Project Human - Robot collaboration and teamwork

Perspective taking and adaptive decision-making in HRI

User Manual

Qichao Xu

Gazelle Zaheer

Maximilian Thorand

August 1, 2017

Contents

1 Introduction

This manual shows you how to run HRC simulation scenarios, where human and robot can interact and collaborate with each other in the task of grasping an object from a conveyor belt. It enables you to visualize and investigate how robot observes the behaviours of human and makes his decision. It also helps you to understand how the collaboration between human and robot works. The simulation scenarios are created on morse platform. For the communication between different components, we use webSocket and ROS service.

The behaviours of human are based on a MDP(Markow Decision Process) model. By using MDP, we can simulate different behaviours for humans. We provide different MDP model files that users can choose from. In the robot side, how robot understands the behavious of human and makes correct response are a tough task. We use a POMDP(Partially Observable Markov Decision Process) model to simulate the decision-making procedure for robot.

This manual works as follows. First of all we will introduce our tool with a getting started guide, which includes how to compile the codes and how to successfully run a morse example scenario. Afterwards the theoritical background are to be explained. The manual closes with a FAQ chapter.

2 Getting started

2.1 Basic Settings

- **system:** Ubuntu 16.04 LTS
- **ROS:** kinetic-1.12.7
- **morse:** morse-1.3-Stable
- **blender** blender-2.76
- some useful links that may help you:
  1. This project can be found in:
     - https://gitlab.tubit.tu-berlin.de/aac-hrc/emphatic-hrc-boxing
  2. morse and ROS:
     - http://www.openrobots.org/morse/doc/stable/morse.html
     - http://wiki.ros.org/kinetic/Installation/Ubuntu
     - https://www.openrobots.org/morse/doc/latest/user/beginner_tutorials/ros_tutorial.html
     - http://www.openrobots.org/morse/doc/1.3/user/beginner_tutorials/hri_tutorial.html
  3. MDP/POMDP:
     - http://pomdp.org/
     - http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.Download
     - http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/uploads/Main/README-0.96.txt

2.2 Compile

2.2.1 MDP

navigate to code/despot_MDP:

```
mkdir build
cd build
cmake ..
make
```

2.2.2 POMDP

navigate to code/despot_POMDP:

```
mkdir build
cd build
cmake ..
make
```

2.2.3 ROS

navigate to code/ros_ws:

```
catkin_make
```

2.2.4 morse

The morse open source codes are not well developed and they lack a lot of functionalities that we need in this project. We complement the source codes of morse with our own codes. In order to successfully run the project, you need to add or replace some of the source codes by following the instructions below. To make sure that you don't lose original source codes, we recommend you to make a copy before replacing them.

1. navigate to code/hrc_morse/src:
   a) replace human.py:
   ```
   sudo cp human.py /opt/lib/python3/dist-packages/morse/robots/
   ```
   b) replace pr2.py:
   ```
   sudo cp pr2.py /opt/lib/python3/dist-packages/morse/robots/
   ```
   c) replace main.py:
   ```
   sudo cp main.py /opt/lib/python3/dist-packages/morse/blender/
   ```
2. navigate to code/hrc_morse/data:
   a) replace human.blend:
   ```
   sudo cp human.blend /opt/share/morse/data/robots/
   ```
   b) replace pr2.blend:
   ```
   sudo cp pr2.blend /opt/share/morse/data/robots/
   ```
   c) add a conveyor belt:
   ```
   sudo cp conveyor.blend /opt/share/morse/data/environments/
   ```
3. Now you can create a morse workspace(named as "hrc") and put our morse script file into it:
   ```
   morse create YOUR_MORSE_PATH/hrc
   ```
   navigate to code/hrc_morse:
   ```
   cp builder_script.py YOUR_MORSE_PATH/hrc/
   ```

2.3 Run an Example

In this section, we show how to run a complete HRC example scenario. All the tools, including morse, blender, ROS, MDP and POMDP, are manipulated via commands from ternimal.

1. initialize ROS:
   ```
   roscore
   ```
2. run morse scenario:
   in a new terminal, navigate to YOUR_MORSE_PATH/:
   ```
   morse run builder_script.py
   ```
3. run ROS nodes:
   in a new ternimal, navigate to code/ros_ws:
   ```
   source devel/setup.bash
   ```
   ```
   roslaunch ros_hrc hrc.launch
   ```
4. If everything runs successfully, you will see besides current three terminal windows(see Fig 2.1, Fig 2.2, Fig 2.3), there are two new terminal windows poping up: one window runs MDP(from despot_MDP, see Fig 2.4) and the other runs POMDP(from despot_POMDP, see Fig 2.5). Fig 2.6 shows the simulation scenario.

