A Vision-Guided Parallel Parking System for a Mobile Robot using Approximate Policy Iteration

Marwan Shaker, Tom Duckett and Shigang Yue

Abstract— **Reinforcement Learning (RL) methods enable autonomous robots to learn skills from scratch by interacting with the environment. However, reinforcement learning can be very time consuming. This paper focuses on accelerating the reinforcement learning process on a mobile robot in an unknown environment. The presented algorithm is based on approximate policy iteration with a continuous state space and a fixed number of actions. The action-value function is represented by a weighted combination of basis functions. Furthermore, a complexity analysis is provided to show that the implemented approach is guaranteed to converge on an optimal policy with less computational time.**

A parallel parking task is selected for testing purposes. In the experiments, the efficiency of the proposed approach is demonstrated and analyzed through a set of simulated and real robot experiments, with comparison drawn from two well known algorithms (Dyna-Q and Q-learning).

I. INTRODUCTION

In order to work in an unknown environment, robots require the ability to acquire novel behaviors, motor skills and control policies, as well as to improve existing ones. One of the most common and general frameworks for this type of learning and adaptation is reinforcement learning (RL). However, many RL methods are time consuming, especially for a real robot to learn complex tasks with large state spaces from scratch [5], [1].

Nevertheless, many authors, such as [4], [18], [17], [20], [23], [16], [25], [6], tried to improve the performance of reinforcement learning to accelerate the learning process and make it applicable to continuous state spaces by combining RL with different methods, such as neural networks, planning, etc. However, the high computational complexity, tuning of the initial parameters or lack of sufficient demonstration on a real world application prevented these techniques from achieving their potential to solve many real world problems. To ameliorate these problems, we apply and extend a reinforcement learning algorithm called Least-Squares Policy Iteration (LSPI) [14]. Least-Squares Policy Iteration is designed to solve control problems [14], [15], and uses value function approximation to cope with large state spaces and batch processing for efficient use of the training data. In addition, LSPI converges faster with fewer samples and no initial tuning of parameters is required, as explained in detail in Section III. In this paper, we extend LSPI so that it will work on a continuous state space by directly working on the state variables.

Development of an automatic parallel parking controller for autonomous vehicles is an important task for both industrial and military applications, where the objective is to store autonomous vehicles efficiently when not in operation [26]. Many researchers, such as [19], [7], [2], [8], [26], used parallel parking as a case study in their experiments. We demonstrate the performance of the proposed approach in both simulated and real robot experiments on the parallel parking task. In our experiments, we used two different types of basis functions - polynomial basis functions (PBF) and radial basis functions (RBF) [13] - to approximate the representation of the Q-value function. We continue our study by conducting a complexity analysis of the implemented approach to show that the implemented approach converges to an optimal policy with less computational time. The results of the simulation and real robot experiments demonstrate that the learning process has been accelerated.

The main contribution of this paper is to introduce a vision guided parallel parking system that accelerates the learning of a car parking maneuver from scratch using the LSPI algorithm. The implemented approach works directly on the state variables without discretizing the state variables. The rest of the paper is organized as follows. After discussing related work in Section II, we explain the methodology of the implemented approach for the learning process in Section III. Then, we give a complexity analysis of the approach in Section IV. In Section V, we describe the problem specification, state space representation and the use of an omnidirectional camera for the state space variable calculation. Section VI presents the experimental results, followed by our conclusions in Section VII.

II. RELATED WORK

To build an automatic parallel parking vehicle, an accurate model of the environment and the robot body and dynamics is required in advance. However, obtaining such a model is not trivial and any change in the environment may cause the approach to fail. An alternative approach to the classical planning approach is to make the encoding of a task simpler by enabling the robot to learn and adapt its behaviors using fuzzy logic, neural networks or reinforcement learning. For example, Gómez-Bravo et al. [7] introduced a fuzzy behavior-based control for parallel and diagonal parking in wheeled vehicles. Zhao et al. [26] proposed an algorithm for parking in tight spaces using a genetic fuzzy system. Gu and Hu [8] presented a path-tracking scheme for a car-like mobile robot based on neural predictive control, where a wavelet-based neural network is employed to model the non-linear kinematics of the robot. Martinéz-Marín [19] introduced a reinforcement learning approach to obtain optimal motion of non-holonomic robots.

Many authors tried to improve the performance of reinforcement learning to make it applicable to continuous state

Lincoln University, Lincoln, UK, LN6 7TS,
{mshaker,tduckett,syue}@lincoln.ac.uk

spaces by combining it with different methods (such as a neural network, etc). Millán et al. [20] presented a modification of Q-learning that works in continuous domains. The modification is based on an incremental topology preserving map (ITPM) to partition the input space, and incorporation of a bias to initialize the learning process. The continuous state-action Q-learning approach has been evaluated on a real robot that must learn to follow walls. Mahadevan [17] introduced an enhancement of the policy iteration approach called Representation Policy Iteration (RPI). RPI uses spectral graph theory [3] to build basis representations for smooth value-functions on graphs induced by MDP. The approach represents the action-value functions using a function approximator based on a number of basis functions. Johns et al. [10] introduced an extension of Mahadevan's [17] work by providing a compact spectral bases function representation using Kronecker Factorization. Mahadevan et al. [18] presented a framework for simultaneously learning in continuous MDPs, where a least-squares policy iteration method is used to learn the control policy. The proposed approach was demonstrated using the standard benchmark tasks of an inverted pendulum and mountain car.

