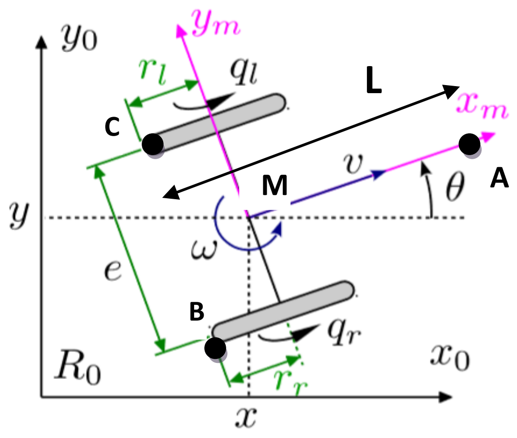
Parag Khanna S4537931
EMARO

DEEP Q Learning for Continuous World & Real Robot Simulation

Problem Formulation:

1. **Agent**:
   A [2, 0] mobile Robot was considered with the following Kinematic Model:



$$
\begin{aligned}
v &= (r_r\dot{q}_r + r_l\dot{q}_l)/2 \\
\omega &= (r_r\dot{q}_r - r_l\dot{q}_l)/e \\
\dot{x} &= v\cos\theta \\
\dot{y} &= v\sin\theta \\
\dot{\theta} &= \omega
\end{aligned}
$$

(Assumed No Slipping/ Pure Rolling)

Robot Sizes:
Radius of wheels= $r_l = r_r = 0.5$
Distance between centre and wheels = e/2 = 1
Distance from Back to Front = L = 3
Robot as Triangle:
Formed by Position of A, B, C (extreme points) as shown above, equations:

Tranformation Matrix relating World Frame with Robot Frame:	Vertices in Robot Frame (Homogenous Coordinates)		

${}^0T_m$=[cos(θ) -sin(θ) X
sin(θ) cos(θ) Y
0 0 1]
	A=
	[1 0 2*L/3;
	0 1 0;
	0 0 1]
		B=
		[1 0 -L/3;
		0 1 e/2;
		0 0 1]
			C=
			[1 0 -L/3;
			0 1 -e/2;
			0 0 1]

Vertice V in World Frame: ${}^oT_m$*V.
Robot as Point M represented by ‘ * ’

Actions:

Differential Drive was considered; hence the actions corresponded to setting speeds of the Left-Right wheels of the Robot.

2. Environment:

The state space involved robot moving in a 50*50 sized area with a goal(Square) and various mines(circles) placed around as shown:

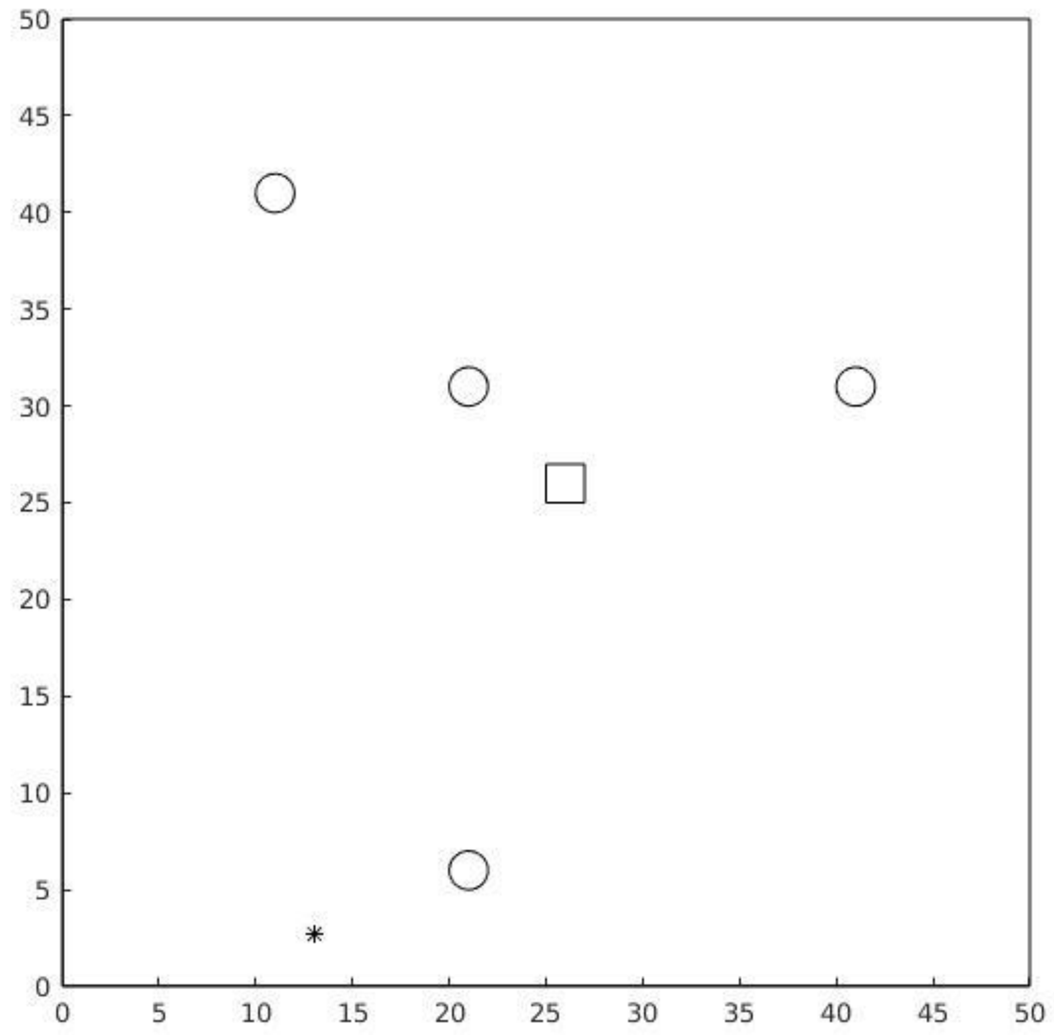


Figure 1 489x489 (actual) sized State Space

Deep Q-Learning:

Unlike Q-learning where Q(s,a) for a State-Action pair is stored in a table, Deep Q-learning involves predicting the Q-value for all actions in a given State by using a (Deep)Neural Network. If the State is represented as an Image then a Convolution Neural Network can be used.
Thus, it is essentially a **Regression problem** for the CNN, also called Deep Q-Network (DQN) in this use, **where DQN predicts the Q-values for all actions corresponding to the Input State Image.**

1. **Input to DQN :**
   - The State Image was aquired from MATLAB figure by:
     F = getframe; [I,Map] = frame2im(F);
     Thus, the MATLAB program involved initiating a figure initially with the Goal [**Square**],Mine(s) [**Circle**] and Robot(**asterisk/triangle**) and changing the Robot Position on the go.
   - The Size of the figure is an important constraint to be maintained (as this has to be the size of first layer of DQN). This size also determines **the size of the State Space.**
     A while loop as shown was used to enforce this size, using manually set '*Position*' property of *'figure'* function.

