Probabilistic Robotics Course

Discrete Filtering: Localization

Bartolomeo Della Corte
dellacorte@diag.uniroma1.it

Dominik Schlegel
schlegel@diag.uniroma1.it

Dept of Computer Control and Management Engineering
Sapienza University of Rome

Implementing a Bayes Filter

Choose how to represent the state

Choose how to represent the controls

Choose how to represent the observations

Implement a transition model

Implement an observation model

Outline

- Scenario: grid-orazio

- Modeling the problem
  - A) Transition model $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_{t-1})$
    - 1) without noise
    - 2) with noise
  - B) Observation model $p(\mathbf{z}_t \mid \mathbf{x}_t)$

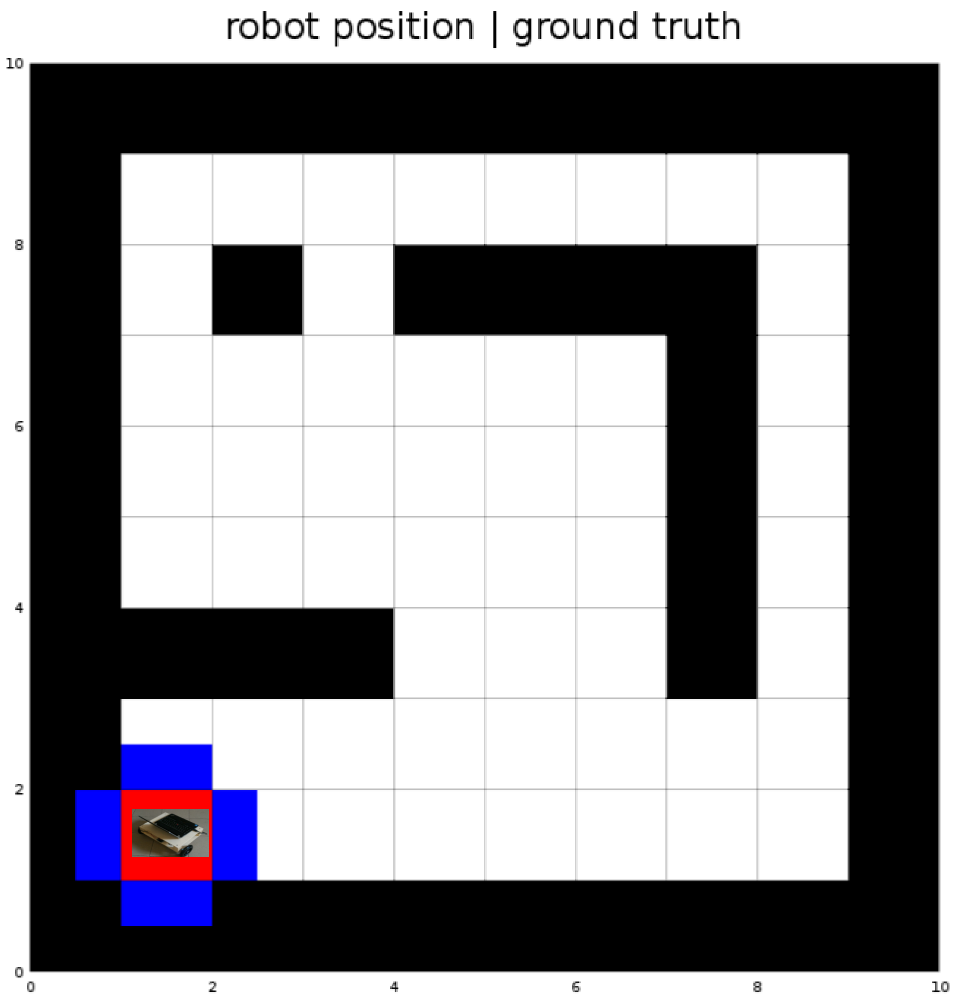
- Building the filter
  - C) **Predict** belief $p(\mathbf{x}_t | \mathbf{u}_{1:t-1}, \mathbf{z}_{1:t-1})$
  - D) **Update** belief $p(\mathbf{x}_t | u_{1:t-1}, \mathbf{z}_{1:t})$

Scenario: grid-orazio

grid-orazio (red) lives in a grid world. The cells of this world are either free (white) or occupied (black)

At each point in time, grid-orazio can receive one of the 4 commands to move: UP/DOWN/LEFT/RIGHT

grid-orazio senses the state around it with 4 bumpers (blue) mounted at its 4 sides



Scenario: Map

In this problem we will use some prior knowledge: a 2D map.