Our approach differs from the above approaches in that we learn the control policy from scratch on the real robot using only vision for state estimation, and that the time and number of trials required to learn the parallel parking task is significantly reduced using approximate policy iteration. In addition, it required less computation time compared to policy iteration and value iteration approaches.

III. Methodology

A. Least-Squares Policy Iteration

In this work, we applied least-squares policy iteration (LSPI) [14], [15] as the learning algorithm for the following several factors:

1) LSPI converges faster with fewer samples than traditional approaches (such as gradient-descent, Q-learning and Dyna-Q), since the samples are used more efficiently. This property comes from the fact that LSPI evaluates the policy with a single pass over the set of samples, and all the samples can be used in each iteration to evaluate the policy.
2) LSPI is particularly suited to mobile robot applications because it does not require careful tuning of initial parameters, e.g., learning rate. As it has no learning rate parameters to tune, and does not take gradient steps, there is no risk of overshooting, oscillation or divergence, which are difficulties many other algorithms have to face. This property comes from the fact that the policy is evaluated over a history of samples. Thus, LSPI is insensitive to initial parameter settings.
3) LSPI learns the weights of the linear functions, which can update the Q-values based on the most updated information regarding the features, while Q-learning makes a decision directly based on Q-values.

LSPI approximates Q-values, Q^π, for a given policy, π, with a parametric function approximation instead of evaluating the optimal state-action values function directly to find the optimal policy. More precisely, the action-value function is approximated as a linear weighted combination of k basis functions (features):

$$Q(s,a) \approx \hat{Q}^\pi(s,a,w) = \sum_{i=1}^{k} \phi_i(s,a) w_i = \Phi(s,a)^T W, \tag{1}$$

where ϕ_i is the i^{th} basis function and w_i is its weight in the linear equation, W is the weights vector, k is the number of basis functions (features), s is the current state and a is the current action. The k basis functions (features) represent information extracted from the state-action pairs. They were designed manually in our experiments.

With the help of Eq. 1, the TD update equation given in [24] can be re-written as $\Phi W \approx R + \gamma P \Pi_\pi \Phi W$, where Φ is a $(|S||A| \times k)$ matrix, representing the basis functions for all state-action pairs, where $|S|$ and $|A|$ are the total number of states and actions respectively. This equation can be reformulated [14] as: $\Phi^T(\Phi - \gamma P \Pi_\pi \Phi) W = \Phi^T R$, where P is a stochastic matrix that contains the transition model of the process, and R is a vector that contains the reward values.

The weights W of the linear function $\hat{Q}^\pi$ can be extracted by solving the following linear system of equations [14]:

$$W = A^{-1} b, \tag{2}$$

$$A = \Phi^T(\Phi - \gamma P \Pi_\pi \Phi), \tag{3}$$

$$b = \Phi^T R, \tag{4}$$

but the values of P and R will be unknown or too large to be used in practice. To overcome this problem, LSPI learns A and b by sampling from the environment. A sample is defined as $\{s, a, s', r\}$, where s, a, s', r are the current state, action, next state, and immediate reward, respectively. Given a set of samples, $D = \{\{s_i, a_i, s'_i, r_i\} | i = 1, 2, \ldots, L\}$, an approximate form of Φ, $P\Pi_\pi\Phi$ and R can be constructed as follows:

$$\hat{Q} = \begin{pmatrix} \phi(s_1,a_1)^T \\ \cdots \\ \phi(s_L,a_L)^T \end{pmatrix}, \tag{5}$$

$$\widehat{P\Pi_\pi\Phi} = \begin{pmatrix} \phi(s'_1,\pi(s'_1))^T \\ \cdots \\ \phi(s'_L,\pi(s'_L))^T \end{pmatrix}, \tag{6}$$

$$\hat{R} = \begin{pmatrix} r_1 \\ \ldots \\ r_L \end{pmatrix}. \tag{7}$$

With Φ, $P\Pi_\pi\Phi$ and R, the optimal weights can be found using Eq. 2. Thus, by combining the policy-search efficiency of approximate policy iteration with the data efficiency of approximate estimation of the Q-value function, we obtain the Least-Square Policy Iteration (LSPI) algorithm [15]. The aim of LSPI is to learn a policy, π, that maximizes the corresponding Q-function.

The policy evaluation step of the approximate policy iteration depends on the Q-value function estimation de-

scribed in Eq. 1. So, whenever a new sample is collected, the weights of the approximation are updated. After the policy evaluation phase is finished processing, the policy improvement starts by selecting a policy that maximizes the approximate representation of the Q-value, as shown in Eq. 8.

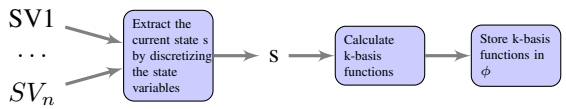
$$\pi(s|w) = \arg\max_a \phi(s,a)^T w. \quad (8)$$

Lagoudakis and Parr [14], [15], show that these estimates converge on the optimal weights of the linear function approximation as the number of samples increases. To study the performance of the implemented approach, we provide a complexity analysis in Section IV.

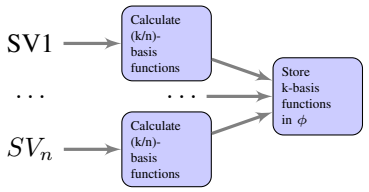
B. Modified Basis Function (Feature) Representation

In standard RL algorithms, the agent perceives the current state of the agent in the environment, denoted by s, and then executes an action a. This unique number, s, is extracted from the state space variable by discretizing the state space variables [22]. Alternatively, an approximate representation using a number of basis functions can be used, where any state, s, in the state space is approximated by a linear combination of k basis functions. However, the current state, s, is still abstracted from a number of input state variables. In this section, we explained how to avoid this abstraction by working directly with the state variables.