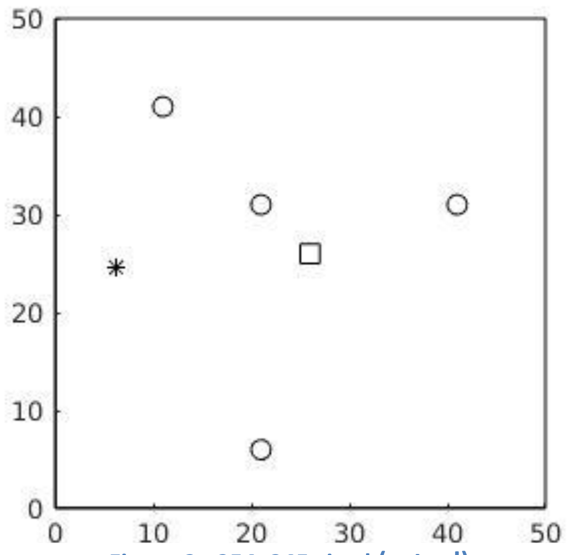


Figure 2 : 254x245 sized (actual)

```
while(1) %forcing to get a image of 245x245
close all;
figure('Position',[0 0 316 300]);
F = getframe;
[I,Map] = frame2im(F);
    if(size(I,1)==245 && size(I,2)==245)
        break;
    end
end
```

The 254*254 size was chosen for experiments (only limited by GPU memory available)
(throws 'Out of memory on device' error for GPU for 489x489)

- Once the above figure was set, Robot was to be moved in it. So the Robot/robot triangle was initialized as a 'Graphics Object' and 'Set' used to change the Property i.e. modify its position later.

  ```
  p = plot(X(1),X(2),'k*');  set(p,'YData',X(1));  set(p,'YData',X(1))

  p=plot(tri(1,:), tri(2,:), 'linewidth', 1,'color',[0 0 0]);
  [X,tri]=stateUpdate_getRoboTriangle_DQN(X,act_st)
  set(p,'XData',tri(1,:));;
  set(p,'YData',tri(2,:));
  ```

- The Image thus acquired correspond s to the State at each step and fed to DQN as input.

2. **Defining the CNN/DQN architecture:**
   This used MATLAB provided '*layers*' to create layers for Regression (Following Example) and '*options*' to define training option for the Network

   ```
   layers = [     imageInputLayer([245 245 1],'Name', 'input')
   convolution2dLayer(16,8,'Padding',1,'Name', 'conv1')
    batchNormalizationLayer('Name', 'Batch_N')
   reluLayer('Name', 'relu1')
      %maxPooling2dLayer(2,'Stride',2,'Name', 'maxPool1')
   convolution2dLayer(8,4,'Padding',1,'Name', 'conv2')
   batchNormalizationLayer('Name', 'Batch_N2')
    reluLayer('Name', 'relu2')
   convolution2dLayer(4,4,'Padding',1,'Name', 'conv3')
   batchNormalizationLayer('Name', 'Batch_N3')
   reluLayer('Name', 'relu3')
   fullyConnectedLayer(8,'Name', 'FullyC2')% size is eual to number of actions
   regressionLayer('Name','Output')];
   ```

   (Size of Layers inspired from Atari Paper, Source)
   No Average/Max Pooling Layers since the Spatial Information is to be retained

*options = trainingOptions('sgdm', 'Momentum',0.3, 'MaxEpochs',4, ...
'InitialLearnRate',0.0001, 'Verbose',false, 'Plots','training-progress');*
Each convolutional layer was followed by a Batch Normalisation Layer and rectified linear unit (ReLU) Layer. The output layer was Regression layer which was preceded by a Fully-connected Layer with as many neurons as the **Number of Actions considered.**
Back-propagation and mini-batches stochastic gradient descent (**sgdm**) was used update the network as defined under *options*.

3. **Initializing the DQN weights:**
   Random State-Action-Reward tuples were generated (by Simulation) before Learning Process. For a given State, setting obtained rewards as targets for the action taken and non-terminal reward as target for all other actions, the DQN was trained by using *Options* shown above.
   It is important to verify that RMSE as shown by *'Plots', 'training-progress'* do not go very high at this step, otherwise this may lead to Network 'Exploding' i.e. very high output values.