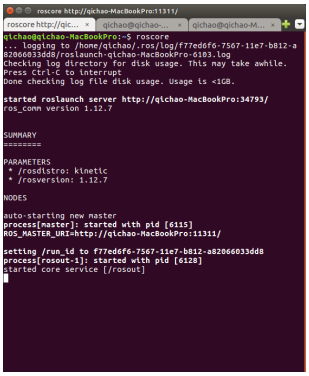

Figure 2.1: roscore

Figure 2.2: morse

Figure 2.3: ROS

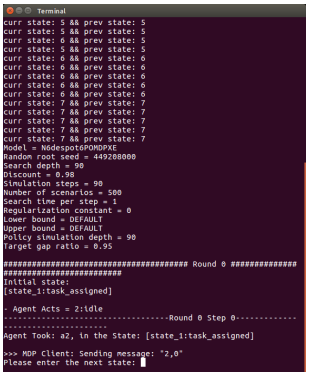

Figure 2.4: MDP

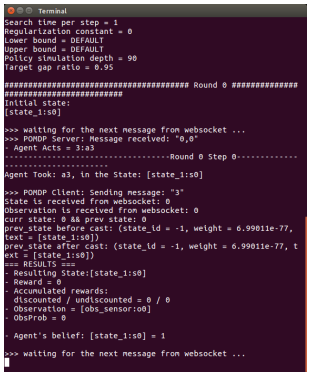

Figure 2.5: POMDP

5. You can control the behavious of human only by inputing the next state of MDP from the MDP terminal(Fig 2.4). When you input a proper

integer in the MDP terminal(there are 8 states: 0 is initial state; you can input 1-7), ROS terminal(Fig 2.3) will output the current human action, human observation, robot observation, human state, robot state, as well as robot action, where the robot action is calculated in POMDP and transfered to ROS via webSocket. As a result, this action is executed in morse.

6. At any time, you can press "Enter" in the ROS terminal(Fig 2.3) to reset the whole scenario. Before reset, you should manually close the MDP and POMDP terminal windows.

Figure 2.6: morse scenario

3 Theoretical Explanation

3.1 Architecture and Communication graph

Fig 3.1 shows the genaral architure of the project. Fig 3.2 shows how we use webSocket and ROS service to transfer information and communicate between different components(morse, ROS, MDP and POMDP).