A map is a grid, conveniently represented by a matrix, with the convention:

- a cell having value 0 is free
- a cell having value 1 is occupied

```
#generate/load our map
global map = getMap('maps/map.txt');
```

Scenario: Our program

Let’s fill the holes

```
# A) retrieve new state ground truth according to our transition model
state_ground_truth =

# B) obtain current observations according to our observation model
observations =

# C) PREDICT state belief
for row = 1:map_rows
        for col = 1:map_cols
            state_belief +=
        endfor
endfor

# D) UPDATE state belief and COMPUTE the normalizer
inverse_normalizer = 0;
for row = 1:map_rows
        for col = 1:map_cols
                state_belief(row, col) *=
                inverse_normalizer     +=
        endfor
endfor
normalizer =
state_belief *=
```

A) Transition Model

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

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

that given:

- a start state
- a control input

```
#available robot controls
global MOVE_UP    = 105; # i
global MOVE_DOWN  = 107; # k
global MOVE_LEFT  = 106; # j
global MOVE_RIGHT = 108; # l
```

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

A1) Transition Model: Motion constraint

The robot can move only to adjacent cells:

```
#compute resulting position difference
translation_rows = row - row_from_;
translation_cols = col - 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:

```
#compute target robot position according to input
target_row = row_from_;
target_col = col_from_;

switch (control_input_)
        case MOVE_UP
                target_row++;
        case MOVE_DOWN
                target_row--;
        case MOVE_LEFT
                target_col--;
        case MOVE_RIGHT
                target_col++;
        otherwise
                return;
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 not feasible
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, col) == 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, col) = 1; #we stay with 100% probability (no motion has full confidence)
    continue;
  else
    continue; #we cannot move
  endif
endif
```

A1) Transition Model: Without noise

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

