Artificial Intelligence-2 Assignment 3

A report submitted in fulfillment of the requirements for Assignment Project

by

Prasenjeet Roy(MP19AI001)
Neelu Verma (MP19AI002)
Nandini Saini (MP19AI003)
Rishabh Ranjan (MP19AI004)
Shraban Kumar Chaterjee(MP19AI005)
Asha Rani (MP19CS002)
Shubham Gupta (MP19CS003)
Pratik Somvanshi (MT19DCS010)

Date of submission : 1 May 2020

Submitted to
Dr. Debarati Bhunia Chakraborty

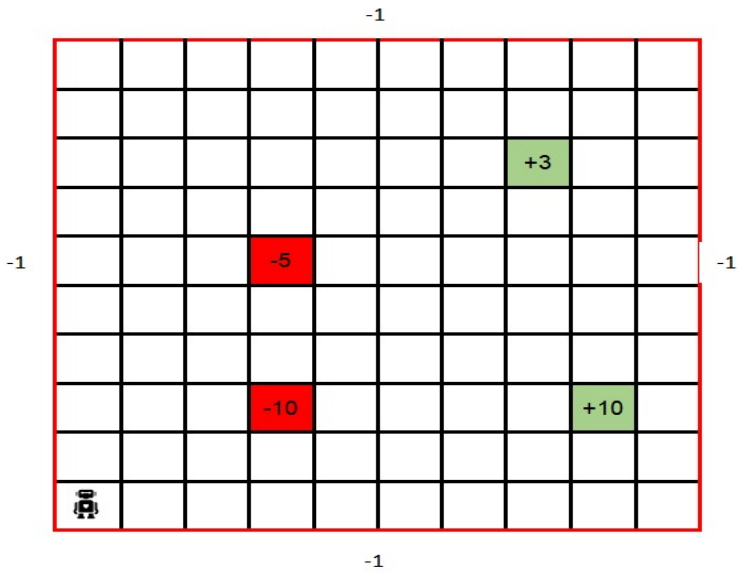
Indian Institute of Technology, Jodhpur

TABLE OF CONTENTS

CHAPTER 1

Problem Statement

Based on the paper analysis, we tried to implement the Robot control gestures in the grid world. For this, we divided this problem into two tasks. Firstly the Hand Gesture Control which performed using OpenCV for the movements of an agent(robot), second task POMDP implementation in the grid world problem, and applied value iteration algorithm to maximize reward value. Application of POMDP used on a 10X10 grid, which has four reward states, and the agent needs to choose four actions from go left, go right, go up and stop. Each block on the grid represents the state, and positive reward states considered as terminal after reaching those states the agent will not receive any reward. If the agent(robot)is moving in the desired direction, the probability will be 0.7 otherwise 0.1 if walking in the other three directions. In progress, we are trying to include both things together.