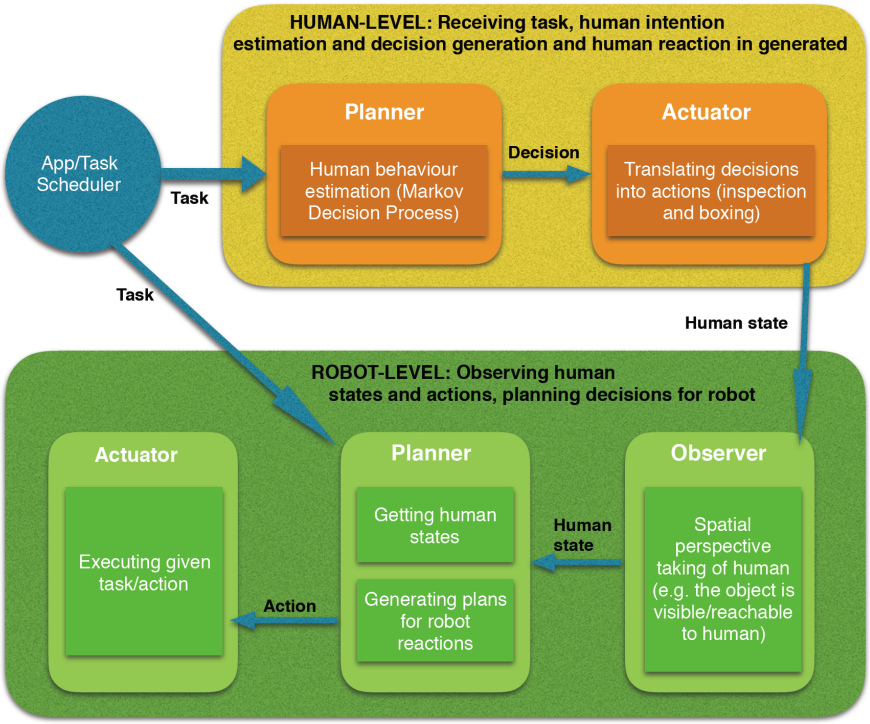

Figure 3.1: general architecture

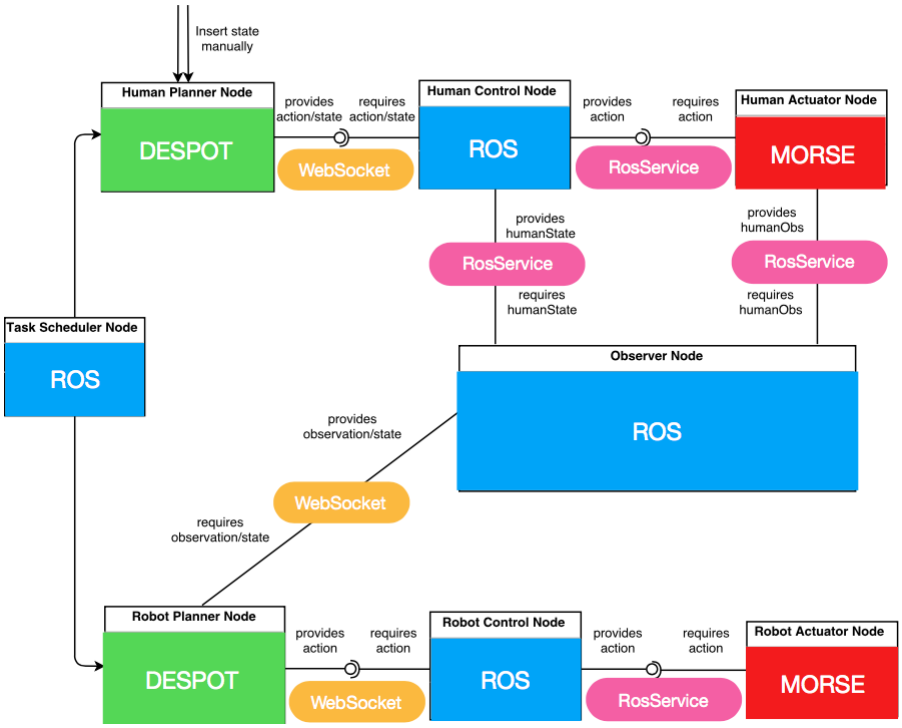

Figure 3.2: communication graph

3.2 Web-Socket and ROS Service

3.2.1 Web-Socket

This section explains how webSocket works. The reason why we use webSocket is that there are no well-developed ROS packages for MDP or POMDP models. As a substitution, we use the despot library for MDP and POMDP models. To communicate between despot model and ROS, we choose the simple and convenient tool: webSocket.
The communication between a client and a serve in webSocket is connected by a identical localhost number. For this project, the localhost numbers used in the three webSocket connections(see Fig 3.2) are 7070, 8080 and 9090. For example, a server with localhost number 7070 is opened and is waiting for a connection. Any client that wants to communicate with this server should declare "connect to server with localhost number 7070". In this way, a client and a server are connected and information can be then transfered between them. (see Fig 3.3)

3.2.2 ROS Service

ROS service is one of the most important communication methods in ROS. There are two nodes in a ROS servive connection: one acts as a client and the other as a server. The connection is specified by an identical and unique ROS service name. A service server has certain pre-defined functions and is always waiting for a request from a service client. If the request is activated,

the service server will execute the functions and return a respondse to service client. Fig 3.4 shows how ROS service works.

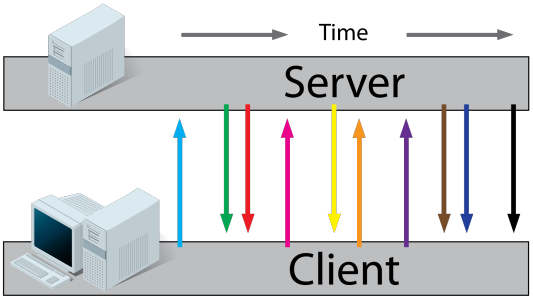

Figure 3.3: functional theory of webSocket

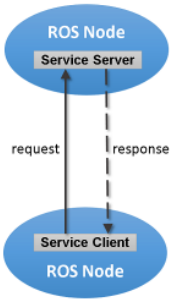

Figure 3.4: functional theory of ROS service

3.3 Morse

Morse is a generic simulator for academic robotics. It provides a lot of realistic 3D simulations of different environments and autonomous robots. In this project, in order to simulate the collaborate work between a human and robot, we utilize a human component and a P@2 robot component in morse.
The current morse platform doesn't feed the need of our project. That's why we need to modify and add some functionalities in the morse source codes. Table 3.1 lists a detailed description of all the files that we modify in the project. Table 3.2 is a list of all the ROS services that are used in this project.

file	change	purpose
human.py	modify	1. add the following functionns(declared as ROS service) to simulate the actions of human: reset; walk away; look around; warn robot; grasp object; attempt to grasp. 2. add the following variables to observe the current states of human: is the object visible to human; is the object reachable to human; if human has the object; what action human is doing.
pr2.py	modify	add the following functionns(declared as ROS service) to simulate the actions of robot: reset; open fingers; grasp object; pointto object.
human_overlays.py	add	This file declares the functions in human.py as ROS services.(See Table 3.2 for detailed information of these ROS services.)
robot_overlays.py	add	This file declares the functions in pr2.py as ROS services.(See Table 3.2 for detailed information of these ROS services.)
main.py	modify	The changed codes are in Line 447-467. Our codes make sure that all the components(human, robot, objects) in the scenario can be overlayed as ROS services.
human.blend	modify	reconstruct the components(arm, eyes looking) of human.
pr2.blend	modify	reconstruct the components of robobt
conveyor.blend	add	a new subject for morse, which simulates the behaviours of a conveyor belt.

