Probabilistic Robotics Course

Discrete Filtering: Localization

Omar Salem
salem@diag.uniroma1.it

Lorenzo De Rebotti
derebotti@diag.uniroma1.it

Department of Computer Control and Management Engineering
Sapienza University of Rome

Exercise

What if grid-orazio has also an *orientation* and its available controls change to:

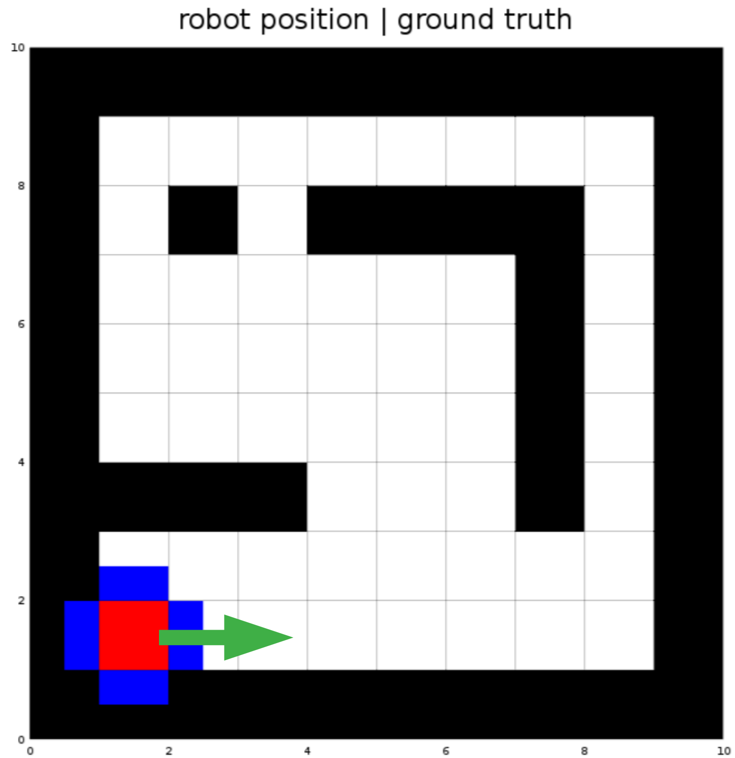
- MOVE_FORWARD
- MOVE_BACKWARD
- ROTATE_LEFT
- ROTATE_RIGHT

How does the state change?

What about the observation and transition model?

State representation

- The state of the robot is 3-dimensional
  - A) Position in the 2D grid (row, col)
  - B) Orientation which can be 0, π, $\pi/2$, $-\pi$



State representation

- The state is 3-dimensional
- The belief state is a cube

```
#initialize robot states: position belief values over the complete grid
number_of_free_cells =  sum(sum(map < 0.01));
belief_initial_value = 1/(number_of_free_cells*THETA_VALUES);
global state_belief = zeros(rows(map), columns(map), THETA_VALUES);
for (theta = 1:THETA_VALUES)
  state_belief(:,:,theta) = (~map)*belief_initial_value;
endfor
global observations  = [0 0 0 0];
```

- Map orientation to indices of the 3rd dimension, CCW and CW rotations become

```
theta_to = mod(theta_from_, THETA_VALUES) + 1;
```

```
theta_to = mod(theta_from_ - 2, THETA_VALUES) + 1;
```

State representation (drawBelief.m)

- Marginalize out the orientation to obtain a 2D matrix
- In this way you can visualize the probability of being in a cell independently of the orientation

```
global THETA_VALUES
num_rows = rows(map_);
num_cols = columns(map_);
state_belief_2D = zeros(num_rows, num_cols);
#state_belief_2D
for (row = 1:num_rows)
    for (col = 1:num_cols)
        for (theta = 1:THETA_VALUES)
            state_belief_2D(row, col) += state_belief_(row, col, theta);
        endfor
    endfor
endfor
#invert belief value for plotting (0:black: 100% confidence, 1:white: 0% confidence)
plotted_state_belief_ = flipud(ones(size(state_belief_2D)) - state_belief_2D);
```

State representation (getNextState.m)

