# Final Project Report – ENPM690 – Anubhava Paras (116905909)

# Autonomous Control and landing for a quadrotor using Least Square Policy Iteration


## Abstract:

Precise autonomous landing of a quadcopter can be a difficult problem under dynamic constraints as the controller has to generate appropriate landing trajectories for the drone to follow and reach the target. This needs to be accomplished considering the disturbances in the environment such as high-speed winds, obstacles, and may be minimal power consumption. A trial has been made to solve this problem partially using a reinforcement learning based controller that uses Least Square Policy Iteration (LSPI) to learn the control policies to generate the trajectories. It is pretty much a different approach to design the controller for a quadcopter.


## Introduction:

The primary concern of the solution proposed is to design the controller for this problem that can get adverse if we bring in different sensor noise, model inaccuracies and uncertainties of the environment. Some of the problems can be solved using different machine learning approaches such as Fuzzy Logic and artificial neural networks.

In reinforcement learning (RL) based approaches the system learns control policies that are optimal by interacting with the environment that gives the agent a feedback in terms of rewards or penalties. RL methods have proven very successful for static environments and much of research is going on to apply these methods for dynamic environments like autonomous driving, mobile robots, drones, etc.

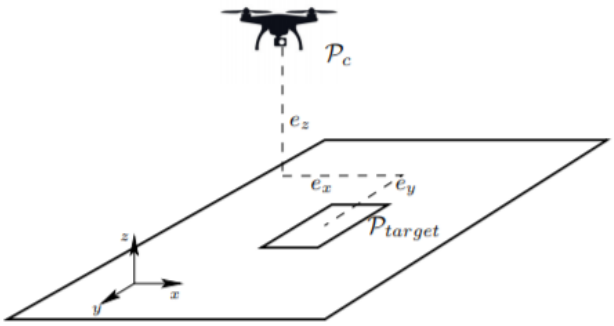
Now, here comes the problem in using RL methods for such physical systems is that it requires a lot of training data to produce optimal and accurate results, and that is many times difficult to obtain from obtain from physical systems like robots. Generating data from physical systems can be a bottleneck in progressing with the training the models. This problem to can be solved using some good simulators but problem arises as we cross the reality gap.

Quadcopter control and landing comes under the domain of continuous state and action space. Considering just the problem of landing, there can still be considerably infinite number of states comprising of quadcopter's position, linear velocity, orientation and angular velocity as a state composition and speed of the four motors as the fundamental input actions. This poses a problem at the time of training the model as this requires a lot of memory and computation to store and process these values.

There have been related projects in which RL based schemes were used for a drone with suspended load to track its trajectories. The solution proposed here is somewhat on similar lines and demonstrates the working of a particular reinforcement learning method, namely, Least Squares Policy Iteration (LSPI) for a drone landing task. **LSPI approximates the state action value function from the training data and then a greedy policy is implemented on this value function to extract the optimal control policy for the given task.**

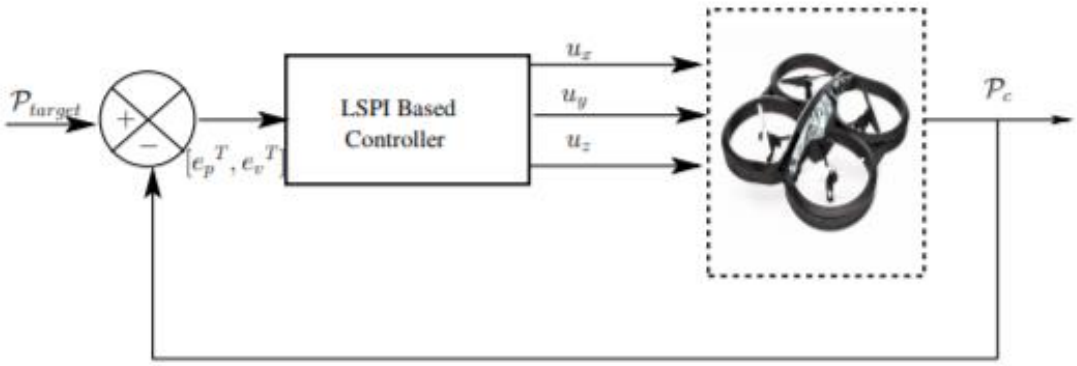
## Intuition and approach:

Our main objective is to design a controller which **minimizes the error between the current and target states for a quadcopter**. Most of the conventional controllers solved this problem but they fail in noisy environments and they need an accurate model information while designing the controller. It is typically difficult to model commercial drones which are available in the market as the end user will not be having access to the internal model and the parameters. To address this issue, we are proposing a model free RL based controller where the model information is not required in designing the controller. The effectiveness of the proposed controller is demonstrated by solving an autonomous landing task with a commercial drone on a targeted landing site.