By contrast, in the proposed approach, the idea is to make ϕ_i represent a continuous state variable instead of representing a specific number abstracted from the state variables. LSPI calculates the k basis functions (features) for the current situation of the agent in the environment, where the current situation is represented by a unique number called the current state s, where s is abstracted from the state variables. We extend this by making LSPI calculate the k basis functions (features) directly from the state variables $(SV_1, SV_2, \ldots, SV_n)$ as shown in Fig. 1.



(a) Standard Feature Representation



(b) Modified Feature Representation

Fig. 1. Feature Representation

Assume that we have $|A|$ actions that can be performed by the agent. Then ϕ will comprise $|A|$ vectors of k elements (independent of the current state, s, in the state space $|S|$ but dependent on the state variables to compute the features in each element of ϕ). Thus, Eq. 1 is changed to

$$\hat{Q}^\pi(SV, a; W) = \sum_{i=1}^{k} \phi_i(SV, a) w_i = \Phi(SV, a)^T W, \quad (9)$$

where $SV = \{SV_1, SV_2, \ldots, SV_n\}$, $SV_1, SV_2, \ldots, SV_n$ represent n state variables. The new feature representation provides a way to deal with a continuous state space by working directly with the state space variables.

More specifically, each interaction between the agent and the environment provides a sample $\{SV, a, r, \acute{SV}\}$ (SV is initialized randomly at the first step), which describes the reward, r, received upon executing action, a, in the state variables, SV, and ending in state $\acute{SV}$. At any time, the estimates of the weights of the linear function $\hat{Q}^\pi$ can be extracted by solving the system of equations $w = A^{-1}b$ [15] as shown in Eq. 2, where the collected samples are used to update A and b as follows:

$$\tilde{A}^{t+1} = \tilde{A}^t + \phi(SV_t, a_t)(\phi(SV_t, a_t) - \gamma\phi(SV'_t, \pi(SV'_t)))^T, \quad (10)$$

$$\tilde{b}^{t+1} = \tilde{b}^t + \phi(SV_t, a_t) r_t, \quad (11)$$

where (SV_t, a_t, r_t, SV'_t) is the t^{th} sample of experience from a trajectory generated by the agent. Thus, LSPI does not explicitly estimate the probability of state transition, P, and reward function, R, because they will be either unknown or too large to be used in practice. It rather estimates the Q-function directly from interactions with the environment through samples. The computation of A and b, derived from the samples, and the resulting W^π from solving the linear system of equations, $AW^\pi = b$, is repeated for a number of iterations or until the algorithm reaches the optimal policy.

Here, we explain how the basis functions (features) are calculated in our experiments. For every action, one sub-vector of ϕ is filled with hand coded values that represent a specific action, while the rest of the sub-vectors will be equal to zero. The first element of the hand-coded sub-vector is equal to 1 and the remaining elements of the sub-vector are calculated using PBF shown in Eq. 12 or RBF shown in Eq. 13:

$$\phi_i(SV, a) = \phi_{i-1}(SV, a) * (\sum_{j=1}^{n} SV_j)/(\sum_{j=1}^{n} SV_{max_j}), \quad (12)$$

$$\phi_i(SV, a) = \exp^{-d^2/(2*\sigma^2)}, \quad (13)$$

where i is from 1 to the size of the sub-vector, SV_j and SV_{max_j} represent the j^{th} state variable and the maximum value of that state variable respectively, a is the current action, d represents the distance of the current state variable value from the center of the state variable, and σ^2 is the variance parameter of the Gaussian curve, which is usually equal to 1. We used PBF or RBF because we believe that it is a simpler notion and provides a more intuitive explanation of various function approximation schemes.

IV. Complexity Analysis

A. Computational Complexity

A standardized measurement of the computational time complexity of an algorithm is the number of elementary computer operations it takes to solve a problem, in the worst case. The number of computer operations depends on

the "size" of the problem. In the following paragraph, we will give the time complexity of some basic reinforcement learning approaches and the implemented approach.

The value iteration approach works by producing successive approximations of the optimal value function. Each iteration requires $O(\{|S|\times|A|\}^2)$ or faster if there is sparsity in the transition function [11]. However, the number of iterations required can grow exponentially as the discount factor approaches 1 and the number of states increases. Although, it costs $O(\{|S|\times|A|\}^2+|S|^3)$ steps per iteration [11], policy iteration converges in less iterations than value iteration.

Value iteration and policy iteration approaches depend on the total number of states and actions, which make these approaches inapplicable for large or continuous state-action space. In our approach, the cost of each iteration of the approximate policy iteration is shown in Eq. 14:

$$O(NB^3 + NB^2 + (NB^2 * hm) + (NB * hm)), \quad (14)$$

where NB is the number of basis function used and $NB \ll |S|\times|A|$, and hm represents the number of samples collected. In our experiments, we investigated the number of samples, hm, required to reach the optimal behavior. We tested for $hm = 50$, 100 or unlimited and observed that $hm = 100$ or unlimited does not make a major difference compared to $hm = 50$. Therefore, we used $hm = 50$ in our experiments (meaning that at each step, we collect one sample and 49 samples from the previous learning). Thus, the time complexity of approximate policy iteration is less than the value iteration and policy iteration approaches as shown in section VI-A, and is not affected by the number of states.