- In all possible values of the next state consider also the next possible orientation

```
#available motion range
min_row = state_ground_truth_(1)-1; #MOVE_UP
max_row = state_ground_truth_(1)+1; #MOVE_DOWN
min_col = state_ground_truth_(2)-1; #MOVE_LEFT
max_col = state_ground_truth_(2)+1; #MOVE_RIGHT
#over for the available motion range check if probability is higher than the extracted sample
for (i=1:THETA_VALUES)
  for (row = min_row:max_row)
    for (col = min_col:max_col)
      cumulative_probability += transition_probability(row, col,i);
      if(cumulative_probability > minimum_probability)
        #return with new position
        state_ground_truth = [row, col, i];
        return;
      endif
    endfor
  endfor
endfor
```

- Use of cumulative distribution to get random sample from prob. distribution (see PF slides)

A) Transition Model (transitionModel.m)

How to move grid-orazio? We need to implement a function in the form:

```
function transition_probability_matrix = transitionModel(map_, row_from_, col_from_, theta_from_, control_input_)
```

that given:

- a start state [row_from_, col_from_, theta_from_]
- a control input

```
#available robot controls (corresponding to keyboard key values)
global MOVE_FORWARD;
global MOVE_BACKWARD;
global ROTATE_LEFT;
global ROTATE_RIGHT;
```

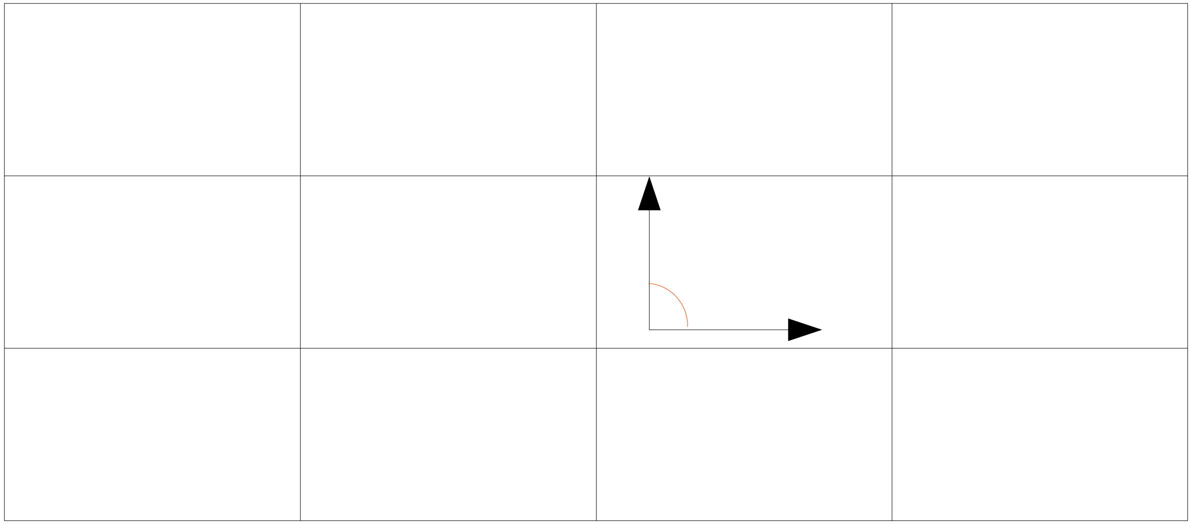
returns the probability of moving to any cell with any orientation in the map from the start state

A1) Transition Model: Without noise

We assume that the controls we issue to grid-orazio, have a deterministic effect (**no noise**).

- To a control ROTATE_LEFT, the robot will respond by rotating clockwise by $\pi/2$ with probability 1.0.

ROTATE_LEFT:

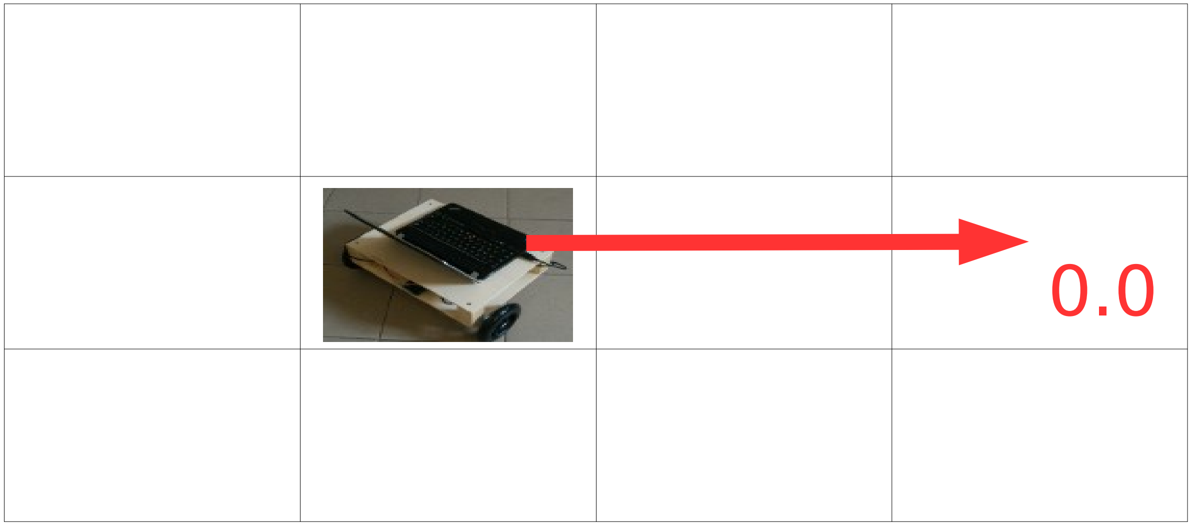


The behavior is symmetric for ROTATE_RIGHT

A1) Transition Model: Motion constraint

The robot can move only to adjacent cells, for farther cells the transition probability becomes 0.0.

MOVE_RIGHT:

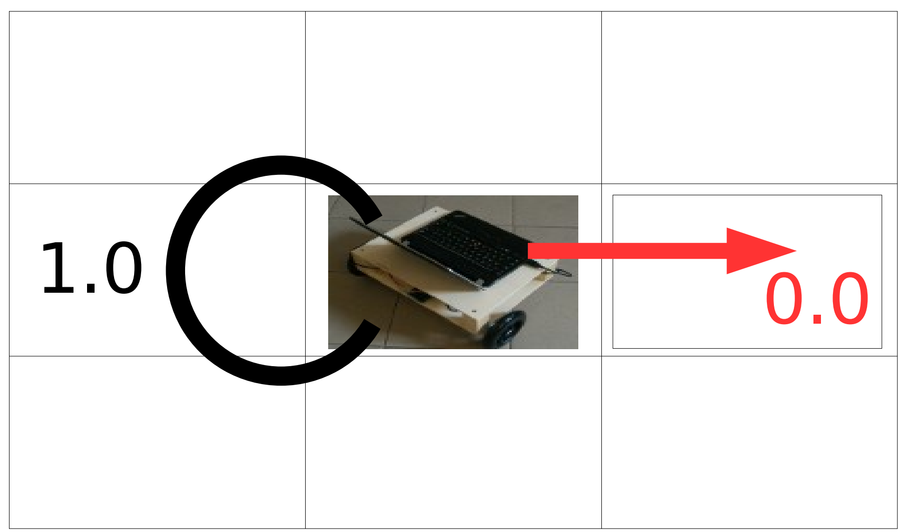


The behavior is symmetric for MOVE_LEFT

A1) Transition Model: Motion feasibility

If the target cell (**noise free** transition) is occupied, the robot will stay where it is with probability 1.0.

MOVE_RIGHT:



The behavior is symmetric for all 4 controls

A1) Transition Model: Motion constraint

Loop over the rows and columns of the map

The robot can move only to adjacent cells:

```
#compute resulting position difference
translation_rows = row_to - row_from_;
translation_cols = col_to - col_from_;

#allow only unit motions (1 cell): check if we have a bigger motion
if(abs(translation_rows) > 1 || abs(translation_cols) > 1)
  continue;
endif
```

If the two cells are farther away than 1, the transition probability remains 0.

A1) Transition Model: Next state

Retrieve the *noise free* next state based on the control input:

```
case MOVE_FORWARD
  switch theta_from_
    case 1
      # move right
      target_col++;
    case 2
      # move up
      target_row--;
    case 3
      # move left
      target_col--;
    case 4
      # move down
      target_row++;
    otherwise
      disp("Not a known angle")
  endswitch
```

```
case MOVE_BACKWARD
  switch theta_from_
    case 1
      # move left
      target_col--;
    case 2
      # move down
      target_row++;
    case 3
      # move right
      target_col++;
    case 4
      # move up
      target_row--;
    otherwise
      disp("Not a known angle")
  endswitch
```