```
      #our motion is feasible - compute resulting transition
      switch (control_input_)
        case MOVE_UP
          if (translation_rows ==  1 && translation_cols ==  0) transition_probability_matrix(row, col) = 1.0; #desired motion
          endif;
        case MOVE_DOWN
          if (translation_rows == -1 && translation_cols ==  0) transition_probability_matrix(row, col) = 1.0; #desired motion
          endif;
        case MOVE_LEFT
          if (translation_rows ==  0 && translation_cols == -1) transition_probability_matrix(row, col) = 1.0; #desired motion
          endif;
        case MOVE_RIGHT
          if (translation_rows ==  0 && translation_cols ==  1) transition_probability_matrix(row, col) = 1.0; #desired motion
          endif;
      endswitch
```

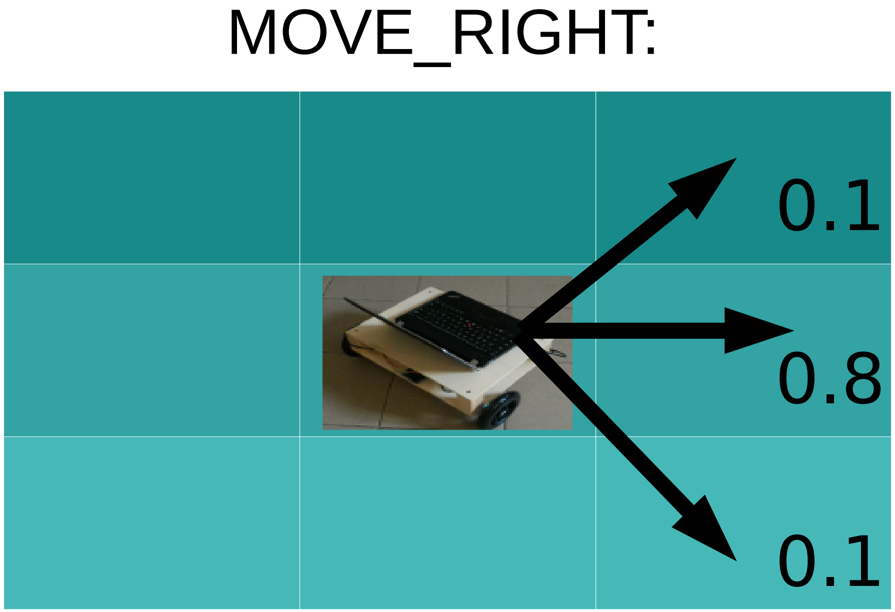
Since we’re assuming no uncertainty, the probability for moving to the next state is maximal. And 0 for all other states.

A2) Transition Model: Modeling the Controls

The controls we issue to grid-orazio, do not have a deterministic effect anymore (because we have *noise*)

To a control MOVE_RIGHT, the robot will respond by moving

- right with prob. 0.8
- top-right with prob. 0.1
- bottom-right with prob. 0.1



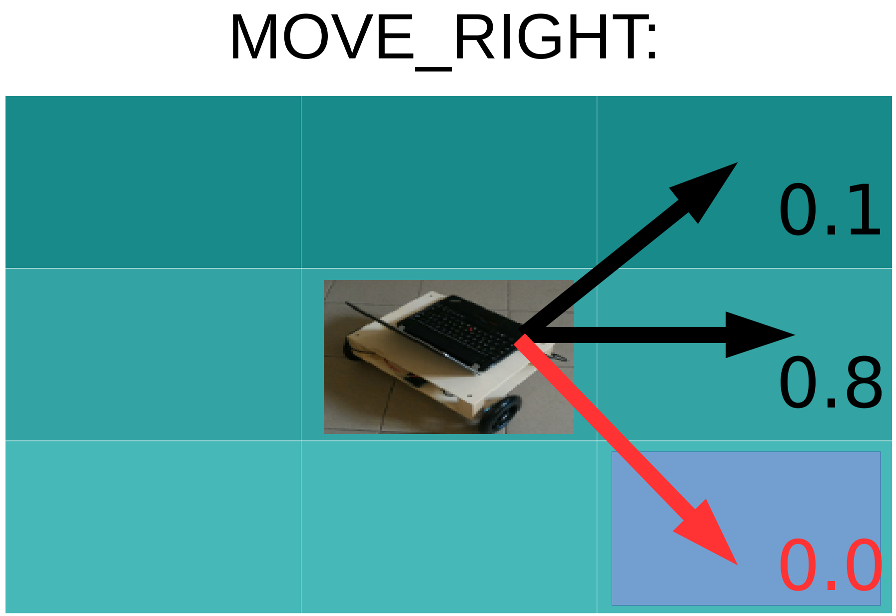
The behavior is symmetric for all 4 controls

A2) Transition Model: Modeling the Controls

The controls we issue to grid-orazio, do not have a deterministic effect anymore (because we have *noise*)

To a control MOVE_RIGHT, the robot will respond by moving

- right with prob. 0.8
- top-right with prob. 0.1
- bottom-right with prob. 0.1



This holds if the destination is not occupied,
Otherwise the probability becomes 0

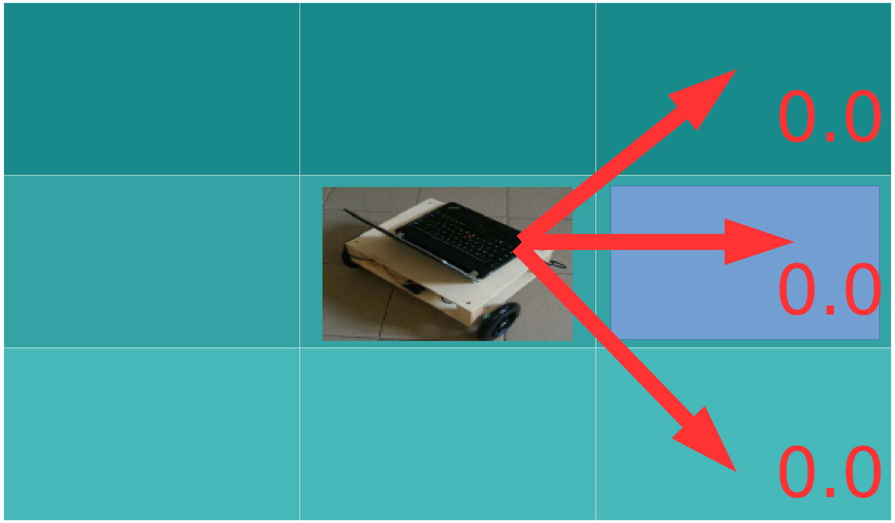
A2) Transition Model: Modeling the Controls

The controls we issue to grid-orazio, do not have a deterministic effect anymore (because we have *noise*)

To a control MOVE_RIGHT, the robot will respond by moving

- right with prob. 0.8
- top-right with prob. 0.1
- bottom-right with prob. 0.1

MOVE_RIGHT:



If the target (noise free) cell is occupied,
the robot will stay where it is with probability 1.

A2) Transition Model: With noise

Introduce noise into the transition:

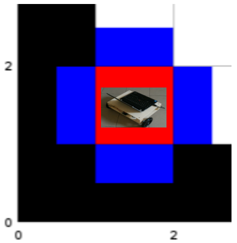
```
#our motion is feasible - compute resulting transition
switch (control_input_)
  case MOVE_UP
    if (translation_rows     ==  1 && translation_cols ==  0) transition_probability_matrix(row, col) = 0.8; #desired motion
    elseif (translation_rows ==  1 && translation_cols ==  1) transition_probability_matrix(row, col) = 0.1;
    elseif (translation_rows ==  1 && translation_cols == -1) transition_probability_matrix(row, col) = 0.1;
    endif;
  case MOVE_DOWN
    if (translation_rows     == -1 && translation_cols ==  0) transition_probability_matrix(row, col) = 0.8; #desired motion
    elseif (translation_rows == -1 && translation_cols ==  1) transition_probability_matrix(row, col) = 0.1;
    elseif (translation_rows == -1 && translation_cols == -1) transition_probability_matrix(row, col) = 0.1;
    endif;
  case MOVE_LEFT
    if (translation_rows     ==  0 && translation_cols == -1) transition_probability_matrix(row, col) = 0.8; #desired motion
    elseif (translation_rows ==  1 && translation_cols == -1) transition_probability_matrix(row, col) = 0.1;
    elseif (translation_rows == -1 && translation_cols == -1) transition_probability_matrix(row, col) = 0.1;
    endif;
  case MOVE_RIGHT
    if (translation_rows     ==  0 && translation_cols ==  1) transition_probability_matrix(row, col) = 0.8; #desired motion
    elseif (translation_rows ==  1 && translation_cols ==  1) transition_probability_matrix(row, col) = 0.1;
    elseif (translation_rows == -1 && translation_cols ==  1) transition_probability_matrix(row, col) = 0.1;
    endif;
endswitch
```

Now we cannot be certain that grid-orazio is moving into the desired direction.

B) Observation Model

Each bumper can be toggled or not.

4 bumpers result in 16 possible configurations:




```
                # [u, d, l, r]
observations = [0, 0, 0, 0]; # no observation

observations = [0, 0, 0, 1]; # right

observations = [0, 0, 1, 0]; # left

observations = [0, 0, 1, 1]; # left-right

observations = [0, 1, 0, 0]; # down
...
```

B) Observation Model

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);
```

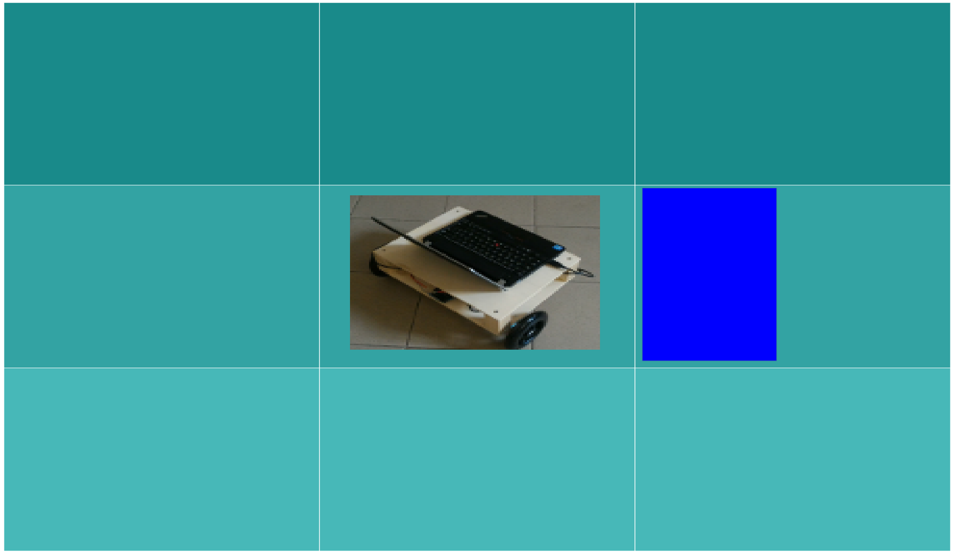
that given:

- 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: With noise

We have:

```
#update probability depending on observations
observation_probability = 1;
if (up_occupied == observations_(1))
  observation_probability *= .8;
else
  observation_probability *= .2;
endif
if (down_occupied == observations_(2))
  observation_probability *= .8;
else
  observation_probability *= .2;
endif
if (left_occupied == observations_(3))
  observation_probability *= .8;
else
  observation_probability *= .2;
endif
if (right_occupied == observations_(4))
  observation_probability *= .8;
else
  observation_probability *= .2;
endif
```

Hence we might obtain bumper readings for cells which are actually not occupied.