B. Convergence Analysis

The linear system of equation that is used to approximate the Q-value function can be re-written as follows [15]:

$$\overbrace{\Phi^T(\Phi - \gamma P\Pi_\pi\Phi)}^{A}\overbrace{w^\pi}^{X} = \overbrace{\Phi^T R}^{b}, \quad (15)$$

$$\Phi^T\Phi(I - \gamma P\Pi_\pi)w^\pi = \Phi^T R, \quad (16)$$

$$(I - \gamma P\Pi_\pi)w^\pi = (\Phi^T\Phi)^{-1}\Phi^T R. \quad (17)$$

In general, $\Phi^T\Phi$ is a non-singular matrix [9], [21]. Also, it is well known that $(I - \gamma P\Pi_\pi)$ is non-singular if, and only if, $\|\gamma P\Pi_\pi\| < 1$, where I is the identity matrix [9], [21]. Thus, the system shown in Eq. 15 has a unique solution if, and only if, $\|P\Pi_\pi\| < \frac{1}{|\gamma|}$. To find the solution of Eq. 15 we need to compute the matrix A and b and this can be done by using Eqs. 10 and 11. Thus, for convergence $\|\phi(SV_t, a_t)(\phi(SV_t, a_t) - \gamma\phi(SV'_t, \pi(SV'_t)))^T\| < 1$, where $\|\phi\| \leq 1$ satisfies the condition. This proves that the system in Eqs. 10 and 11 exists, and has a unique solution that converges for all values of $\gamma \in (0, 1)$ [12], [9]. Note that, if the matrix A is singular, singular value decomposition is used to find the solution of the system.

Also, Lagoudakis and Parr [15] show that these estimates converge to the optimal weights of the linear function approximation as the number of samples increases.

V. Problem Specification

Parallel parking was selected as a testing problem for several reasons. Developing a robust automatic parallel parking controller for autonomous vehicles is an important task for both industrial and military applications, where the objective is to store autonomous vehicles efficiently when not in operation [26]. Parallel parking is a complicated task and difficult to learn, because the driver has to have a clear view of the parking space, other cars, pavements, and any obstacles. In addition, car parking is a non-linear dynamic system and high dimensional problem (3D or more).

Parallel parking of a real mobile robot consists of finding an optimal policy and the necessary input. The optimal policy connects the initial configuration to the final configuration and satisfies existing environment constraints. For developing an autonomous mobile robot system for factory use, control variables should be carefully selected for a smooth and safe movement of the robot between any two configurations. The next subsections describe the test task.

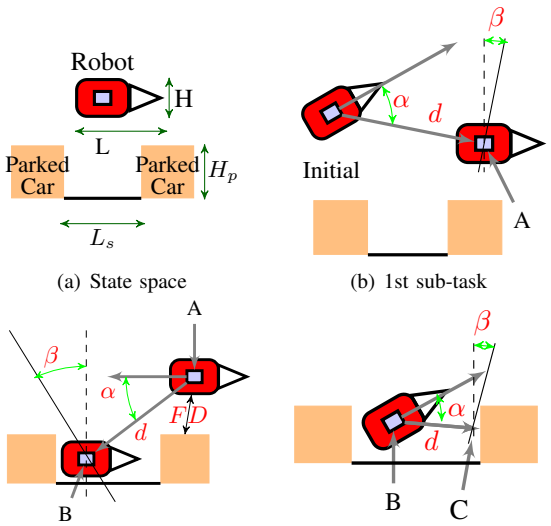
A. Description of the performed state space

The state space for parallel parking can be described as shown in Fig. 2(a), where L_s and H_p are the length and width of the parking slot respectively, and L and H are the length and width of the robot respectively. Two orange squares are used to represent two cars in the parking area as shown in Fig. 2(a). To speed up the learning process, a complex task can be learned more easily and efficiently if it is broken down into a sequence of simpler sub-tasks. The parallel parking state space is specified by three different target positions indicated by "A", "B" and "C" as shown in Fig. 2(b), Fig. 2(c) and Fig. 2(d), respectively. The parallel parking task is divided into three phases for the following reasons:

- Parallel parking involves localizing a sufficient parking bay, obtaining a convenient start location for the robot relative to the bay, and performing the parallel parking maneuver. In other words, learning to move forward until getting a point near the front car is the same idea as finding the optimal start location (hence the necessity of the first phase).
- During parallel parking, the robot will start moving backwards until it reaches the point near the back car and then starts moving forward to align itself in a parallel orientation, with respect to the front car. This validates the necessity of the second and third phases.

The maneuvering process can be completed with the following three phases. In the first phase, the mobile robot navigates forward from any location in the state space to reach the "A" position with orientation parallel to the parking space. The "A" position can be represented by the parameters of the mobile robot and parking space as $(L_s + L, H_p + H)$ in the local state space coordinates.

To reach both the desired position and orientation simultaneously, the state space is represented using two variables $\alpha \in [-50^o, 50^o]$ and $\beta \in [-50^o, 50^o]$, where α is the angle between the robot heading and "A" position, β is the angle



(a) State space (b) 1st sub-task

(c) 2nd sub-task (d) 3rd sub-task

Fig. 2. State space representation of the parallel parking task.

between the parking state space width axes and perpendicular line to d, where d is the distance between the mobile robot and the goal for that phase, as shown in Fig. 2(b).