Table 3.1: Description of all modified files for morse

name of ROS service	name of service type	description
/human/walk_away	std_srvs/Trigger	execute the action of walking away.
/human/look_around	std_srvs/Trigger	execute the action of looking around.
/human/warn_robot	std_srvs/Trigger	execute the action of raising left arm to warn robot.
/human/attempt_grasp	std_srvs/Trigger	execute the action of attemping to grasp the object.
/human/grasp	std_srvs/Trigger	execute the action of grasping the object.
/human/is_ov	std_srvs/Trigger	return if object is visiable to human or not.
/human/is_oir	std_srvs/Trigger	return if object is in the range of human or not.
/human/is_ho	std_srvs/Trigger	return if human is having the object or not.
/human/is_a0	std_srvs/Trigger	return if human is executing the action of attemping to grasp object.
/human/is_a2	std_srvs/Trigger	return if human is staying idle and doing nothing.
/human/is_h4	std_srvs/Trigger	return if human is executing the action of warning the robot.
/human/reset	std_srvs/Trigger	put human to initial position. put the object to initial position.
/robot/point_to_obj	std_srvs/Trigger	execute the action of pointing to the object.
/robot/grasp	std_srvs/Trigger	execute the action of grasping the object.
/robot/cancel_action	std_srvs/Trigger	stop current actions and stay idle.
/robot/reset	std_srvs/Trigger	put robot to initial position.

Table 3.2: List of all the used ROS services

3.4 MDP

Finding an implementation solution for designing MDP model was a bit challenging since all implementations sources were documented for POMDP model(Partially Observable MDP) and to overcome this challenge so many search attempts have been made to find an implementation where it can support not only partially observable MDP model but also fully observable MDP model.
APPL is a C++ toolkit for approximate POMDP planning which uses XML format to model POMDPx. PomdpX is an XML file format for specifying models of Markov decision processes(MDPs), partially observable Markov decision processes(POMDPs), and mixed observability Markov decision processes(MOMDPs). As a result, the PomdpX file format can specify any of the following models: MDPs, when all state variables are fully observable and POMDPs, when all state variables are partially observable. A PomdpX document consists of a header and a pomdpx root element which in turn contains child elements, Description, Discount, Variable and thereafter: InitialStateBelief, StateTransitionFunction, ObsFunction and RewardFunction. The state, action and observation variables which factorize the state S, action A, and observation O spaces are declared within the Variable element. Reward variables, R are also declared here. By setting fullyobs tag to true, our model will be defined as a fully observable MDP model(see Fig 3.5).

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<pomdpx version='0.1' id='autogenerated'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='pomdpx.xsd'>

<Description>This is an auto-generated POMDPX file</Description>
<Discount>0.98</Discount>

<Variable>

<StateVar vnamePrev="state_0" vnameCurr="state_1" fullyObs="true">
<ValueEnum>task_assigned
           success
           failed_to_grasp
           no_attention
           evaluating
           tired
           recovery
           warningTheRobot
</ValueEnum>
</StateVar>

<ActionVar vname="action_agent">
<ValueEnum>graspAttempt
           lookAround
           idle
           walkAway
           warnRobot
</ValueEnum>
</ActionVar>

<RewardVar vname="reward_agent"/>

</Variable>
```

Figure 3.5: MDP explanation 1

We considered S0(task_assigned) as our initial beliefs state:

```
<InitialStateBelief>
<CondProb>
<Var>state_0</Var>
<Parent>null</Parent>
<Parameter type = "TBL">
<Entry>
<Instance>-</Instance>
<ProbTable>1.00 0.000 0.000 0.000 0.000 0.000 0.000 0.000</ProbTable>
</Entry>
</Parameter>
</CondProb>
</InitialStateBelief>
```

Figure 3.6: MDP explanation 2

Probabilities for each action to reach desired (specified) state:

```
<StateTransitionFunction>
<CondProb>
<Var>state_1</Var>
<Parent>action_agent state_0</Parent>
<Parameter type = "TBL">
<Entry>
<Instance>graspAttempt - - </Instance>
<ProbTable>
0.1     0.45    0.45    0       0       0       0       0
0       1       0       0       0       0       0       0
0       0.6     0.4     0       0       0       0       0
0       0.4     0.6     0       0       0       0       0
0       0.6     0.3     0       0.1     0       0       0
0       0       0       0       0       1       0       0
0       0       0       0       0       0       1       0
0       0.5     0.5     0       0       0       0       0
</ProbTable></Entry>
```

Figure 3.7: MDP explanation 3

And finally rewards for taking actions:

```
<RewardFunction>
<Func>

<Var>reward_agent</Var>
<Parent>action_agent state_1</Parent>
<Parameter type = "TBL">

<Entry>
<Instance>warnRobot warningTheRobot</Instance>
<ValueTable>4.6</ValueTable></Entry>

<Entry>
<Instance>graspAttempt success</Instance>
<ValueTable>5</ValueTable></Entry>

<Entry>
<Instance>idle evaluating</Instance>
<ValueTable>4.9</ValueTable></Entry>

<Entry>
<Instance>lookAround no_attention</Instance>
<ValueTable>2</ValueTable></Entry>

<Entry>
<Instance>walkAway recovery</Instance>
<ValueTable>4</ValueTable></Entry>
</Parameter>

</Func>
</RewardFunction></pomdpx>
```

Figure 3.8: MDP explanation 4

To check our model in APPL following commands can be used:

```
gazelle@ubuntu:~/appl-0.96/src$ ./pomdpsol ../examples/POMDP/humanModel.POMDPx

Loading the model ...
  input file   : ../examples/POMDP/humanModel.POMDPx
  loading time : 0.00s
Generate MDP Policy
gazelle@ubuntu:~/appl-0.96/src$ ./pomdpsim --simLen 100 --simNum 1000 --policy-

Loading the model ...
  input file   : ../examples/POMDP/humanModel.POMDPx

Loading the policy ...
  input file   : out.policy

Simulating ...
  action selection :  one-step look ahead

---------------------------------
 #Simulations  | Exp Total Reward
---------------------------------
 100           212.403
 200           212.375
 300           212.305
 400           212.309
 500           212.35
 600           212.371
 700           212.371
 800           212.356
 900           212.345
 1000          212.34
---------------------------------

Finishing ...

-------------------------------------------------------------
 #Simulations  | Exp Total Reward | 95% Confidence Interval
-------------------------------------------------------------
 1000          212.34           (212.202, 212.478)
-------------------------------------------------------------
```

Figure 3.9: MDP explanation 5

As you can see using ./podmpsol command APPL solves provided model and generates policy for that model and to simulate our model ./pomdpsim with number of simulations is provided. In order to run our model and insert states manually, modified version of APPL DESPOT algorithm toolkit have been used. This toolkit is modified by Orhan Can Gorur to insert states manually for our semi-autonomous model.

By default DESPOT algorithm follows its own beliefs according to probabilities and reward calculations, therefore no matter what we will input it will follow its own belief estimations. This is good for our fully autonomous model but for semi-autonomous this belief estimations will not allow us to test our few use cases manually and in MDP world removing belief means the sensors values we get(the states in our case) are 100% true. This gives no randomness, or stochasticity in action selection. Therefore, DESPOT had been adjusted in a way that the state we input comes with noise, and other beliefs are kept too. So, if we enter S4 for the next state, the belief of S4 is always 80% where the rest of the possible states are added up to 20%. That gives our human to be skeptical sometimes but still follow the exact states we are inputting in. To change this belief percentage, go to belief.cpp under src/core and edit lines 431 and 434 to adjust this belief distribution.

```
//====== This for loop is to remove all other believes but the one manually input as the state for MDP model ! ======//
int count_newState = 0;
int count_others = 0;
for (int i = 0; i < particles_.size(); i++) {
        if (particles_[i]->text() == new_state){
                count_newState += 0.8;
        }
        else if (particles_[i]->weight != 0.0){
                count_others += 0.2;
        }
}
```

Figure 3.10: MDP explanation 6

We will run Tired Human use case where after many grasp attempts when task is again assigned he will not grasp and will stay idle and after inserting Tired state he will take action walk away and will transit to Recovery state and will not do anything by staying idle.

```
###################################### Round 0 ######################################
Initial state:
[state_1:task_assigned]

- Agent Acts = 2:idle
----------------------------------Round 0 Step 0----------------------------------
Agent Took: a2, in the State: [state_1:task_assigned]

Please enter the next state [integer from 0 to 6, press any non-int key for autogen]: 5
State is manually entered: 5
curr state: 5 && prev state: 0
prev_state before cast: (state_id = -1, weight = 0, text = [state_1:task_assigned])
prev_state after cast: (state_id = -1, weight = 0, text = [state_1:tired])
=== RESULTS ===
- Resulting State:[state_1:tired]
- Reward = 0
- Accumulated rewards:
  discounted / undiscounted = 0 / 0

- Agent's belief for the obs:0:  [state_1:tired] = 0.6
 [state_1:tired] = 0.6
 [state_1:task_assigned] = 0.4
```

Figure 3.11: MDP explanation 7

```
----------------------------------Round 0 Step 1----------------------------------
Agent Took: a3, in the State: [state_1:tired]