CHAPTER 2

SOLUTION APPROACH

2.1 Underlying Assumptions

For the gesture recognition ,Libraries are required

1. cv2
2. imutils
3. numpy
4. sklearn

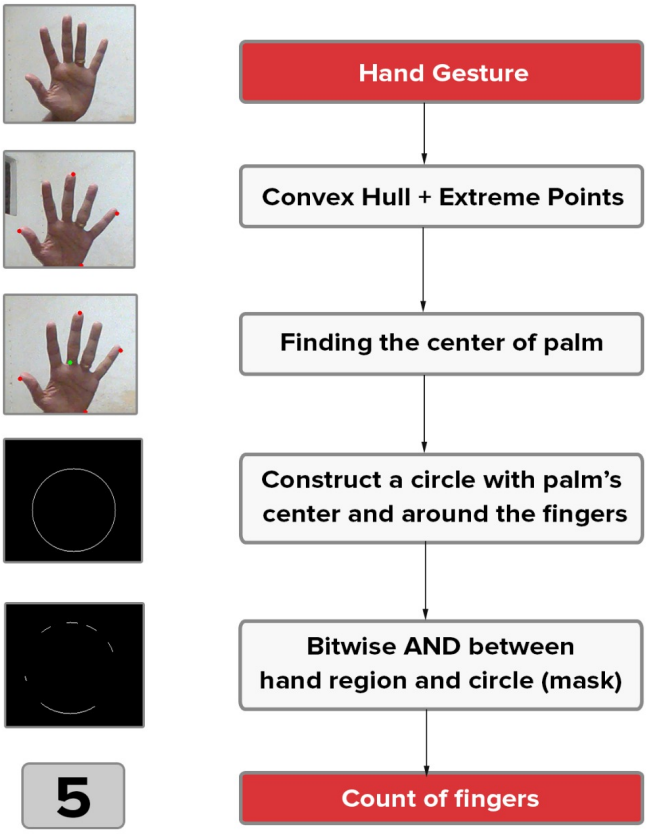
For the POMDP implementation ,we used

1. Jupyter with Julia Kernel
2. Dependencies are : POMDPs.jl interface for POMDP toolbox and SparseCat to handle Transition probabilities
3. sub2ind() function used for indexing between functions
4. Sim function from POMDP toolbox used for simulation

2.2 Data Collection

We wrote a recognize.py file for recognizing the number of fingers through the webcam. We use the following mapping of the number of fingers to the actions:

- 1 finger - Go left
- 2 fingers - Go right
- 3 fingers - Go up
- 4 fingers - Stop



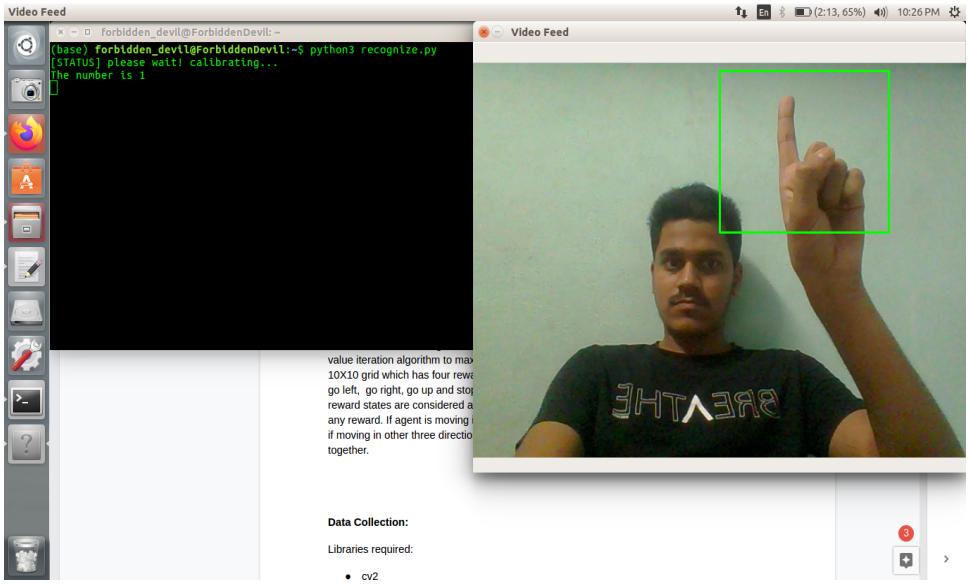


Figure 2.1: Image for Go -left using one finger

2.3 Implementation and Setting of Parameters

2.3.1 States

The state of agent in grid world problem defined as :

- X position : x(integer)

