PCMRPP: A POMDP file Creator package for Mobile Robot Path Planning for a 2-robot System

1. Introduction

"PCMRPP" software package implements a novel algorithm to programmatically generate the POMDP model in which the sparseness of the T and O matrices can be decided such that a trade-off between the time complexity of resulting POMDP and the level of robustness of the solution can be achieved. This is done by controlling two aspects of the model-one, the probability spread of the belief state, and two, the granularity of discretization of the components of the state. The software package is implemented in C on Linux. In the first version, the scope is limited to only two mobile robots moving in an open 2D space with the possibility of collision. The code is tested with examples of the POMDP model with 1152 states with non-zero Probability spread over ten observations, and the results are presented.

POMDP has been a well-proven standard framework for handling uncertainty using probabilistic techniques. It allows the robot to perceive the state of the environment as a probability distribution over multiple states called belief states. The starting point of POMDP based solution is a model file listing all the possible states and various probability matrices. The size of these matrices depends on the granularity of discretization of multiple parameters affecting the state. The sparseness of the matrices depends on the probability spread of the belief state. Even for a moderate-sized system of about 1000 states, the matrix can quickly become of size 10^9 entries. Creating such a huge-sized file manually by meticulously deciding the probability value for each state after every action is too difficult or impossible. Hence, a software package named PCMRPP is proposed for programmatically generating the POMDP model file based on two parameters configured by the user - the granularity of discretization and the span of the probability distribution of the belief state.

2. Brief basics about POMDP

The POMDP model is constructed using a 7-component tuple $\{S, A, Z, T, O, R, \gamma\}$, where S is the set of states in which the robot can be at any time, A is the set of possible actions that the robot may take, and Z is the set of observations that the robot will receive in response to an action. Refer Fig. 1.

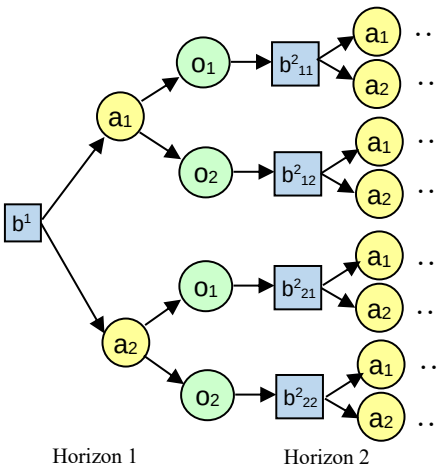


Fig. 1 : The chain of POMDP – current Belief State >> Action >> Observation >> next belief State

The probability of the robot being in state s_{t+1} after taking action a_t at time t in state s_t is modeled as Transformation matrix T as given in Eq. 1.

$$T\ (s_t, a_t, s_{t+1}) = P\ (s_{t+1}|s_t, a_t) \qquad \ldots 1$$

O is the observation matrix indicating the probability of the robot receiving observation o_t after it takes action a_t in state s_t. Refer Eq. 2.

$$O\ (s_t, a_t, o_t) = P(o_t|s_t, a_t) \qquad \ldots 2$$

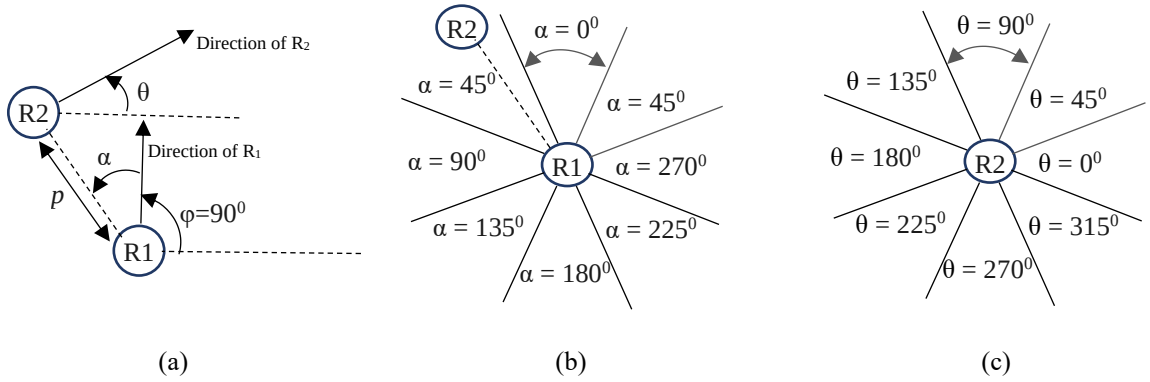
R is a reward function indicating the positivity or negativity of the robot's action in a specific state. A positive value of R indicates that the action was appropriate for the robot in that state, and a negative value indicates a penalty for taking an incorrect or inappropriate action in that state. The action taken by the robot is not only for immediate gain, but it also considers future gains.