4. **Q-Learning with DQN: (Sources: 1, 2, 3)**
   The Q learning procedure followed was:

   - Acquire Current State Image and do a Forward pass through DQN to get all possible Q-values. *Q_st=predict(net,state(:,:,:,1));*
   - Select and perform the action with highest Q-values or select a random action (With Probability **Epsilon***), i.e. Update Robot Position, acquire new State Image and get Reward(r).
   - Do a Forward pass for new state and set the targets:
     Q_st_new= predict(net,state_new);
     i) For the action taken:
        -If New State is terminal State(Goal/Mine):
        target=r;
        -else
        target=r + gamma*max(Q_st_new)
        [where parameter *'gamma'* is to be set for Future rewards(Maximum

Q-value predicted for New State), corresponding to BELLMAN Equation]

ii) For all other actions:
target=Q_st(action)
[i.e. Target is same as the value predicted for the action by DQN]

- Train the DQN with the State-Image and Targets:

  layersTransfer = net.Layers(1:end); %transferring all previous layers(with weights)
  options = trainingOptions('sgdm', 'Momentum',0.3,MaxEpochs',1, 'InitialLearnRate',0.0001,'Verbose',false, 'Plots','none');
  %to supress plot occuring at each step when CNN is trained net = trainNetwork(state,target,layersTransfer,options);
  [Slightly Modified options here, no plots and only one Epoch used]

***Epsilon-greedy approach** to add some random action accounting for Exploration-Exploitation issue. A **Decaying Epsilon** was used.*

----- *Can be seen in "DeepQN.m"*------------------------------------

Problem with above:
Since the Robot movement is such that it leads to many similar looking states (locally), this guide the DQN to learn just a replay of the episode. To avoid this, an **Experience Replay (Memory)** was created where in the tuple.
(State,Action,Reward, State_new) was stored at every step of episode. This was made as a FIFO queue so that the oldest tuple is removed first and queue is filled with Newer Experiences. Also this is initially populated with Randomly Generated Experience Tuples. (*see add_in_memory.m*)
Memory-size taken=50
*In later experiments **a prioritized experience replay** was used, which always stored 10 terminal state associated tuples and 40 other tuples. (see add_in_memory_priority.m)*
At each step a Mini-Batch(size taken =5) of tuples from the Replay Memory was siphoned, and used to train the DQN(Doing Forward Passes and finding targets as

previously shown).
----- Can be seen in “DeepQN_with_ExpReplay.m”------------------------------------

Miscellaneous Points:

1. **Reward:** Currently a fixed reward of 250 for reaching Goal(Goal Radius=4), 1 for reaching the Mine(Mine Radius =2), and 2 for all other position. Considering a maximum step number of 100, it was ensured that Maximum reward is reached when reaching the Goal only.
This wasn’t put as generic [-100,-1, +100] for [Goal, Action, Mine] since the symmetry around zero of targets can easily cause DQN to explode (Predict very high or NaN values)

2. **Learning Parameters**: Two learning parameters to be set:
   - *gamma* for Bellman Equation used in Targets above.
   - *Initial Learning Rate* for the DQN in Training-*options***. :** This was carefully selected after trials. Less value causes slow learning while high value causes DQN to explode.

3. **Keeping Check for NaN values**: DQN can explode, particularly at beginning due to lack of sufficient input data/ random data fed for initialization.

4. **Robot Representation:**
   Clearly the Robot as an **‘asterisk’** doesn’t provide any information about the Robot’s Orientation information. Thus, the **Triangle** representation *shall* be better for providing information in the State-Image information, so was used for later experiments.

Results Obtained:

- The Learning results till now have been particularly good for some of the fixed Initial positions, e.g. [20, 20, pi/4], for which the robot successfully reaches the goal.
- After learning, the Q-values seem to favour one of the actions.