Please enter the next state [integer from 0 to 6, press any non-int key for autogen]: 6
State is manually entered: 6
curr state: 6 && prev state: 5
prev_state before cast: (state_id = -1, weight = 0, text = [state_1:tired])
prev_state after cast: (state_id = -1, weight = 0, text = [state_1:recovery])
=== RESULTS ===
- Resulting State:[state_1:recovery]
- Reward = 17
- Accumulated rewards:
  discounted / undiscounted = 16.66 / 17

- Agent's belief for the obs:0:  [state_1:recovery] = 0.6
 [state_1:recovery] = 0.6
 [state_1:task_assigned] = 0.4
```

Figure 3.12: MDP explanation 8

```
----------------------------------Round 0 Step 2----------------------------------
Agent Took: a2, in the State: [state_1:recovery]

Please enter the next state [integer from 0 to 6, press any non-int key for autogen]: 6
State is manually entered: 6
curr state: 6 && prev state: 6
prev_state before cast: (state_id = -1, weight = 0, text = [state_1:recovery])
prev_state after cast: (state_id = -1, weight = 0, text = [state_1:recovery])
=== RESULTS ===
- Resulting State:[state_1:recovery]
- Reward = 17
- Accumulated rewards:
  discounted / undiscounted = 32.9868 / 34

- Agent's belief for the obs:0:  [state_1:recovery] = 0.6
 [state_1:recovery] = 0.6
 [state_1:tired] = 0.368254
 [state_1:recovery] = 0.6
 [state_1:tired] = 0.368254
 [state_1:task_assigned] = 0.031746
```

Figure 3.13: MDP explanation 9

Possibilities for following use case are:

```
<Entry>
<Instance>idle - - </Instance>
<ProbTable>
0.1     0       0       0       0       0.9     0       0
0       1       0       0       0       0       0       0
0       0       1       0       0       0       0       0
0       0       0       1       0       0       0       0
0       0       0       0       1       0       0       0
0       0       0       0       0       1       0       0
0       0       0       0       0       0       1       0
0       0       0       0       0       0       0       1
</ProbTable></Entry>
<Entry>
<Instance>walkAway - - </Instance>
<ProbTable>
1       0       0       0       0       0       0       0
0       1       0       0       0       0       0       0
0       0       1       0       0       0       0       0
0       0       0       1       0       0       0       0
0       0       0       0       1       0       0       0
0       0       0       0       0       0       1       0
0       0       0       0       0       0       1       0
0       0       0       0       0       0       0       1
</ProbTable></Entry>
```

Figure 3.14: MDP explanation 10

And we gave more reward to stay idle in state Recovery and a bit less reward to walk away to reach state Recovery from Tired state. To transit to state Tired from Task Assigned state, we put more probability (90%) from S0 to S5 to idle action not other actions.

For full autonomous model, since we pointed an example of despite inputting manually any state, our agent was following his own beliefs because we set belief to 100%. Now here we will show a general and bigger picture of fully autonomous model state transitions with random action selections(we considered beginner human).

```
####################################### Round 0 ########################################
Initial state:
[state_1:task_assigned]

---------------------------------Round 0 Step 0---------------------------------
- Action = 2:idle
- State:
[state_1:evaluating]
- Observation = []
- ObsProb = 1
- Reward = 9
- Current rewards:
  discounted / undiscounted = 9 / 9

---------------------------------Round 0 Step 1---------------------------------
- Action = 2:idle
- State:
[state_1:evaluating]
- Observation = []
- ObsProb = 1
- Reward = 9
- Current rewards:
  discounted / undiscounted = 17.82 / 18