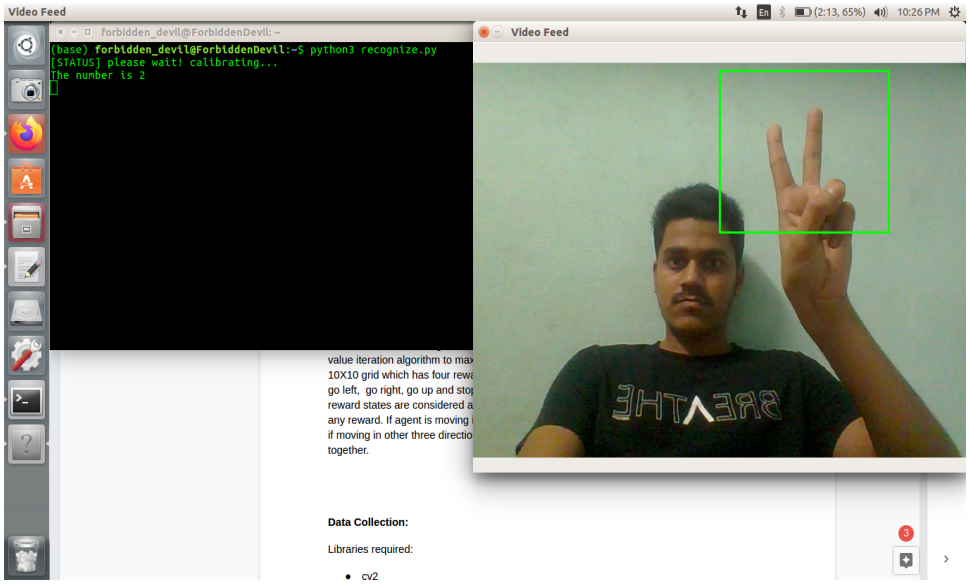


Figure 2.2: Image for Go -right using two finger

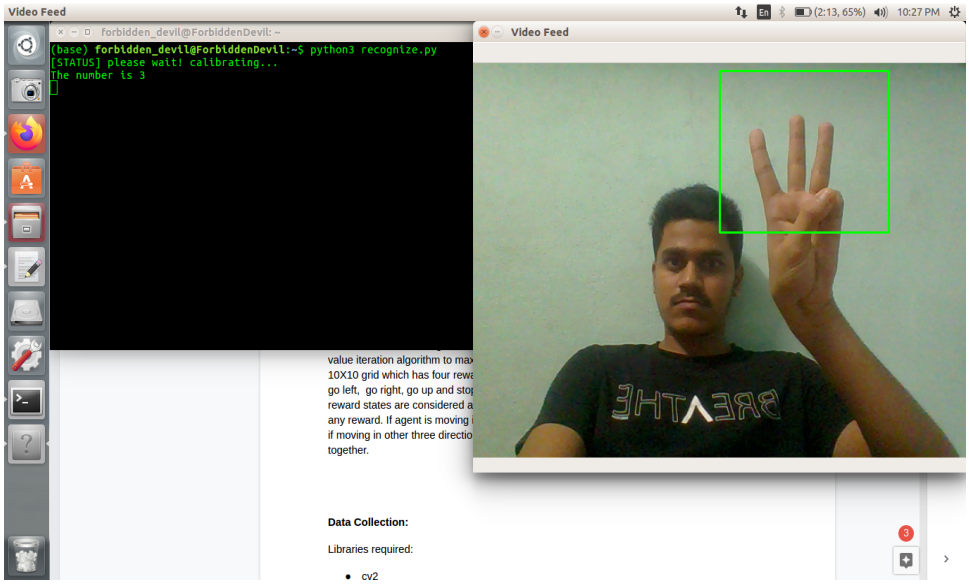


Figure 2.3: Image for Go -up using three finger

- Y position : y (integer)
- Terminal state : True or False (Boolean)

We assume that initial position will (x,y,False).

2.3.2 Actions

There are four action defined for agent(robot). A ={go_left,go_right,go_up,stop}Each action is operated using hand gestures to corresponding to each command.