Results for Particular Initial States in Detail:

1. **Init = [10, 10 , pi/4]; Robot as Asterisk, without experience replay.:**
   Robot learns to reach goal, avoiding mines.
   Learning Parameters: Max_episodes =20; Max_steps=100; gamma=0.0002;
   Training Options: MaxEpochs=1; InitialLearnRate=0.001;
   Action Space: 4 actions - [3,3];[0,3];[3,0];[-3,-3]

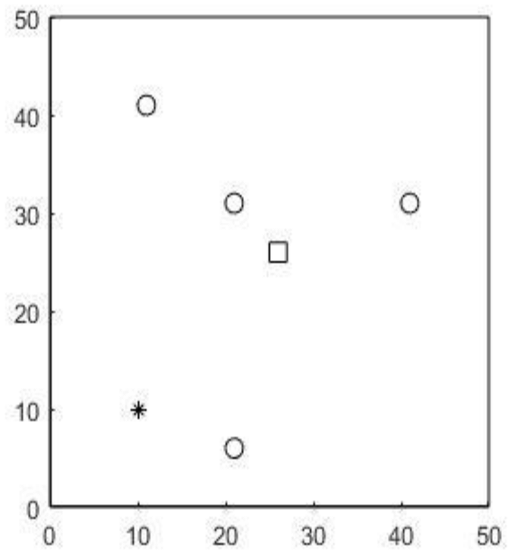


Figure 3 :245x245 State Space, Robot at Initial Pos

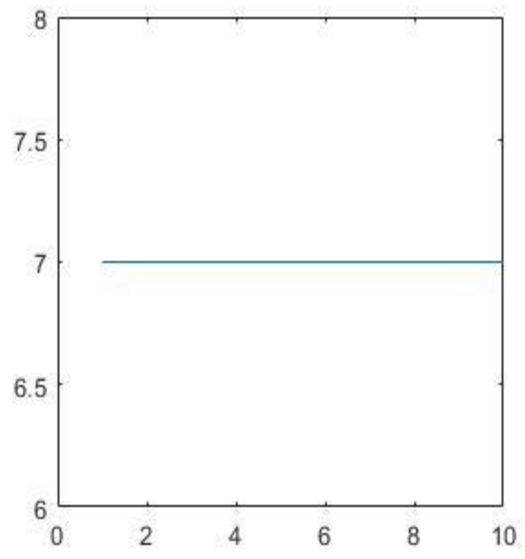


Figure 4 : Validation Plot: Number of Steps to Goal per Episode

DQN Architecture Used:

Layer Type	**Size**	**No. of Filters(if any)**
Image Input Layer	[245 245 1]	--
Convolution 2D layer-1*	8(filter size)	8
Convolution 2D layer-2	4	4
Convolution 2D layer-3	4	4
Fully Connected Layer	4(equals No. of actions)	--
Regression Layer	--	--

* Each convolution layer is followed by *batchNormalizationLayer and reluLayer* [Also applies for further discussion]

See: ‘DQN_withoutEXP_fixedStart_10_10_pi4’ folder

2. **Init = [10,10,pi/4]; Robot as Triangle, with Experience Replay.:**
   Robot learns to reach goal, avoiding mine.
   Learning Parameters: Max_episodes =30; Max_steps=100; gamma=0.5;
   Training Options: MaxEpochs = 3; InitialLearnRate=0.0001;
   Action Space: 8 actions - [3,3];[0,3];[3,0];[-3,-3];[2,2];[-1,3];[3,-3];[3,-1]