The states and observations form the core part of the POMDP framework. For a reasonably accurate solution of POMDP for Mobile Robot Navigation, S can easily cross a few thousand states and almost the same number of Observations (Z). This will create substantial-sized T and O matrices. The time complexity of solving the POMDP depends on the dimensions of these matrices and how sparse these matrices are. PCMRPP allows the user to control the size of these matrices by controlling the granularity of discretization of the robot parameters and the O matrix's sparseness by controlling the belief state's span. The POMDP solver takes in the POMDP model, generating either a set of *alpha vectors, a policy graph,* or both. These *alpha vectors* are used in a dot-product with the belief state to decide the most optimal action the robot must take. Thus, controlling the probability spread of belief state also results in faster arriving at a decision during run-time.

3. Software description

3.1 States and actions formulation

Two mobile robots are assumed to move in a 2-D bounded field without static obstacles. Each robot calls itself R_1, and the other robot is R_2. Refer to Fig. 2. Each robot fixes the frame of reference by always assuming its direction straight northward, indicated by $\varphi=90^0$. The robot does all computations with respect to this frame of reference. The state of the robot can be formulated using a combination of parameters. proposes that some parameters are fully deterministic, whereas some are uncertain, leading to a state with mixed observability. In this research work, the state of the robot is modelled using five parameters (Fig 1(a)) – its speed (*r1s*), the angular direction of the other robot (α), the direction in which the other robot is moving (θ), the speed of the other robot (*r2s*) and the distance between the two robots (p).



(a) (b) (c)

Fig. 2: (a) Parameters of the state definition (b) Discretisation of α (c) Discretisation of θ

The parameters α and θ are discretized to 8 sectors for a typical configuration, as shown in Fig 2 (b) and Fig 2(c). The proximity parameter *p* is discretized to No-Proximity, Close-proximity, and Collision. The speeds of the two robots are discretized into low, medium, and high. These discretization levels are configurable in the program. The robot's state is named as a text string concatenating the discretized values of these parameters in the order – α (in degrees), θ (in degrees), r1s, r2s, and p. Each state also has a corresponding entry in the Observation space. Hence a typical state and observation entry will look like – "st0123_30_60_Low_Medium_No" and "obs0123_30_60_Low_Medium_No". If a non-verbose state name is selected, these entries would be restricted to "st0123" and "obs0123," respectively. The action by the robot is also discretized to a level that can be configured. For example, a typical configuration would be a turning angle in multiple of 45^0. So, the robot can turn right 90° (ac_TR90), turn right 45° (ac_TR45), go straight ahead without turning (ac_MF), turn left 45° (ac_TL45) or turn left 90° (ac_TL90). Thus no. of actions is 6 (including the "Stop" action).

3.2 Software architecture

PCMRPP is written in C and implemented for a Linux environment. It is constructed as a combination of 5 software modules.

Init module initializes the various data structures, like Transition Matrix, Observation matrix, etc., with basic parameters. It also writes the pre-amble of the POMDP file by iterating through all discrete values of all the parameters.