A1) Transition Model: Motion feasibility

We have to check if the next state is feasible on our map (i.e. the cell is not occupied and we’re not going over the border):

```
#check if the desired motion is infeasible
invalid_motion = false;
if (target_row < 1 || target_row > map_rows || target_col < 1 || target_col > map_cols) #if we're going over the border
  invalid_motion = true;
elseif (map_(target_row, target_col) == 1 || map_(row_to, col_to) == 1) #obstacle in the goal cell
  invalid_motion = true;
endif
if (invalid_motion)

  #if the desired translation is zero
  if (translation_rows == 0 && translation_cols == 0)
    transition_probability_matrix(row_to, col_to) = 1; #we stay with 100% probability (no motion has full confidence)
    continue;
  else
    continue; #we cannot move
  endif
endif
```

A1) Transition Model:

MOVE_FORWARD/BACKWARD

Set the probability of moving to a cell depending on the control input:

```
switch theta_from_
    case 1
      # move right
      if (translation_rows     ==  0 && translation_cols ==  1)
        transition_probability_matrix(row_to, col_to, theta_from_) = 1;
      endif
    case 2
      # move up
      if (translation_rows     ==  -1 && translation_cols ==  0)
        transition_probability_matrix(row_to, col_to, theta_from_) = 1;
      endif
    case 3
      # move left
      if (translation_rows     ==  0 && translation_cols ==  -1)
        transition_probability_matrix(row_to, col_to, theta_from_) = 1;
      endif
    case 4
      # move down
      if (translation_rows     ==  1 && translation_cols ==  0)
        transition_probability_matrix(row_to, col_to, theta_from_) = 1;
      endif
    otherwise
      disp("Not a known angle")
endswitch
```

Analogous for MOVE_BACKWARD

A1) Transition Model:

ROTATE_LEFT/RIGHT

- The robot cannot translate and rotate at the same time
- When it rotates, it can only remain in the same state with different orientation
- Modulo to always get admissible angle

```
case ROTATE_LEFT
  # theta_to = theta_from_ + pi/2;
  theta_to = mod(theta_from_, THETA_VALUES) + 1;
  if (translation_rows     ==  0 && translation_cols == 0)
    transition_probability_matrix(row_to, col_to, theta_to) = 1.0;
  endif
case ROTATE_RIGHT
  theta_to = mod(theta_from_ - 2, THETA_VALUES) + 1;
  if (translation_rows     ==  0 && translation_cols == 0)
    transition_probability_matrix(row_to, col_to, theta_to) = 1.0;
  endif
```

B) Observation Model (observationModel.m)

To retrieve the 4 observations around grid-orazio we use our observation model

```
current_probability = observationModel(map_,
                                       state_ground_truth_(1),
                                       state_ground_truth_(2),
                                       current_observations);
```

that given:

- a start state
- a observation sample (4 values)

