Probabilistic Robotics Course

Discrete Filtering

Giorgio Grisetti
grisetti@dis.uniroma1.it

Dept of Computer Control and Management Engineering
Sapienza University of Rome

Outline

- Grid-Orazio
- Modeling the problem
  - Transition Model
  - Observation Model
- Synthesis of the solution

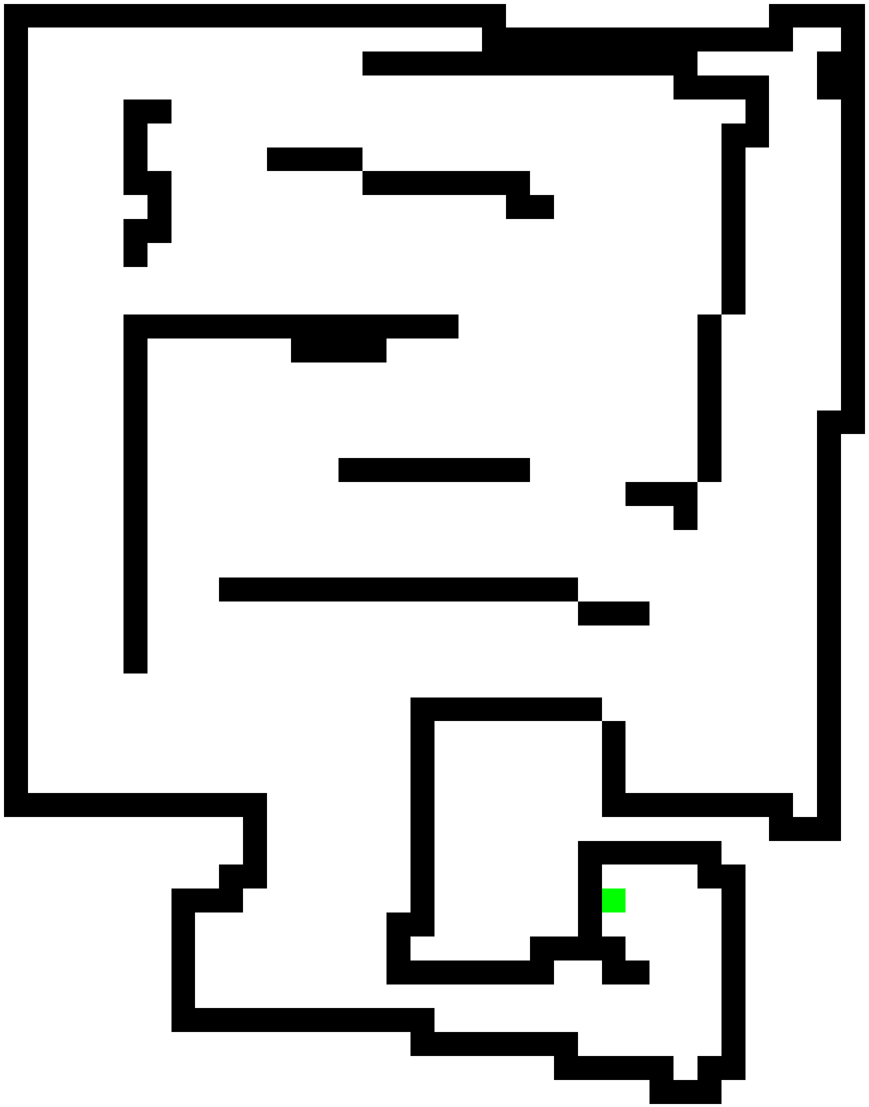
Scenario

Grid-Orazio lives in a grid world

The cells of this world are either free or occupied

At each point in time Grid-Orazio can receive one of the 4 commands to move up/down/left/right

Orazio senses the state around it with 4 noisy bumpers mounted at its 4 sides



Localizing Grid-Orazio

We can observe:

- the controls orazio receives
- the measurements from the bumpers

We want to determine the distribution over possible locations, after observing

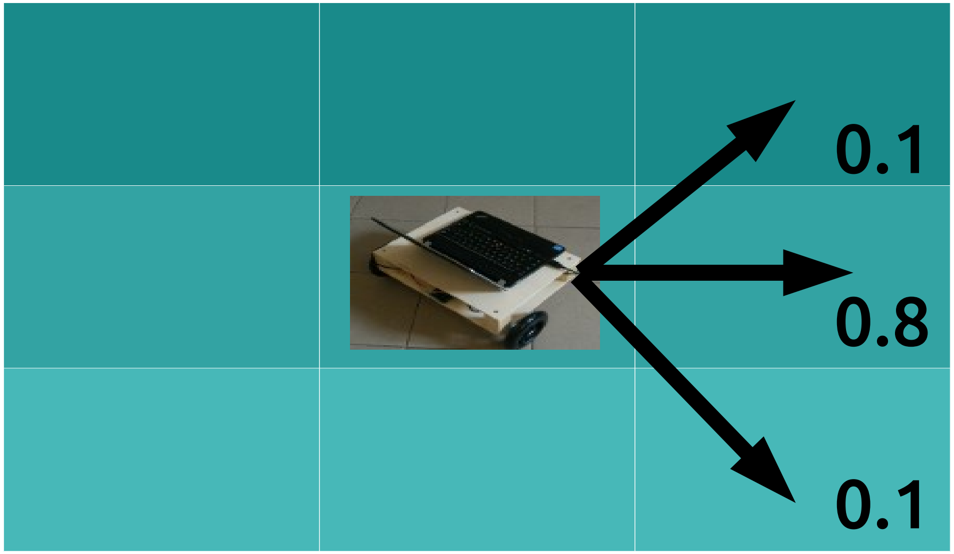
- measurements
- controls

Modeling the Controls

The controls we issue to Grid-Orazio, do not have a deterministic effect

To a control “go-right”, the robot will respond by moving

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



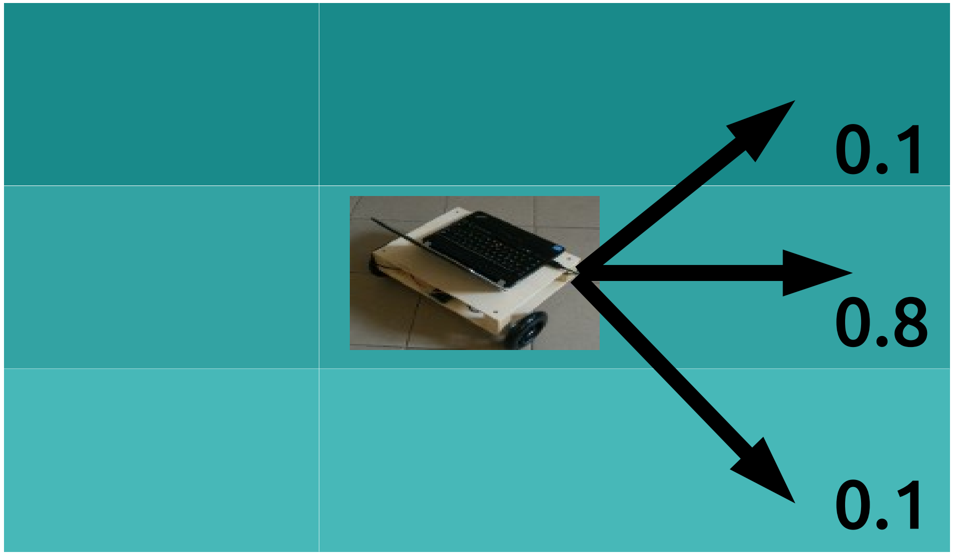
Modeling the Controls

The controls we issue to Grid-Orazio, do not have a deterministic effect

To a control “go-right”, the robot will respond by moving

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

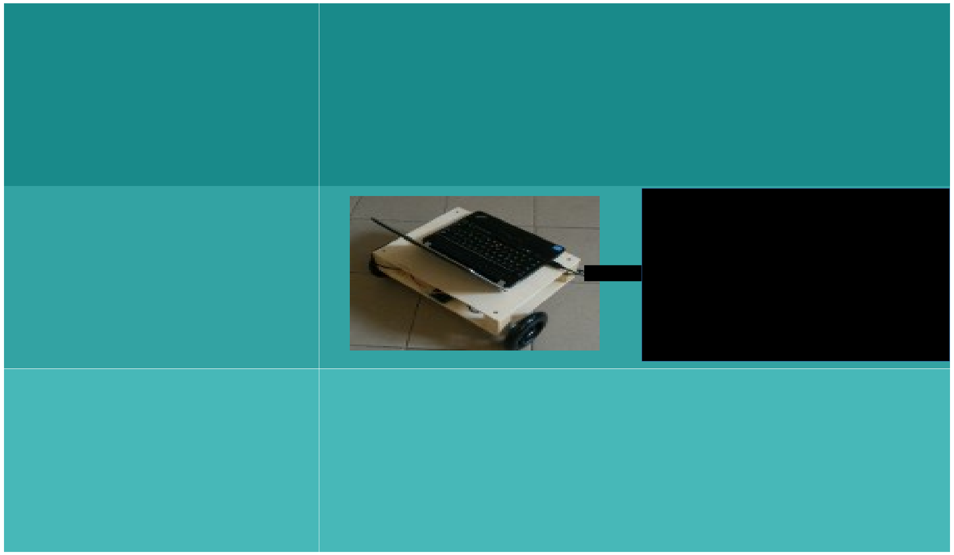
This holds if the destination is not occupied



The behavior is symmetric for all 4 controls

Modeling the Controls

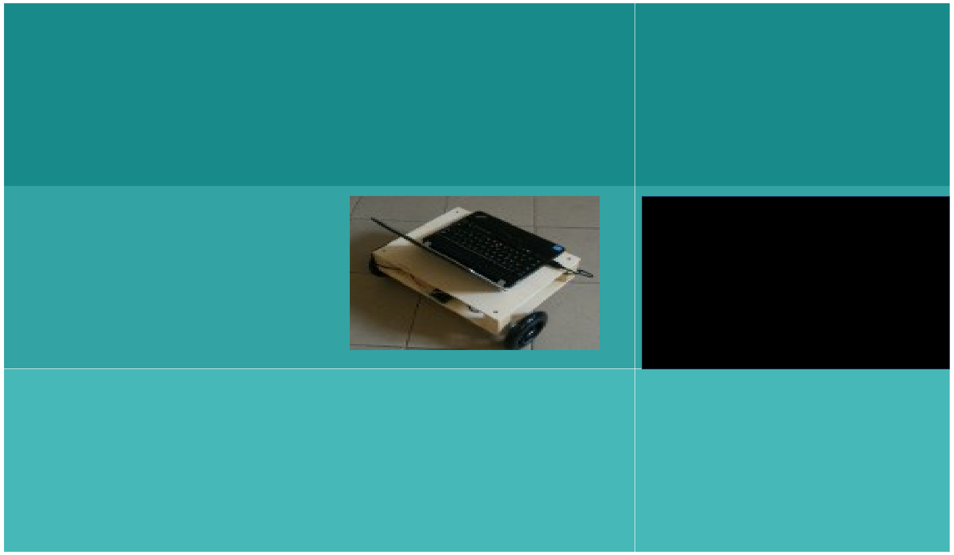
If the noise-free destination is occupied, the robot will stay where it is with probability 1



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

$p(Z^{right}=\text{toggled})=0.8$

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

Implement 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

Map

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

A map is a grid, conveniently represented by an opencv matrix, with the convention that a pixel having 0 value represents a cell

```
cv::Mat _map;
```

State

The domain is discrete.

The feasible states include all free cells in a grid, and it can be mapped to an integer set

For convenience we will introduce two mappings

```
cv::Mat _cell_to_index;
std::vector< std::pair<int, int> > _index_to_cell;
```

Belief

The belief should contain a probability value for each state.

```
typedef std::vector <float> Belief;

...

Belief _belief;
```

Actions

Actions are just an enum, that can be mapped to an integer in C/C++

```
enum Action {Up=0, Down=1, Left=2, Right=3};
```

Observations

Each bumper can be toggled or not.