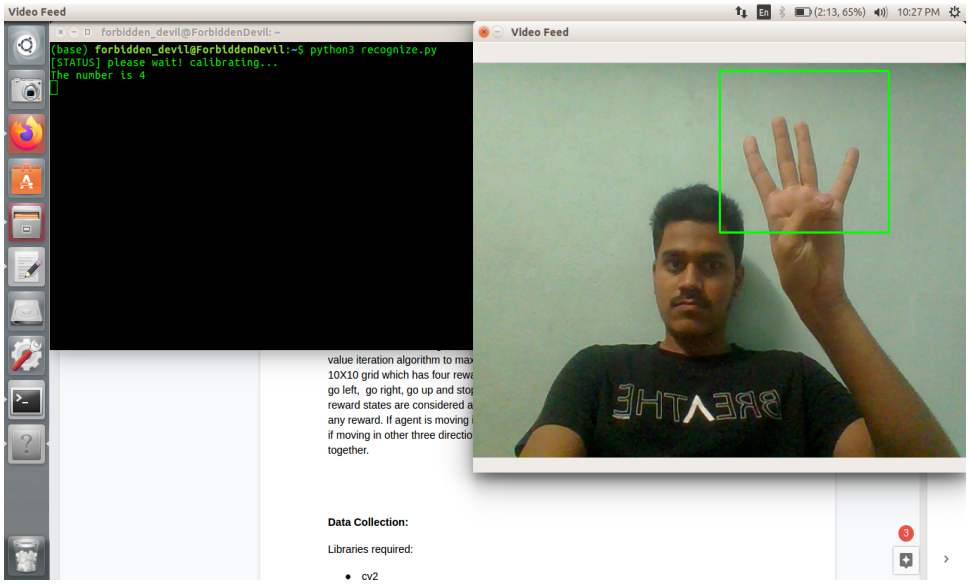


Figure 2.4: Image for stop using four finger

2.3.3 POMDP Tuple

The POMDP tuple defines as {S,A,T,R} where
S = State space
A= Action space
T = Transition Probability
R = Reward value In our problem,we initialized these parameter's :

- x size of the grid(size_x) : 10
- y size of the grid(size_y) : 10
- the states in which agent receives reward (reward state): (4,3),(4,6),(9,3),(8,8)
- reward values for those states(reward values) : [-10,-5,10,3]
- probability of transitioning to the desired state : 0.7
- disocunt factor: 0.9

2.3.4 Transition Model

Transition function $T(s' \mid s, a)$ captures the dynamics of the system. Specifically, $T(s' \mid s, a)$ is a real value that defines the probability of transitioning to state s' given that you took action a in state s. The transition distribution $T(\cdot \mid s, a)$ is the actual distribution over the states that our agent can reach given that its in state s and took action a. In other words this is the distribution over s'. For this ,we used SparseCat from POMDP Toolbox.A **SparseCat** object contains a vector of states and an associated vector of their probabilities. The probabilities of all other states are implied to be zero.

2.3.5 Reward Model

The reward model $R(s, a, s')$ is a function that returns the reward of being in state s, taking an action a from that state, and ending up in state s'. In our problem, we are rewarded for reaching a terimanl reward state ,this could be positive or negative.

2.3.6 Simulation

For see the working of problem,simulation of problem is done by **Sim function**from POMDP toolbox.It provides a convenient do block syntax for exploring the behavior of the mdp. The do block receives the state as the argument and should return an action.

2.3.7 Value Iteration Algorithm

Value iteration is a dynamic porgramming apporach for solving MDPs.In value iteration, which is also called backward induction, the π function is not used; instead, the value of $\pi(s)$ is calculated within V(s) whenever it is needed. Substituting the calculation of $\pi(s)$ into the calculation of V(s) gives the combined step:

$$V_{i+1}(s) := \max_a \left\{ \sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V_i(s') \right) \right\},$$

where i is the iteration number. Value iteration starts at i = 0 and V_0 as a guess of the value function. It then iterates, repeatedly computing V_{i+1} for all states s, until V converges with the left-hand side equal to the right-hand side

CHAPTER 3

RESULTS

Using POMDPToolbox we imported sim(::MDP) for simulations for exploring the behavior of the mdp. In the do block, it takes the state as the argument and returns an action as output. Hence, it acts as a "hook" into the simulation for defining the quick ad-hoc policies.

```
state is: GridWorldState(4, 1, false)
moving right
state is: GridWorldState(5, 1, false)
moving right
state is: GridWorldState(6, 1, false)
moving right
state is: GridWorldState(7, 1, false)
moving right
state is: GridWorldState(8, 1, false)
moving right
state is: GridWorldState(9, 1, false)
moving right
state is: GridWorldState(10, 1, false)
moving right
state is: GridWorldState(10, 1, false)
moving right
state is: GridWorldState(10, 1, false)
moving right
state is: GridWorldState(10, 1, false)
moving right
```