After reaching the "A" position, the second phase starts by driving in reverse until the mobile robot gets to the "B" position as shown in Fig. 2(c). In the second phase, the state space is represented using three variables $\alpha \in [-50^o, 50^o]$, $\beta \in [-5^o, 50^o]$ and front distance ($FD \in [0, 0.5]m$), where α is the angle between the robot back heading and "B" position, β is the angle between the parking space width axes and the perpendicular line to d, where d is the distance between the mobile robot and "B" position, and front distance (FD) is the distance of the robot from the front car. If FD is greater than 0.5m, the robot is in a safe area from hitting the front car. We fixed FD to less than 0.5cm to ensure that the robot learns its behavior near the front car but without touching/bumping it. We used $\beta < -5^o$ (experimentally determined) to represent the pavement of the parking slot to avoid pavement detection. Any value of $\beta < -5^o$ means the robot has hit the pavement.

The third phase is to park the mobile robot in the parking space with the required orientation. The state space is represented by three state variables $\alpha \in [-50^o, 50^o]$, $\beta \in [-5^o, 50^o]$ and front distance ($FD \in [0, 0.5]m$) as shown in Fig. 2(d), where α is the angle between the robot heading and "C" position, β is the angle between the parking space width axes and the perpendicular line to d, where d is the distance between the mobile robot and "C" position, and front distance (FD) is the distance of the robot from the front car. The goal is represented with the state space parameters ($-7^o \leq \alpha \leq 7^o$, $-7^o \leq \beta \leq 7^o$, FD $>$ 200mm). The robot can perform two actions, either turn right or turn left by 5^o at each step. Also, the robot drives at a constant speed (forward in the 1st and 3rd phase, and backward in the 2nd phase).

To make the robot movement realistic, the robot is only allowed to turn while it is moving. The range of α and β was restricted to match the reality. It is not realistic to park a vehicle with $\alpha > 45^o$ or $\alpha < -45^o$. Also, $\beta > 45^o$ or $\beta < -45^o$ means that the vehicle is too far from the car behind. The range of values of α and β are selected as shown in Table I.

TABLE I
STATE SPACE PARAMETERS

Parameter	1st Phase	2nd Phase	3rd Phase
α	$[-50^o, 50^o]$	$[-50^o, 50^o]$	$[-50^o, 50^o]$
β	$[-50^o, 50^o]$	$[-5^o, 50^o]$	$[-5^o, 50^o]$
FD		[0,50]cm	[0,50]cm
Reward	(1) 120 if it gets to the goal, (2) -1500 if it finishes outside state space, (3) equal to a value, this value is decreased as α or β increases		
Goal State	Ready-to -reverse	reverse- into-slot	α=$[-7^o, 7^o]$ β=$[-7^o, 7^o]$ $FD \geq$200mm
Control	Two actions: turn right and turn left		

B. State Space Variable Calculation Using Omnidirectional Vision

In the real robot experiments, there are several options for vision sensors (single camera, stereo vision and omnidirectional camera). In our experiments, we need to detect two cars (orange squares to simplify the detection). Thus, a single camera is not useful if it cannot detect two distant objects. Stereo vision uses two cameras to detect two distant objects, which requires more image preprocessing time. So, we selected an omnidirectional camera for the square object detection and state space variable detection. However, the use of the omnidirectional camera raised another issue; the distortion and noise inherent from the omnidirectional images. The following paragraph explains how distortion was minimized.

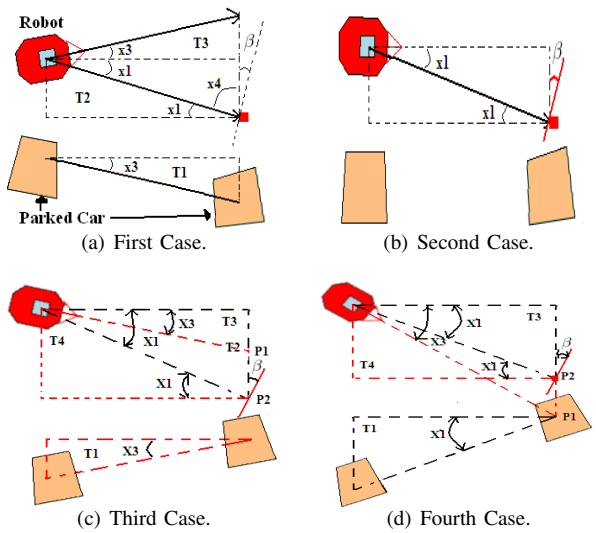
The image resolution captured from our omnidirectional camera is 2048×2048 pixels and the data had a 360-degree field of view in the horizontal plane. Thus, the orange square will have an isosceles trapezoidal shape with curved upper and lower sides. From the experiments, we notice that the average length of the upper and lower curves was 115 and 97 pixels respectively. However, a curve with 115 or 97 pixels with respect to a circle with diameter=2048 and circumference=6433.98 can be approximated by a straight line. So, we assumed that the orange square will have an isosceles trapezoid shape with four straight sides, which helped us to neglect the distortion obtained from the captured image and avoid the extra processing time required to convert omnidirectional images to panoramic images.

The robot is equipped with a 360^o omnidirectional camera shown in Fig. 8. The robot attempts to detect two orange squares (representing two cars to simplify the detection). Experimentally, we found that if the robot turns a specific amount it will cause the location of one orange square to change, depending on the direction of the turn, while the location of the other orange square will stay relatively fixed. Thus, we found that triangle T3 is approximately a mirror

of the triangle T1 in Fig. 3 and Fig. 4. When the robot is between the two orange squares, as in the third phase as shown in Fig. 5, the location of the two orange squares is changing as the robot turns. So, we created a virtual triangle T3, which connects the intersection of the robot's x-axes and the y-axes of the car behind with the center of the front car, to make T3 a mirror of T1. The visual processing for the three phases of parallel parking introduced in Section V-A is described as follows.