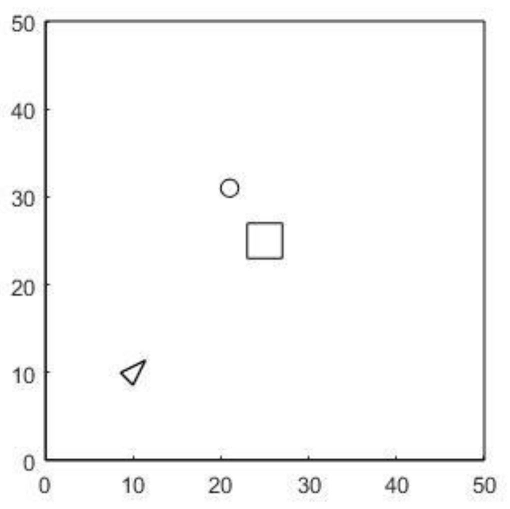


Figure 5 : 256x256 State Space, Robot at Initial Pos

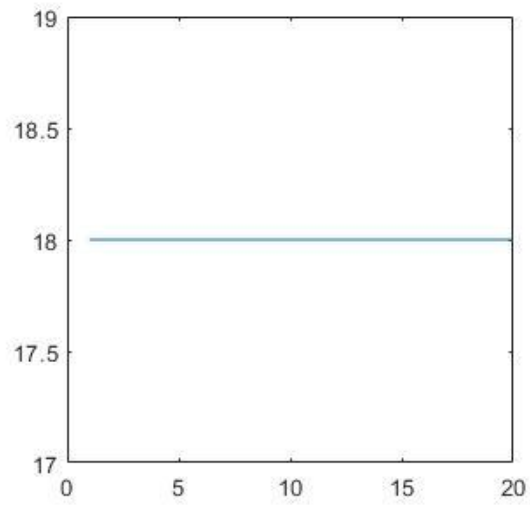


Figure 6 : Validation Plot: Number of Steps to Goal per Episode

DQN Architecture Used:

Layer Type	Size	No. of Filters(if any)
Image Input Layer	[256 256 1]	--
Convolution 2D layer-1	16(filter size)	8
Convolution 2D layer-2	8	4
Convolution 2D layer-3	4	4
Fully Connected Layer	8(equals No. of actions)	--
Regression Layer	--	--

See:' DQN_withEXP_fixedStart_10_10_pi4_latestVersion' folder

3. **Init = [15,20,pi/4]; Robot as Triangle, with Experience Replay.:**
   Robot learns to reach goal, avoiding mine.
   Learning Parameters: Max_episodes =20; Max_steps=100; gamma=0.2;
   Training Options: MaxEpochs = 1; InitialLearnRate=0.0001;
   Action Space: 8 actions - [3,3];[0,3];[3,0];[-3,-3];[2,2];[-1,3];[3,-3];[3,-1]

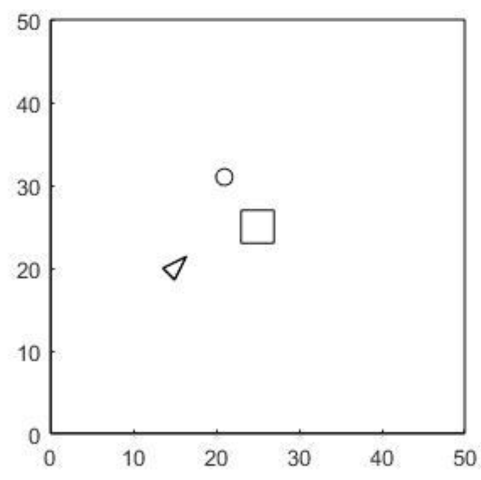


Figure 7 : 256x256 State Space, Robot at Initial Pos

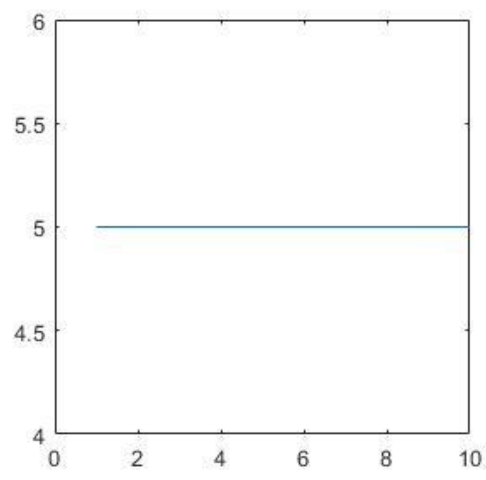


Figure 8 : Validation Plot: Number of Steps to Goal per Episode

DQN Architecture Used:

Layer Type	Size	No. of Filters(if any)
Image Input Layer	[256 256 1]	--
Convolution 2D layer-1	16(filter size)	8
Convolution 2D layer-2	8	4
Convolution 2D layer-3	4	4
Fully Connected Layer	8(equals No. of actions)	--
Regression Layer	--	--

See:' DQN_withEXP_fixedStart_15_20_pi4' folder

4. Init = [20, 20, pi/4]; Robot as Triangle, with Experience Replay.:

Robot learns to reach goal, avoiding mine.