Iteration module iterates through all the values of the five parameters - α, θ, $r1s$, $r2s$, and p in discrete steps. These steps must be of the same or lower level of granularity than the discretization level configured for that parameter. In each iteration, the system is moved in incrementally from *t* to *t+1,* and the robots' positions are updated as per their present pose, action, and speed. The new values of all the parameters are discretized the new state is formulated. For each start state s_x, an array records which are the end states e_y that are reached and how many times each of them is reached (N_x^y).

State Transition Probability module takes in the record of start and end states created by the iteration block. It calculates the State Transition Probability *T* by Eq. 3. Here, the start state s_x transitions into end state e_y, and the number of occurrences of the transition $T(s_x, e_y)$ is N_x^y where *y* is the total number of end states reached from the start state s_x.

$$Transition\ Probability\ T(s_x, e_y) = \frac{N_x^y}{\sum_{a=0}^{y} N_x^a} \quad \ldots 3$$

Algorithm 1: Generating State Transition Matrix

Input: ***α, θ, p, R1Speed, R2Speed***
Output*: T(s,a,s') - Transition probability Matrix*
begin
fix initial position of R1 (Typically at the center of the field)
iterate through all values of a, p and α (discretized)
find the position of R2
iterate through all values of θ, R1Speed and R2Speed (discretized)
code the current state from all parameters (ss).
calculate position of R1 and R2 for t=t+1
calculate new values of α, θ and p
based on new values of α, θ and p, code the next state (es)
record the ss→es transition
count no. of occurrences of each es for each ss
*calculate Transition probability T(ss,es) = No. of occurrences of each es * 100/Total occurrences of es*
create a T Matrix table entry with a, ss, es and probability.
end

Observation Probability module is responsible for creating the O matrix. Each of the end states is decoded into individual five parameters. The parameters are given weightages such that current value of the parameter has 80% weightage and the adjacent values on both sides have 10% weightage. So, for example, if α is in sector 3 (between 90^0 and 135^0) then α being in sector 3 is assigned 80% weightage, α being in sector 1 (between 45^0 to 90^0) is assigned 10% weightage and α being in sector 4 (between 135^0 to 180^0) is given 10% weightage. The same process is repeated for other parameters (θ, *r1s*, *r2s,* and *p*). One combination of all parameters having non-zero weightage values gives one observation o_y and the addition of the corresponding weightages is denoted by w_y. The observation probability is then computed as in Eq. 4.

$$Observation\ Probability\ (s_x, e_y) = o_y = \frac{w_y}{\sum_{i,j,k,l,m=1}^{3}(w_i^{\alpha}+w_j^{\theta}+w_k^{r1s}+w_l^{r2s}+w_m^{p})} \quad \ldots\ldots 4$$

Algorithm 2: Create Observation Probability Matrix

Input: es, act, ss – End state, Action, Start state
Output : array obs[1...N] – Observation Probability O(s,a,s')
begin
extract individual components from es
//allocate Observation probability weightages to angles alpha and theta
allocate 80% weightage to the current value
allocate 10% weightage to the discrete values of alpha and theta on both sides of the current value
// allocate Observation probability weightages to speed values
if speed is normal, allocate 80% to normal and 10% each to slow and fast.
if speed is slow, allocate 90% to slow and 10% to normal.
if speed is fast, allocate 90% to fast and 10% to normal.
// allocate Observation probability weightages to proximity values
if proximity is "no-proximity" then allocate 90% to "no-proximity" and 10% to "close-proximity."
if proximity is "close-proximity," then allocate 80% to "close-proximity" and 10% each to" no-proximity" and "collision"
if proximity is "collision," then allocate 90% to "collision" and 10% to"close-proximity"
repeat through all discrete values of angles, speeds, and proximity
compose state from all components having non-zero weightage value
add up all the weightages for this state. Treat this as observation.
the probability of this observation = addition of weightage for its corresponding composed state / Σ all weights computed for this end state es.
next
Sort the probability array created above.
Pick up only the first N entries where N is the required spread for the Observation probability distribution.
Normalize the array for the correct sum of probability to 1 and return the array.
end

With this, a list of observation probabilities is obtained for an end state e_y. The probabilities in this array are then sorted in ascending order. If the belief state spread is configured as *M,* then only first *M* values in this array are picked up for further processing, discarding the rest. The discarded probability values are adjusted with other elements in the array so that the final probability addition is 1.