1) First Phase: After detecting the state space parameters, there are four cases for calculating α and β as shown in Fig. 3. Using trigonometry, x_1 & x_3 can be easily calculated. Since the triangle, T3, is an approximate mirror of the triangle, T1, in either horizontal or vertical axes, we can calculate x_3 from triangle T1 using the information obtained from the detected orange squares in the images. Now, the four cases are: (1) $\alpha = x_1 + x_3$; (2) $\alpha = x_1$; (3) $\alpha = x_1 - x_3$; (4) $\alpha = -(x_3 - x_1)$.

For all cases, β can be calculated as follows: $x_4 = 90 - x_1$, $\beta = 90 - x_4$ because β is the angle between the parking state space width (vertical) axes and the perpendicular line to d.



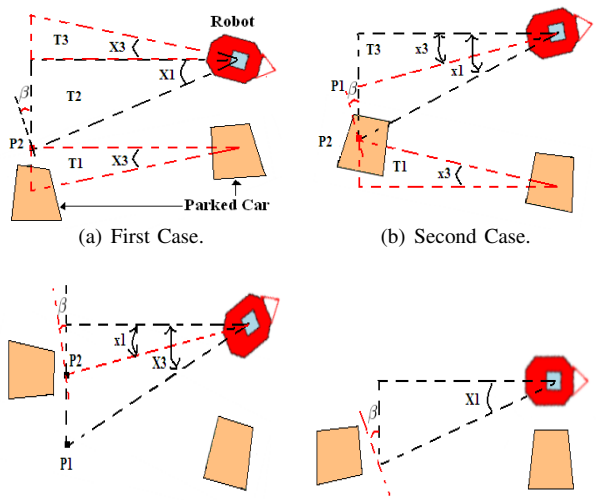
(a) First Case. (b) Second Case.

(c) Third Case. (d) Fourth Case.

Fig. 3. Calculation of α and β in the 1st phase using omni-directional camera. Orange blocks represent parked cars and drawn from view angle of the omnidirectional camera

2) Second Phase: After reaching the "A" position, there are four cases for calculating α and β as shown in Fig. 4. For the first two cases, the triangle T3 is a mirror of triangle T1 in either horizontal or vertical axes. Thus, x_3 can be calculated from T1. In case three and four, we can calculate x_1 & x_3 by simple trigonometry. The four cases are: (1) $\alpha = x_1 + x_3$; (2) $\alpha = x_1 - x_3$; (3) $\alpha = -(x_3 - x_1)$; (4) $\alpha = x_1$. For all cases, β can be calculated as shown: $x_4 = 90 - x_1$, $\beta = 90 - x_4$.

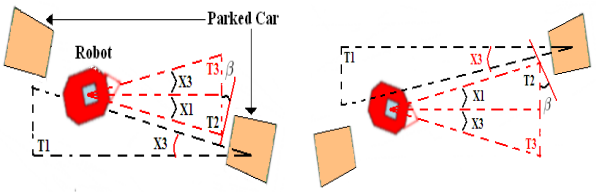
3) Third Phase: Finally, the third phase has two cases for calculating α & β as shown in Fig. 5. For both cases T1 is mirror of T3, so we can calculate x_3 from T1. For the first case $\alpha = x_1 + x_3$, but for the second case $\alpha = -(x_1 + x_3)$. There is a special case in which both cars are aligned on the same axes which means that $\alpha = 0$. For both cases β can be calculated as $x_4 = 90 - x_1$, $\beta = 90 - x_4$.



(a) First Case. (b) Second Case.

(c) Third Case. (d) Fourth Case.

Fig. 4. Calculation of α and β in the 2nd phase using omni-directional camera. Orange blocks represent parked cars and drawn from view angle of the omnidirectional camera



(a) First Case. (b) Second Case.

Fig. 5. Calculation of α and β in the 3rd phase using omni-directional camera. Orange blocks represent parked cars and drawn from view angle of the omnidirectional camera

VI. Experiments and Results

In our experiments, we compared the performance of the improved LSPI using two different types of basis functions, i.e. polynomial basis functions (PBF) and radial basis functions (RBF), with the pre-existing Dyna-Q and Q-learning algorithms [24] on the parallel parking task. To compare the performance of the proposed improvement using RBF and PBF, we performed 100 experiments and the average results are shown in Table II.

TABLE II
Compare RBF and PBF

Basis Function	Order	Number of Iteration	Average Number of Iterations over 100 experiments
PBF	2	23-231	27
PBF	4	3-9	4.8
RBF	N/A	2-3	2.1

In the case of using PBF, we tested using polynomials of order 2 and 4. For PBF with order 2, LSPI required more than 27 iterations to converge for each state and sometimes did not select the correct value. While for the polynomial of order 4, LSPI required between 3 and 9 iterations to converge for each step. Thus, we used PBF of order 4 in our simulated experiments. In case of using RBF, we used the Gaussian

function. Furthermore, LSPI with PBF requires from 3 to 9 iterations with an average 4.8 iterations to converge on the optimal policy, while LSPI with RBF requires from 2 to 3 iterations with an average 2.1 iterations. In order to gauge the efficiency of the algorithm, a series of real robot and simulated experiments were conducted. In the simulated experiments, for each algorithm and phase, we conducted 10 trials of learning and then performed 40 trials without learning to measure the performance of the learned behavior. This process was repeated until we reached 140 trials of learning.