---------------------------------Round 0 Step 2---------------------------------
- Action = 4:warnRobot
- State:
[state_1:evaluating]
- Observation = []
- ObsProb = 1
- Reward = 0
- Current rewards:
  discounted / undiscounted = 17.82 / 18
```

Figure 3.15: MDP explanation 11

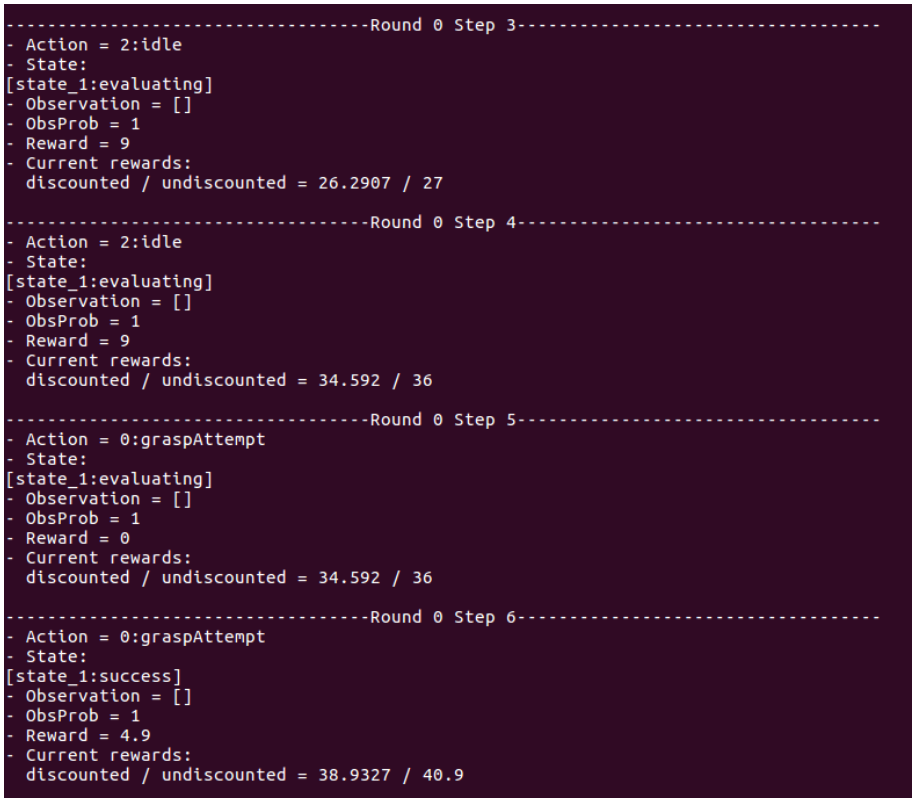

Figure 3.16: MDP explanation 12

3.5 POMDP

POMDP will help the robot to estimate the current state human might be in and based on that take an action to help reach human its desired state (SUCCESS state). The pomdp file consists of 8 parts, which include: 1.The Discount value and definition of rewarding; 2.The human state belief (estimation by robot); 3.Robot actions; 4.Observations based on the environment; 5.Initial Belief probability (starting state); 6.State Transitions for robot action; 7.Observation Probabilities for estimated state of human and 8.Rewards for taken actions.

```
# POMDP file for plan estimation, a person looks for and takes incoming object.
#

discount: 0.9
values: reward

# human states in the world, regarding the desire for taking the object
states:
        S0 # human needs no help
        S1 # human has object. SUCCESS
        S2 # human may need help in grasping
        S3 # human needs help in grasping
        S4 # human may need help in grasping but also may need help finding
        S5 # human needs help in finding
        S6 # human sent warning. cancel

########################################
# Actions:
# human and robot actions regarding the human desire of having the glasses
# Note that only one action can be executed at the same time (can be in_progress)
actions:
        rIdle         # Robot does no active task, but keeps observing
        rPoint_obj    # Robot first asks and shows (if still help needed, i.e. needs_find) where the object to human
        rGrasp_obj    # Robot first asks and fetches (if still help needed, i.e. needs_fetch) the glasses to human
        rStop_all     # Robot stops all action, if doing any and enters idle mode

########################################
# Observations:
#
# info: Robot actions are not observed. Even though the robot moves it still observes human actions
observations:
        O-0 #  object not visible, object not in range, no grasp attempt, has no object, received no warning, not idling
        O-1 #  object     visible, object not in range, no grasp attempt, has no object, received no warning, not idling
        O-2 #  object not visible, object     in range, no grasp attempt, has no object, received no warning, not idling
        O-3 #  object     visible, object     in range, no grasp attempt, has no object, received no warning, not idling
        O-7 #  object     visible, object     in range,    grasp attempt, has no object, received no warning, not idling
        O-10 # object not visible, object     in range, no grasp attempt, has    object, received no warning, not idling
        O-11 # object     visible, object     in range, no grasp attempt, has    object, received no warning, not idling
        O-16 # object not visible, object not in range, no grasp attempt, has no object, received    warning, not idling
        O-17 # object     visible, object not in range, no grasp attempt, has no object, received    warning, not idling
        O-18 # object not visible, object     in range, no grasp attempt, has no object, received    warning, not idling
        O-19 # object     visible, object     in range, no grasp attempt, has no object, received    warning, not idling
        O-32 # object not visible, object not in range, no grasp attempt, has no object, received no warning,     idling
        O-33 # object     visible, object not in range, no grasp attempt, has no object, received no warning,     idling
        O-34 # object not visible, object     in range, no grasp attempt, has no object, received no warning,     idling
        O-35 # object     visible, object     in range, no grasp attempt, has no object, received no warning,     idling
```

Figure 3.17: Discount, Human State Beliefs, Robot action and Observations

In Figure 3.17 you can see the estimated human belief states for robot. A discount value near “1” means, that every action taken and thus the given reward is accounted immediately, which means that the robot gets encouraged in a short term through positive rewarding. A low value for “discount” would make the robot think in a “long-term” process which means that robot might take negative action in an approach to achieve good results at the end.
Figure 3.18 describes with which probability robot will transition from state x to state y. For example for the first matrix (robot action: idle), first row, robot has the probability of 30% to stay in its initial state (state 0), has 40% chance to go from state 0 to state 1 and 15% each to go from state 0 to either state 2 or 4.