4 bumpers result in 16 possible configurations, that fit comfortably in the 4 least significant bits of a char.

This is necessary, so we can map our observations into integers/chars and use the implementation of the previous lesson.

```
typedef char Observation;
```

Transition Model

We need to implement a function in the form

```
float transitionModel(int from, int to, Action action)
```

that given:

- a start state
- a destination state
- an action

returns the probability of the transition

Transition Model

Retrieve from the index of start and end state the cell of the grid

```
std::pair<int, int> from_cell = _index_to_cell[from];
std::pair<int, int> to_cell = _index_to_cell[to];
```

If the two cells are farther than 1, the probability is 0 (the robot can move only to nearby cells)

```
int dr = to_cell.first - from_cell.first ;
int dc = to_cell.second - from_cell.second;
if (fabs(dr)>1 || fabs(dc)>1)
  return 0;
```

Transition Model

If the destination is out of the map, or it is occuoied then the probability is

- 1, if the source is the same as the destination
- 0 otherwise

Retrieve the “noise free” next state based on the action

```
// compute the leading cell, based on the current state and the action
  int r = from_cell.first;
  int c = from_cell.second;

  switch (action) {
  case Up: r--; break;
  case Down: r++; break;
  case Left: c--; break;
  case Right: c++; break;
  }
```

Transition Model

```
bool invalid_motion = false;
if (r<0 || r>_map.rows-1 || c<0 || c>_map.rows-1)
  invalid_motion = true;

cv::Vec3b v = _map.at<cv::Vec3b>(r,c);
 if (v[0]!=255)
   invalid_motion=true;

if (invalid_motion){
  if (dr==0 && dc==0)
    return 1;
  return 0;
}
```

Transition Model

If none of the other cases occurred, we compute the probability of the transition

```
  switch (action) {
  case Up:
    if (dr==-1 && dc == 0) return 0.8;
    if (dr==-1 && dc == 1) return 0.1;
    if (dr==-1 && dc == -1) return 0.1;
    return 0;
  case Down:
    if (dr==1 && dc == 0) return 0.8;
    if (dr==1 && dc == 1) return 0.1;
    if (dr==1 && dc == -1) return 0.1;
    return 0;
…...
  }
}
```

Observation Model

We need to implement a function

```
float LocalizerMap::observationModel(int state, Observation z)
```

that returns the probability of an observation in a state

to predict the observation, we need

- to retrieve cell of the map, given the index
- based on the cell we need to retrieve the occupancy state of all cells around the current location
- use the independent observation model of each bumper to compute the likelihood of the prediction

Observation Model

Retrieve the occupancy around the state

```
std::pair<int, int> state_cell = _index_to_cell[state];
int r = state_cell.first;
int c = state_cell.second;

bool up_occupied= true;
if (r>0) {
  cv::Vec3b v = _map.at<cv::Vec3b>(r-1,c);
  if (v[0]==255)
    up_occupied = false;
}

bool down_occupied= true;
if (r<_map.rows-1) {
  cv::Vec3b v = _map.at<cv::Vec3b>(r+1,c);
  if (v[0]==255)
    down_occupied = false;
}.....
```

Observation Model

```
float cumulative_prob = 1;
if (up_occupied == (bool)(z & 0x01))
  cumulative_prob *= .8;
else
  cumulative_prob *= .2;

if (down_occupied == (bool) (z & 0x02))
  cumulative_prob *= .8;
else
  cumulative_prob *= .2;....
```

Observation Model

`cumulative_prob` **will contain the product of the 4 independent likelihoods and thus**

```
return cumulative_prob;
```

will be the last instruction.