Considering a target position $P_{target}$, target velocity $v_{target}$, current position $P_{current}$, and current velocity $v_{current}$ and a small threshold *thresh*, a safe or successful landing or target is reached if $|P_{error}| < thresh$, where $P_{error} = [P_{target}, v_{target}] - [P_{current}, v_{current}]$.

To achieve this efficiently, the quadcopter needs an optimal set of actions to minimize error between the current state and the target. The problem is formulated with a reinforcement learning based method named least squares policy iteration (LSPI). The RL method takes instantaneous error in position and velocity as the input and estimates control velocities for the quadcopter as shown in the figure below:




Here, in this figure we observe that the error in the pose/state of the drone is being fed as an input to the LSPI controller that in turn generates the appropriate linear velocities and the motor speeds.

RL methods can be categorized into model-based and model-free methods. In the former, given the state, appropriate actions will be taken by searching and planning in the constructed world model. While on the other hand, model-free methods use experience to directly learn state/state-action value function to derive optimal policies **without constructing the model** of the world.

## Least Squares Policy Iteration

Quadcopter landing or generating its trajectory can be considered as an MDP (Markov Decision Process) where the drone, at some state, S, takes an action A to reach the target and gets a reward R to subsequently reach to a state S' with a probability of P(s'|s, a).

The objective is to find a policy to maximize the expected sum of rewards and we end up computing state-value function Q(s,a).

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \,\epsilon\, \mathcal{S}} P(s'|s,a) \max_{a' \,\epsilon\, \mathcal{A}} Q(s',a')$$

The optimal value function is the one that provides maximum value across ALL GIVEN STATES and ACTIONS! The optimal policy π∗ can be extracted over $Q^*$ (s, a):

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

$$\pi^*(s) = arg \max_{a \,\epsilon\, A} Q^*(s,a)$$

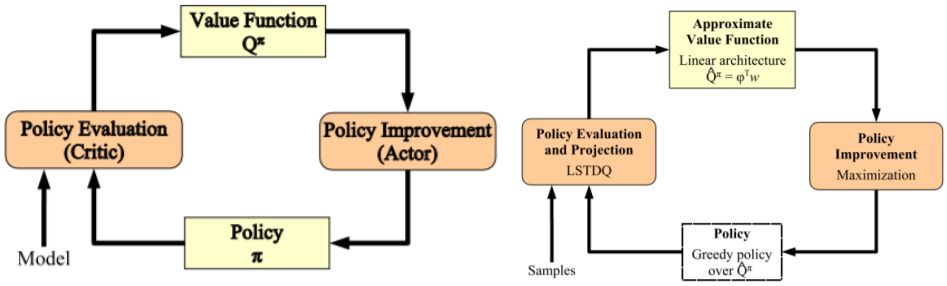
Instead of calculating an optimal value function, LSPI method tries to approximate the value function by parametrizing it into basis functions as per the equation below:

$$Q(s,a) \approx \hat{Q}(s,a;W) = \sum_{i=1}^{k} \phi_i(s,a) w_i$$

Where $phi_i$ is the $i^{th}$ basis function and $w_i$ is its corresponding weight.

The main idea of this approximation is to tune the free parameters W to reach closer to the actual state action value function so that it can be used in place of Q(s, a) while learning the optimal policy.

LSPI can be summarized by comparing the figures below:




The figure on the left side is the general policy iteration algorithm of the Q function, whereas figure on the right represents the value approximation of the state-action value function in order to evaluate the policy (step1) and the improve the same(step 2) until the policy converges.