A. Computational Complexity

To study the computational complexity of the implemented approach, we provided a formula to calculate the computational complexity for some of the reinforcement learning algorithms and the implemented approach in Section IV. We used the third phase state space representation of the parallel parking task to show the computational complexity, as shown in Table III. In Table III, we compared different frameworks of learning in terms of the number of computational operations required for each iteration. These frameworks are value iteration (VI), policy iteration (PI) and least-squares policy iteration with continuous state space and fixed number of Actions (LSPI-CSFA). Value iteration (VI) and policy iteration (PI) approaches are not applicable on a continuous or large state space because it is hard to solve the policy iteration or value iteration equations if the number of states is large [22]. So for comparison purposes we discretized the state space for PI and VI, while we calculated the computational complexity for the implemented approach using continuous state variables (the first row of Table III shows how the state space are discretized for PI and VI). Table III shows that the number of computational operations required per iteration is greatly reduced using LSPI. In our experiments, we used $hm = 50$.

TABLE III

PARALLEL PARKING COMPLEXITY

Complexity			
parameter	VI	PI	LSPI-CSFA
No. Of State Calculation	$\frac{\alpha_{max}-\alpha_{min}}{5} \times \frac{\beta_{max}-\beta_{min}}{5} \times \frac{FD_{max}-FD_{min}}{10}$	$\frac{\alpha_{max}-\alpha_{min}}{5} \times \frac{\beta_{max}-\beta_{min}}{5} \times \frac{FD_{max}-FD_{min}}{10}$	α, β and FD
No. Of State	1100	1100	Continuous state
Action	Two actions (5^o right or left)	Two actions (5^o right or left)	Two actions (5^o right or left)
NB			10
Complexity (Number of Computer Operations)	Worst Case: 4,840,000	Worst Case: 1,335,840,000	Worst Case: for hm= 50 is 6600 for hm= 100 is 12100

B. Simulated Experiments

All algorithms were compared using the number of successful attempts to reach the goal for each phase versus the number of learning trials. To test the performance of the approach over the number of learning trials, we conducted 10 trials of learning and then performed 40 trials without learning. To calculate the number of successful trials to reach the final goal, this process was repeated until we reached 140 trials of learning as shown in Fig. 6. The results given in Fig. 6 show that LSPI results in superior performance and reduced time required to learn the parallel parking task, with fewer than 50 trials required to reach the optimal behavior.

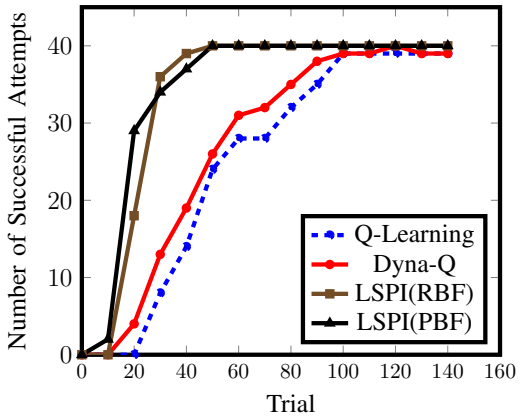


Fig. 6. Number of success (simulated experiments).

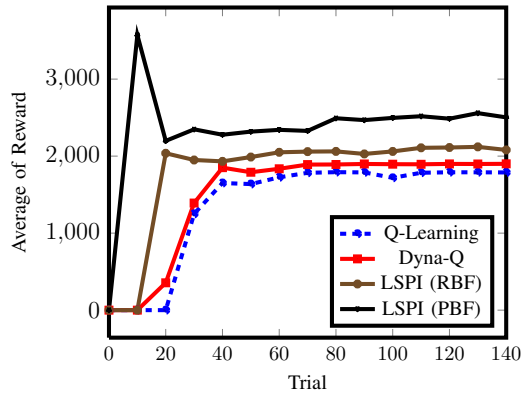


Fig. 7. Average Reward received (simulated experiments).

C. Convergence speed

At each episode, we measured the success rate of arriving at the final goal, as shown in Fig. 6, and the cumulative rewards of all the phases, as shown in Fig. 7 for all algorithms.

From Fig. 6, we can conclude that LSPI with both types of basis functions takes significantly fewer iterations to reach the optimal behavior than Dyna-Q and Q-Learning. Also, Fig. 7 shows that LSPI with both basis functions takes fewer iterations to reach the best trade-off than Dyna-Q and Q-Learning. Thus, LSPI with both basis functions converges much faster than Dyna-Q and Q-Learning. These results empirically confirmed our analysis that LSPI uses data more efficiently than Dyna-Q and Q-learning.

D. Real Robot Experiments

The proposed approach was implemented on a real mobile robot (Pioneer P3-AT, as shown in Fig. 8) to test the learning efficiency in real time. The robot is equipped with an omnidirectional camera (curved mirror & upward-pointing camera), and the method to calculate the state space variables from image was explained in Section V. During the real time processing, the robot speed was restricted to 10 mm/sec for safety reasons. To test the performance of the approach over the number of learning trials, we stopped the learning every

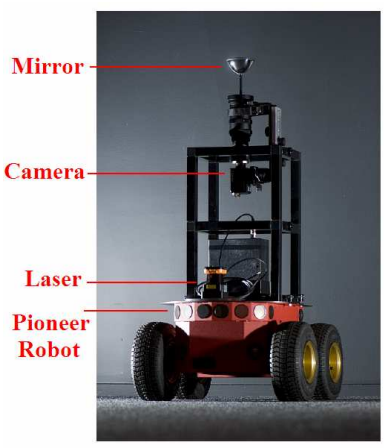


Fig. 8. Pioneer P3-AT robot used in real robot experiments.

