speed of reinforcement learning: An analysis of shaping*. in *ICML*. 2003.

[10] Konidaris, G. and A. Barto. *Autonomous shaping: Knowledge transfer in reinforcement learning* in *Proceedings of the 23rd international conference on Machine learning*. 2006. ACM.

[11] Matignon, L., G.J. Laurent, and N. Le Fort-Piat, *Reward function and initial values: better choices for accelerated goal-directed reinforcement learning*, in *Artificial Neural Networks–ICANN 2006*2006, Springer. p. 840-849.

[12] Ma, X., et al., *State-chain sequential feedback reinforcement learning for path planning of autonomous mobile robots.* Journal of Zhejiang University Science C, 2013. 14(3): p. 167-178.

[13] Taylor, M.E. and P. Stone, *Transfer learning for reinforcement learning domains: A survey.* The Journal of Machine Learning Research, 2009. 10: p. 1633-1685.

[14] Drummond, C., *Accelerating Reinforcement Learning by Composing Solutions of Automatically Identified Subtasks.* Journal of Artificial Intelligence Research (JAIR), 2002. 16: p. 59-104.

[15] Taylor, M.E. and P. Stone. *Speeding up reinforcement learning with behavior transfer*. in *AAAI 2004 Fall Symposium on Real-life Reinforcement Learning*. 2004.

[16] Peters, J. and S. Schaal, *Reinforcement learning by reward-weighted regression for operational space control*, in *Proceedings of the 24th international conference on Machine learning*2007, ACM: Corvalis, Oregon. p. 745-750.

[17] Takano, T., et al., *TRANSFER LEARNING BASED ON FORBIDDEN RULE SET IN ACTOR-CRITIC METHOD.* INTERNATIONAL JOURNAL OF INNOVATIVE COMPUTING INFORMATION AND CONTROL, 2011. 7(5 B): p. 2907-2917.

[18] Norouzzadeh, S., L. Busoniu, and R. Babuska. *Efficient Knowledge Transfer in Shaping Reinforcement Learning*. in *Proceedings of the 18th IFAC World Congress*. 2011.

[19] Gao, Y. and F. Toni, *Argumentation Accelerated Reinforcement Learning for RoboCup Keepaway-Takeaway*, in *Theory and Applications of Formal Argumentation*, E. Black, S. Modgil, and N. Oren, Editors. 2014, Springer Berlin Heidelberg. p. 79-94.
[20] Terashima, K., H. Takano, and J. Murata, *Acceleration of Reinforcement Learning with Incomplete Prior Information.* JACIII, 2013. 17(5): p. 721-730.
[21] Bianchi, R.A.C., et al., *Heuristically-Accelerated Multiagent Reinforcement Learning.* Cybernetics, IEEE Transactions on, 2013. PP(99): p. 1-1.
[22] Senda, K., S. Mano, and S. Fujii. *A reinforcement learning accelerated by state space reduction*. in *SICE 2003 Annual Conference*. 2003. IEEE.
[23] Grounds, M. and D. Kudenko, *Parallel reinforcement learning with linear function approximation*, in *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*2008, Springer. p. 60-74.
[24] Braga, A.P.d.S. and A.F.R. Araújo, *Influence zones: A strategy to enhance reinforcement learning.* Neurocomputing, 2006. 70(1–3): p. 21-34.
[25] Kartoun, U., et al. *Collaborative Q (λ) Reinforcement Learning Algorithm-A Promising Robot Learning Framework*. in *IASTED International Conference on Robotics and Applications (RA 2005), Cambridge, U.S.A*. 2005. ACTA Press.
[26] Price, B. and C. Boutilier, *Accelerating reinforcement learning through implicit imitation.* J. artif. intell. res.(jair), 2003. 19: p. 569-629.
[27] Potapov, A. and M. Ali, *Convergence of reinforcement learning algorithms and acceleration of learning.* Physical Review E, 2003. 67(2): p. 026706.
[28] McGovern, A., R.S. Sutton, and A.H. Fagg. *Roles of macro-actions in accelerating reinforcement learning*. in *Grace Hopper celebration of women in computing*. 1997.
[29] Koenig, S. and R.G. Simmons, *Complexity Analysis of Real-Time Reinforcement Learning Applied to Finding Shortest Paths in Deterministic Domains*, 1992, Carnegie Mellon University.

Safreni Candra Sari was born in Medan on January 14, 1975, completed her undergraduate and master study in electrical engineering, Technishe Universiteit Delft the Netherlands in 1999. She followed her second master study in electrical engineering majoring Digital Media and Game Technology at the Institut Teknologi Bandung Indonesia and had completed her study in July 2010. Safreni worked as a faculty member at the General Achmad Yani University (UNJANI) in Bandung, her research interests are robotic system, robotic soccer system, learning in robotics, reinforcement learning, and multi agent learning system.

Kuspriyanto was born in Yogyakarta Indonesia, 2 January 1950, completed his undergraduate degree at Electrical engineering, Institut Teknologi Bandung in 1974. He received his Master and Doctoral degree from Université des Sciences et Techniques de Montpellier (USTL) France. He is currently a Full Professor at the Department of Electrical Engineering, Institut Teknologi Bandung, Indonesia. His current research interests include real time computing systems, computer architecture, and robotics.

Ary Setijadi Prihatmanto was born in Bandung, Indonesia on August 1972. He received his Bachelor and Master degree in Electrical Engineering from ITB, and doctorate degree in informatics from Johannes Kepler University of Linz. His research interests include dualism of computer vision & computer graphics, human-computer interface, brain-computer interface, game theory on intelligent system and its applications.

Widyawardana Adiprawita, lecturer at STEI-ITB. Received his electrical engineering degree at Electrical Engineering ITB with honour in 1997. Finished master degree at Informatics Engineering ITB in 2000. He finished his doctoral degree with honour at Electrical Engineering ITB. His Research interests are embedded system, robotics and intelligent agent autonomy. He has written more than 20 papers published in international and national publication.