Localizing grid-orazio

We have knowledge of:

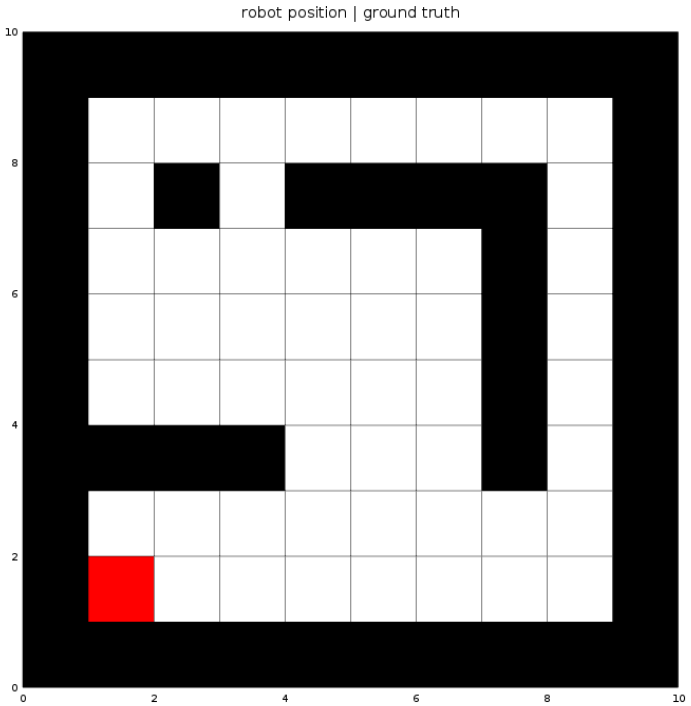
- the controls grid-orazio receives (MOVE_..)
- 0-4 observations from the bumpers

We want to determine the distribution over all possible locations on the map using this information.

State

The domain is discrete.

The feasible states include all free cells in a grid, i.e. in the map matrix

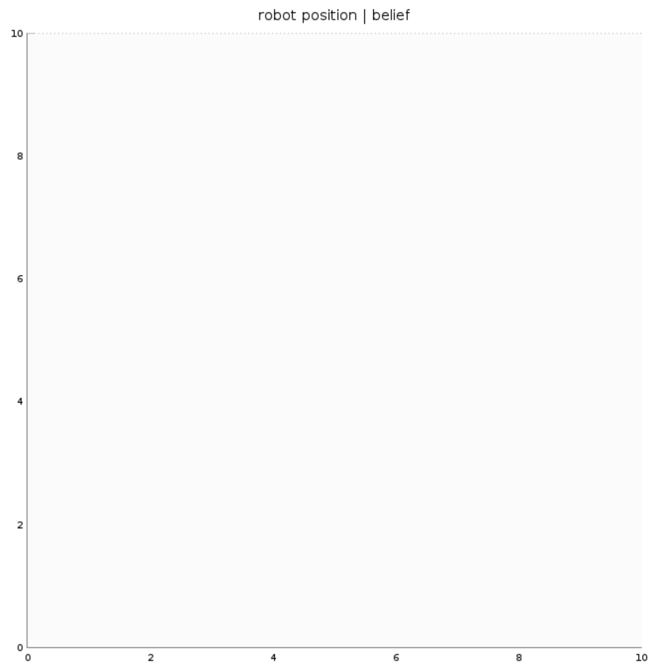


Belief

The belief should contain a probability value for each state.

```
#initialize state_belief over the complete grid
number_of_free_cells = rows(map)*columns(map);
belief_initial_value = 1/(number_of_free_cells);

state_belief = ones(rows(map), columns(map))*belief_initial_value;
```



Belief: Predict & Update

C) Predict belief

```
for row = 1:map_rows
  for col = 1:map_cols
    state_belief += transitionModel(map, row, col, control_input)*state_belief_previous(row, col);
  endfor
endfor
```

D) Update belief

```
inverse_normalizer = 0;
for row = 1:map_rows
  for col = 1:map_cols
    state_belief(row, col) *= observationModel(map, row, col, observations);
    inverse_normalizer      += state_belief(row, col);
  endfor
endfor

#normalize the belief probabilities to [0, 1]
normalizer = 1./inverse_normalizer;
state_belief *= normalizer;
```

Test it

- In a console run:

  octave grid_localizer <map_file.txt>

- grid-orazio can be controlled with the keys W,A,S,D

You will notice that issuing a motion command has non-deterministic effects.

Observe the “belief” window, and see the probability mass changing as grid-orazio explores the map.

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?