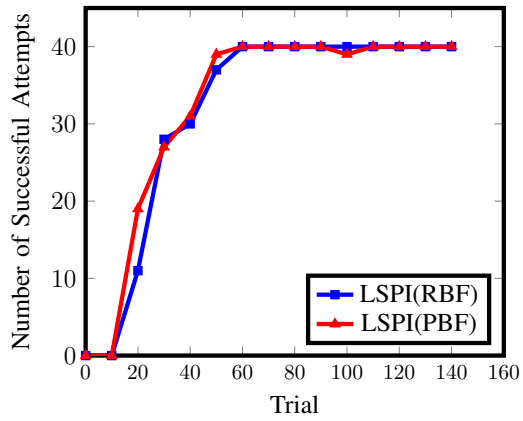


Fig. 9. Number of success (Real Robot experiments).

10 trials to perform 40 trials with the learning switched off and calculate the number of successful trials to reach the final goal, as shown in Fig 9. It was shown in Fig. 9 that the proposed approach with RBF/PBF basis functions required as few as 55 trials to reach the optimal behavior. Then using the learned policy to test the performance of the parallel parking task, we doubled, tripled and quadrupled the normal speed with learning switched off. Furthermore, we tested the performance by changing the parking slot from 1.95 to 1.7 of the robot's length and used the collected samples with learning switched off. This makes the learned policy adaptable and flexible for changes in the environment. As a future work, we will test this approach on a more complicated task or test it with a continuous action space.

VII. Conclusion

In this paper, we implemented an approach for accelerating learning of a parallel parking task. From the simulated and real robot experiments, we conclude that LSPI generated better performance (number of successes and average received reward required to reach the goal) than value-based methods, such as Dyna-Q and Q-learning. The experiments show that the time and number of trials required to reach the optimal performance was reduced dramatically.

References

[1] Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proc. International Conference on Machine learning (ICML)*, pages 1–8, Bonn, Germany, 2005.

[2] Shih-Jie Chang and Tzuu-Hseng S. Li. Design and implementation of fuzzy parallel-parking control for a car-type mobile robot. *J. Intell. Robotics Syst.*, 34(2):175–194, 2002.

[3] Fan R. K. Chung. *Spectral Graph Theory*. 1994.

[4] Dong Daoyi, Chen Chunlin, and Li Hanxiong. Reinforcement strategy using quantum amplitude amplification for robot learning. In *Proc. Chinese Control Conference (CCC)*, pages 571–575, 2007.

[5] Arkady Epshteyn, Adam Vogel, and Gerald DeJong. Active reinforcement learning. In *Proc. 25th international conference on Machine learning (ICML)*, pages 296–303, Helsinki, Finland, 2008.

[6] Chris Gaskett, David Wettergreen, Alexander Zelinsky, and Er Zelinsky. Q-learning in continuous state and action spaces. In *Australian Joint Conf. on Artificial Intelligence*, pages 417–428, 1999.

[7] F. Gómez-Bravo, F. Cuesta, and A. Ollero. Parallel and diagonal parking in nonholonomic autonomous vehicles. *Engineering Applications of Artificial Intelligence*, 14:419–434, 2001.

[8] Dongbing Gu and Huosheng Hu. Neural predictive control for a car-like mobile robot. *International Journal of Robotics and Autonomous Systems*, 39:73–86, 2002.

[9] Eugene Isaacson, Herbert Bishop Keller, and Robert Ed Isaacson. *Analysis Of Numerical Methods*. Wiley, Dover Publications, 1966.

[10] Jeff Johns, Sridhar Mahadevan, and Chang Wang. Compact spectral bases for value function approximation using kronecker factorization. In *Proc. national conference on Artificial intelligence (AAAI)*, pages 559–564, Vancouver, British Columbia, Canada, 2007.

[11] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artifical Intelligence Research*, 4:237–285, 1996.

[12] Daphne Koller and Ronald Parr. Policy iteration for factored mdps. In *proc. 6th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 326–334. Morgan Kaufmann, 2000.

[13] Ivica Kostanic and Fredric M. Ham. *Principles of Neurocomputing for Science and Engineering*. McGraw-Hill Higher Education, 2000.

[14] Michail G. Lagoudakis and Ronald Parr. Model-free least squares policy iteration. In *Proc. Advances in Neural Information Processing Systems (NIPS-14)*, 2001.

[15] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:2003, 2003.

[16] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In *NIPS*, 2007.

[17] Sridhar Mahadevan. Representation policy iteration. Proc. 21st Conference on Uncertainty in AI (UAI), 2005.

[18] Sridhar Mahadevan, Mauro Maggioni, Kimberly Ferguson, and Sarah Osentoski. Learning representation and control in continuous markov decision processes. In *Proc. 21st national conference on Artificial intelligence (AAAI)*, pages 1194–1199, Boston, Massachusetts, 2006.

[19] T. Martinez-Marin. Learning optimal motion planning for car-like vehicles. In *Proc. International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, volume 1, pages 601–612, 28–30 Nov. 2005.

[20] José Del R. Millán, Daniele Posenato, and Eric Dedieu. Continuous-action q-learning. *Mach. Learn.*, 49:247–265, 2002.

[21] L. Mirsky. *Introduction to Linear Algebra*. Dover Publications Inc., 2007-07-16.

[22] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2007.

[23] Juan C. Santamaría, Richard S. Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adapt. Behav.*, 6:163–217, 1997.

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

[25] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *ICML*, pages 1015–1022, 2007.

[26] Zhao Yanan and Jr. Emmanuel G. Collins. Robust automatic parallel parking in tight spaces via fuzzy logic. *Robotics and Autonomous Systems*, 51:111 – 127, 2005.