Figure 3.1: simulation result

A dynamic programming approach called Value Iteration Solver is implemented for solving MDPs. The POMDPs.jl solver provides two data types for interaction- the Solver type and the Policy type.

```
[Iteration 1   ] residual:       10 | iteration runtime:     0.493 ms, (  0.000493 s total)
[Iteration 2   ] residual:      6.3 | iteration runtime:     0.570 ms, (   0.00106 s total)
[Iteration 3   ] residual:     4.54 | iteration runtime:     0.490 ms, (   0.00155 s total)
[Iteration 4   ] residual:     3.39 | iteration runtime:     0.506 ms, (   0.00206 s total)
[Iteration 5   ] residual:     2.57 | iteration runtime:     0.489 ms, (   0.00255 s total)
[Iteration 6   ] residual:     1.92 | iteration runtime:     0.491 ms, (   0.00304 s total)
[Iteration 7   ] residual:     1.39 | iteration runtime:     0.496 ms, (   0.00354 s total)
[Iteration 8   ] residual:     1.07 | iteration runtime:     0.492 ms, (   0.00403 s total)
[Iteration 9   ] residual:    0.861 | iteration runtime:     0.499 ms, (   0.00453 s total)
[Iteration 10  ] residual:    0.662 | iteration runtime:     0.491 ms, (   0.00502 s total)
[Iteration 11  ] residual:    0.489 | iteration runtime:     0.499 ms, (   0.00552 s total)
[Iteration 12  ] residual:    0.405 | iteration runtime:     0.530 ms, (   0.00605 s total)
```

Figure 3.2: Result After applying value iteration algorithm

To know the action, provide the respective policy with state as action parameters.

```
s = GridWorldState(9,2)
a = action(policy, s)
```

```
:up
```

Figure 3.3: Corresponding Action on that state

```
s = GridWorldState(8,3)
a = action(policy, s)
```

```
:right
```

Figure 3.4: Corresponding Action on that state

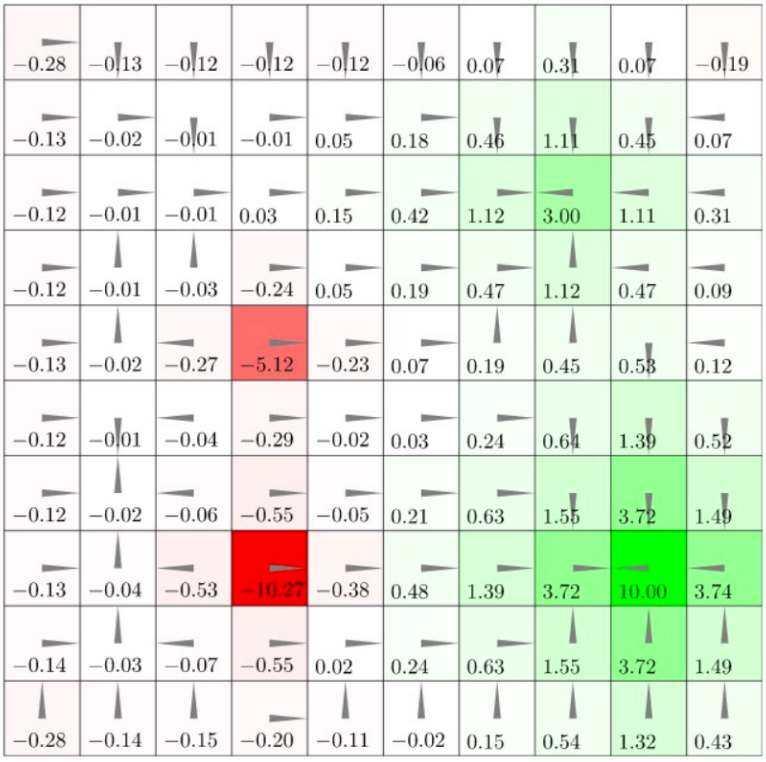


Figure 3.5: Each state with accumulated reward value

CHAPTER 4

ANALYSIS

- After reading and analyze the paper,we tried to learn POMDP implementation,value iteration algorithm and also got the knowledge of importance of Artificial intelligence in game theory.
- In process of implementation of above described problem ,we gave our best try to implement each module.