Learning Parameters: Max_episodes =30; Max_steps=100; gamma=0.5;

Training Options: MaxEpochs = 3; InitialLearnRate=0.0001;

Action Space: 8 actions - [3,3];[0,3];[3,0];[-3,-3];[2,2];[-1,3];[3,-3];[3,-1]

DQN Architecture Used:

Layer Type	Size	No. of Filters(if any)
Image Input Layer	[256 256 1]	--
Convolution 2D layer-1	16(filter size)	8
Convolution 2D layer-2	8	4
Convolution 2D layer-3	4	4
Fully Connected Layer	8(equals No. of actions)	--
Regression Layer	--	--

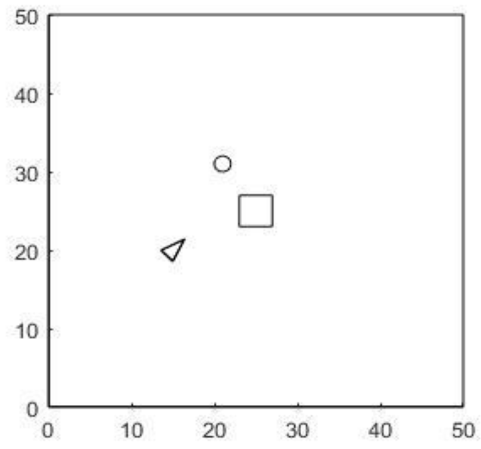


Figure 7 : 256x256 State Space, Robot at Initial Pos

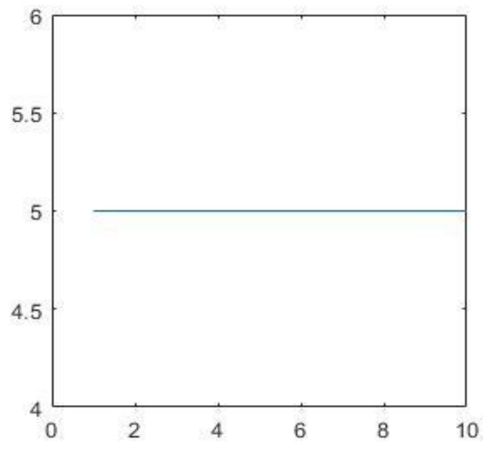


Figure 8 : Validation Plot: Number of Steps to Goal per Episode

See:' DQN_withEXP_fixedStart_20_20_pi4_latestVersion' folder

5. For other Initial States such as [10,10,pi/2], [40,20,pi/4], etc. The robot fails to learn to reach the goal state.
   5.1 It is seen that learning favours one of the action, hence robot performs it repeatedly(turning around at its position or moving forward/backward only)
   5.2 It is worth to be noted that the learning results are *based on the training of 30 episodes only*.(due to limited time and hardware)

Possible Improvements that can be done:

1. **Change in Rewards:** The Procedure can be tried with a +ve Gaussian/Probability Distribution around Goal and –ve Probability Distribution around Mine to cause a varying reward field.
2. **Change in Layers**: The size of layers are limited by GPU, some variations can be tried with larger filter sizes for initial convolution layers. Some detailed analysis can be done in regards to what desired features shall be aimed to identify in convolution layers. *It is also to be noted that in later experiments, similar results were obtained with single convolution layer of 16 sized 8 filters*. Also there can be a larger fully connected layer in between last convolution layer and last fully connected layer.
3. **Longer and more Training episodes:** Maximum steps of an episode and the number of episodes in the training period shall be increased, with more exploration of the state space.