Reward module is responsible for creating the reward matrix. The reward for a robot is decided on various criteria. The robot gets a positive reward on - reduction of distance to the goal position, going away from a potential collision, updating of pose such that it turns away from another robot on a collision course, updating of pose such that it turns towards the goal position, etc. The robot gets a negative reward for – collision, moving towards a potential collision, going away from the target position, etc.

Utility functions module is a collection of functions like discretization functions, distance finding, etc. Other modules use these functions.

3.3 Key Data Structures

T Array: *T*[*noStates*][*noOccurances*] – This array stores the number of times an end state is reached from a start state because of an action. This is used for Transition probability computation.
T Matrix Count: T_mat_cnt[[noActions][noStates][max_es_cnt] – This array is used for recording the transition from each start state to multiple end states. Eq. 3 works on this array to find the transition probability matrix.
O Array: O[noActions][noStates][noStates] – This array is used for Observation probability calculation by accumulating weughtages of all adjoining possible values for all 5 parameters which constitute the state. The Eq. 4 works on this array. This array is re-initialised in every iteration for new start state.
R Array: R[noStates][noActions] – This is rewards array for storing the rewards pertaining to each action and state.

3.4 Software configurability

The different modules of the software are implemented as individual C functions to ease modularity since complete flexibility of configuring the PCMRPP is possible. Following key parameters of the software package can be configured –

- discretization levels of α, *θ, r1s, r2s* and *p*
- Observation probability spread.
- No. of actions the robot can take.

4. Illustrative examples

We now provide a few illustrative examples with a typical configuration of the parameters. We create the POMDP file in the format given in. It is a simple text file with 6 data blocks – actions list, states list, observations list, all non-zero Transition probabilities list, all non-zero observation probabilities list, and all non-zero rewards list. Following examples illustrate the working of the code. For these examples, discretization levels for all 5 state parameters and actions are decided and the output of the code which is a pomdp file is analysed. The no of non-zero entries of T, O and R matrices is recorded. The sparseness of the matrix is calculated by Eq. 5 as follows –

$$Matrix\ sparseness = \frac{No.of\ non-zero\ entries\ in\ the\ matrix}{Total\ size\ of\ Matrix} \quad \ldots 5$$

Example 1: The following configuration is used –
α and *θ*: both discretized into eight levels – 0-45°, 45°-90°, 90°-135°, 135°-180°, 180°-225°, 225°-270°, 270°-315°, and 315°-360°
r1s and r2s: both discretized into three levels – low (L), medium (M), high (H)
p: discretized into three levels – Close proximity (P), No proximity (N), and Collision (C)
a: actions in terms of turning angle are discretized into five levels – turn left 45° (ac_TL45), turn left 90° (ac_TL90), turn right 45° (ac_TR45), turn right 90° (ac_TR90) and move straight forward (ac_MF).
The number of states and observations: 1152 each
Belief state spread: 2
The pomdp file generated: mrsOBS02.pomdp.

Example 2: The following configuration is used –
α and *θ*: both discretized into eight levels – 0-45°, 45°-90°, 90°-135°, 135°-180°, 180°-225°, 225°-270°, 270°-315°, and 315°-360°

r1s and r2s: both discretized into three levels – low (L), medium (M), high (H)
p: discretized into three levels – Proximity (P), No proximity (N), and Collision (C)
a: actions in terms of turning angle are discretized into five levels – turn left 45° (ac_TL45) and 90° (ac_TL90), turn right 45° (ac_TR45) and 90° (ac_TR90), and move straight forward (ac_MF).
The number of states and observations: 1152 each
Belief state probability spread: 10 observations.
The pomdp file generated: mrsOBS10.pomdp.

The POMDP models generated for the above examples are passed through the POMDP solver (APPL Toolkit, https://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/), and a POMDP executioner program is written to test the POMDP policy generated by the solver.