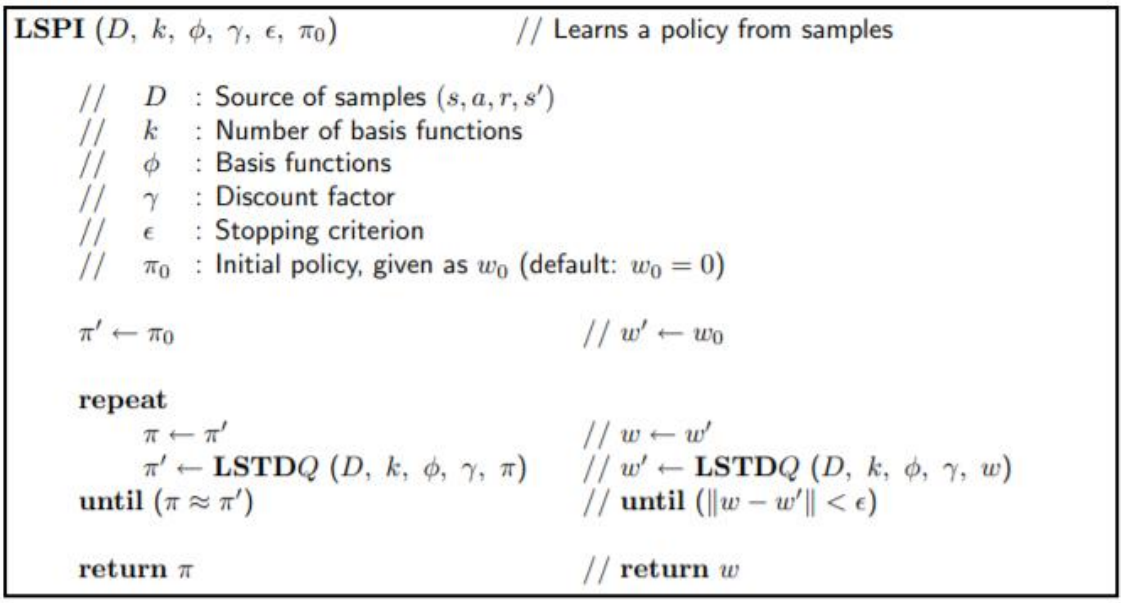
For finite action spaces this is straightforward, but for very large or continuous action spaces, explicit maximization over all actions in A may be impractical. In such cases, some sort of global optimization over the space of actions may be required to determine the best action. Depending on role of the action variables in the approximate state-action value function, a closed form solution may be possible.

Finally, any policy π (represented by the basis functions ϕ and a set of parameters w) is fed to LSTDQ (Least Squares Temporal Difference for Q function) along with a set of samples for evaluation. LSTDQ performs the maximization above as needed to determine policy π for each s 0 of each sample (s,a,r,s') in the sample set. LSTDQ outputs the parameters w π of the approximate value function of policy π and the iteration continues in the same manner.

## Implementation:

We have followed the following steps to train the model and generate the trajectories:

The algorithm can be referred to in the following image:




1) Collected samples from a random policy. Samples were of the form: (s,a,r,s')
2) Each state was represented as error in position and error in velocity.
3) Collected samples (s,a,r,s') were fed to the LSPI algorithm to calculate the weights.
4) Once the weights were calculated:
    a. Given an initial point, position and velocity error was calculated that acted as a state to be fed in the Q function to get an appropriate action.
    b. The process can be repeated till the target is achieved.
5) Rewards:
    a. Positive rewards if z-axis position is decreasing and negative otherwise.
    b. Positive if the linear velocity along z-axis is negative
    c. Large reward if the distance between the current position and the target position is less than a particular threshold.
6) The action space was discretized and a specific set of motor speeds were being fed as the inputs.

## Results:

1) As the action space was not continuous, the trained weights were not able to generate a proper trajectory

2) Training was done for few target points and may be due to that the solution could have been an overfitted one.
3) As more target points were added, a better trajectory was getting generated but that was not optimal again.

## Conclusion and Future work:

Some policy gradient methods, such as actor-critic, DDPG, PPO can be implemented on further to take care of the continuous action space.

Optimization of the approximation can be done to get the continuous action corresponding to the optimal policy.