```
########################################
# Start probabilities - Initial Belief
start include: 0


########################################
# State Transitions
#
#

T : rIdle
0.3     0.4     0.15    0       0.15    0       0
0       1       0       0       0       0       0
0       0.3     0.3     0.4     0       0       0
0       0       0       1       0       0       0
0       0.3     0       0       0.3     0.4     0
0       0       0       0       0       1       0
0       0       0       0       0       0       1


T : rPoint_obj
1       0       0       0       0       0       0
0       1       0       0       0       0       0
0       0       1       0       0       0       0
0       0       0       1       0       0       0
0       0       0       0       1       0       0
0       0.5     0.15    0       0.025   0.3     0.025
0       0       0       0       0       0       1

T : rGrasp_obj
1       0       0       0       0       0       0
```

Figure 3.18: Start probabilities and State transitions

In Figure 3.19 you see the probabilities that can be emitted from each state, based on the state and action robot is in. The columns are different observations (see figure 1 for further explanation) and the states are rows.

```
#####################################
# Observation Probabilities
#
#            Estimated state of human
# Observation

O : rIdle
0       0       0       0.1     0       0       0       0       0       0       0       0       0       0       0.9
0       0       0       0       0       0.2     0.8     0       0       0       0       0       0       0       0
0       0.325   0       0.1     0.1     0       0       0       0       0       0       0       0.325   0       0.15
0.1     0.15    0.05    0.05    0.1     0       0       0       0       0       0       0.1     0.25    0.05    0.15
0.2     0       0.5     0       0       0       0       0       0       0       0       0.1     0       0.2     0
0.3     0       0.7     0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0.05    0.15    0.1     0.7     0       0       0       0
```

Figure 3.19: Observations

Figure 3.20 shows the reward table for certain actions. In this Scenario we want the robot to be heavily encouraged of taking the object and successfully enter state 1 after that (SUCCESS state), while punishing actions we do not desire in certain states. This might be idling in unsuitable situations (here: states), for instance when the human needs help in grasping or finding the object (= idling in state s3, s5, s6 will lead into punish). Also slightly punish the robot for doing wrong actions like pointing when the robot should be grasping and vice versa.

```
#######################################
# Rewards
#
#   ACTION                  START-STATE                 END-STATE            OBSERVATION       REWARD

R:      rIdle           :       *                   :       S6                  :       *       -5
R:      rIdle           :       *                   :       S5                  :       *       -5
R:      rIdle           :       *                   :       S3                  :       *       -5

R:      rPoint_obj      :       S5                  :       S1                  :       *       5
R:      rPoint_obj      :       S5                  :       S2                  :       *       1
R:      rPoint_obj      :       *                   :       S3                  :       *       -1
R:      rPoint_obj      :       *                   :       S4                  :       *       -1

R:      rGrasp_obj      :       S3                  :       S1                  :       *       100
R:      rGrasp_obj      :       *                   :       S5                  :       *       -1
R:      rGrasp_obj      :       *                   :       S4                  :       *       -1
R:      rGrasp_obj      :       *                   :       S2                  :       *       -1

R:      rStop_all       :       S6                  :       S1                  :       *       5
R:      rStop_all       :       *                   :       S5                  :       *       -1
R:      rStop_all       :       *                   :       S6                  :       *       -1
R:      rStop_all       :       S6                  :       S4                  :       *       1
R:      rStop_all       :       *                   :       S3                  :       *       -1
R:      rStop_all       :       S6                  :       S2                  :       *       1
```

Figure 3.20: Rewards for actions

4 Most Frequently Asked Questions

4.1 Are different computer systems(Window, MacOS), different Morse version or different ROS version also supported?

For now we haven't tested our code in any other system environments(system, morse, ROS), so there maybe some unexpected errors or bugs when running our codes in a different environment.

4.2 Replacing the morse source codes with the provided codes seems a bit unconvenient, are there any different ways?

Yes, it's not so convenient to replace the morse sourse codes with our codes. Howerer, our codes only extend the functionalities of morse and dont' conflict with original morsesource codes. This makes sure that you can our extended codes in your other project. If you are still not sure, we would recommend you to make a back-up for all the modified codes first.

4.3 Where are the MDP and POMDP model files? How can I change them?

The model files are under path: /code/despot_MDP/examples/pomdpx_models/data/ and /code/despot_POMDP/examples/pomdpx_models/data/. For MDP, we provide five different scenario files; for POMDP we provide one model file. You can change the model files easily by changing the path in the two files despot_MDP_shell.sh and despot_shell.sh, which are located under path: /code/ros-ws/src/ros_hrc/.