returns the probability of observing the current observation sample

B) Observation Model: Modeling the Bumper

Given the location, each of the 4 bumpers is independent

A bumper gives a wrong measurement with probability 0.2

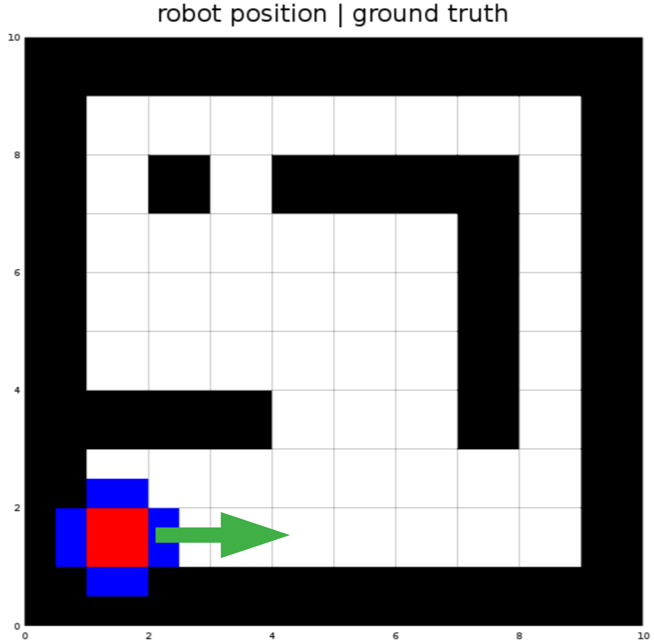
In this situation

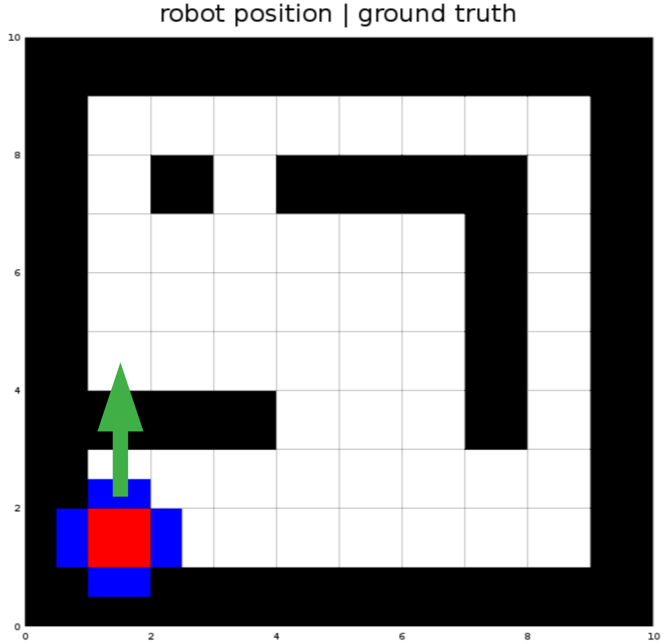
p($\mathbf{z}_{RIGHT}$=toggled)=0.8

During the synthesis of an observation model you **assume** you know **both** state and the measurement. The observation model tells you how likely the measurement is in the state

B) Observation Model: Modeling the Bumper

- Depending on the orientation of the robot, the bumpers capture different cells on the map
- Bumper 1 is in the cell up with orientation 0 *but* in cell left with orientation π





B) Observation Model: Modeling the Bumper

One possible strategy is to write a matrix encoding where the bumpers should be found depending on the orientation

```
orientation_to_cells_mapping(:,:,1) = [ -1 1 0 0;
                                         0 0 -1 1];
orientation_to_cells_mapping(:,:,2) = [ 0 0 1 -1;
                                        -1 1 0 0];
orientation_to_cells_mapping(:,:,3) = [ 1 -1 0 0;
                                         0 0 1 -1];
orientation_to_cells_mapping(:,:,4) = [ 0 0 -1 1;
                                         1 -1 0 0];
```

```
cell_bumper_west = cell_ + orientation_to_cells_mapping(:,1,theta_);
```

cell_up→cell_west, if the robot always points the north)

Observation Model

- Once you retrieved the cells corresponding to the bumpers compute the probability of the observation
- Using the convention robot always pointing north we have

```
#update probability depending on observations
if (cell_bumper_west_occupied == observations_(1))
  observation_probability *= .8;
else
  observation_probability *= .2;
endif
if (cell_bumper_est_occupied == observations_(2))
  observation_probability *= .8;
else
  observation_probability *= .2;
endif
if (cell_bumper_south_occupied == observations_(3))
  observation_probability *= .8;
else
  observation_probability *= .2;
endif
if (cell_bumper_north_occupied == observations_(4))
  observation_probability *= .8;
else
  observation_probability *= .2;
endif
```

Localizing Orazio

- Predict belief

```
#PREDICT robot position belief
state_belief_previous = state_belief;
state_belief = zeros(map_rows, map_cols, THETA_VALUES);
for (theta=1:THETA_VALUES)
  for row = 1:map_rows
    for col = 1:map_cols
      state_belief += transitionModel(map, row, col, theta, control_input)*state_belief_previous(row, col, theta);
    endfor
  endfor
endfor
```

- Update belief

```
#UPDATE robot position belief and COMPUTE the normalizer
disp("observations: ")
disp(observations)
inverse_normalizer = 0;
for (theta=1:THETA_VALUES)
  for row = 1:map_rows
    for col = 1:map_cols
      state_belief(row, col, theta) *= observationModel(map, row, col, observations);
      inverse_normalizer            += state_belief(row, col, theta);
    endfor
  endfor
endfor
```

Useful hints for debugging

- Create certainty by divide-and-conquer

  A) Ensure that the noise-free model is correct

  B) Ensure that predict step is working

  C) Then add update step

  D) Check that the inputs to all the main passages are correct

  E) Try to visualize as much as possible