Predict

```
void predict(Action action) {
  Belief _b2(_b.size());
  std::fill(_b2.begin(), _b2.end(), 0);
  for (size_t from = 0; from<_b.size(); from++) {
    if (_b[from] == 0)
      continue;

    for (size_t to = 0; to<_b.size(); to++) {
      _b2[to] += _b[from] *
                 transitionModel(from, to, action);
    }
  }
  _b = _b2;
}
```

Update

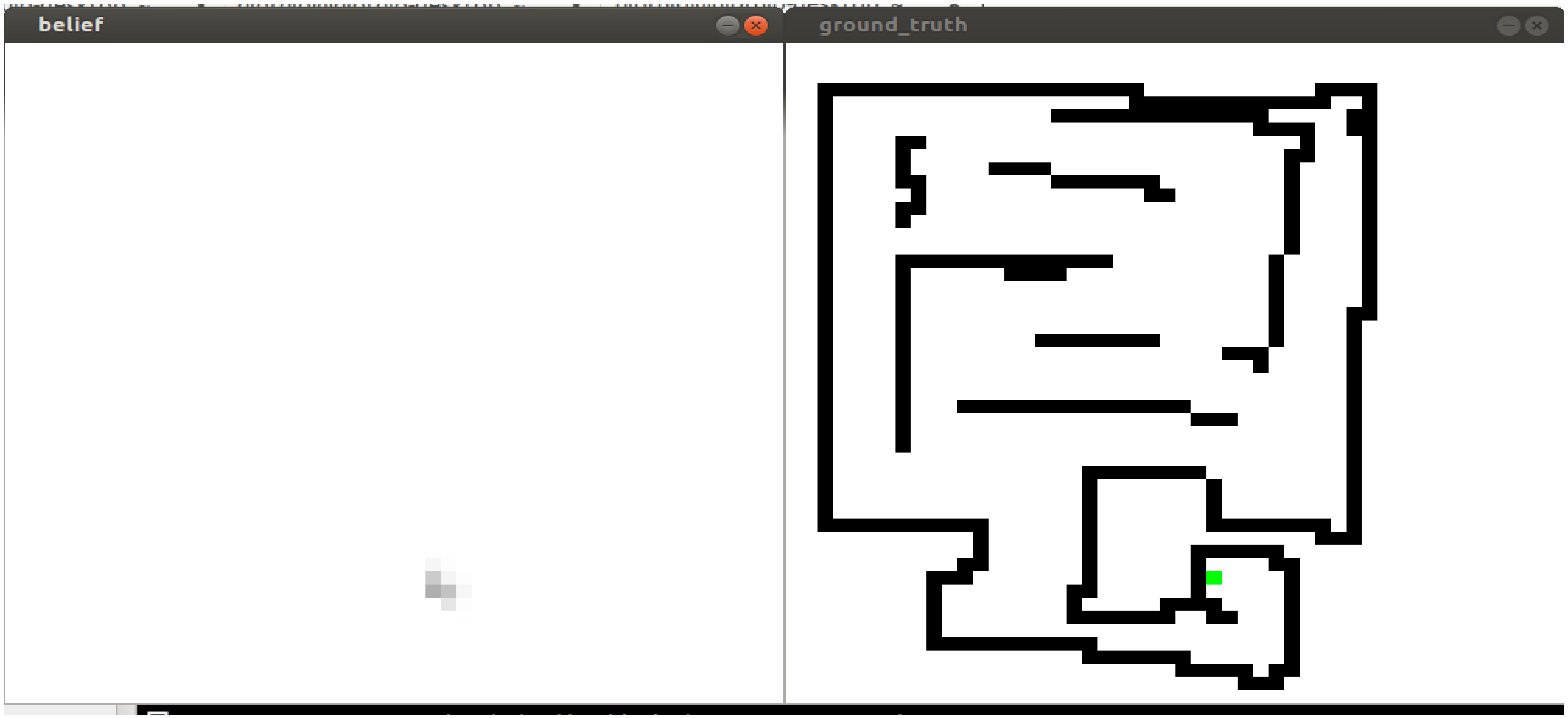
```
void update(Observation z) {
  double normalizer = 0;
  for (size_t x = 0; x<_b.size(); x++) {
    _b[x] *= observationModel(x, z);
    normalizer += _b[x];
  }
  normalizer = 1./normalizer;
  for (size_t x = 0; x<_b.size(); x++) {
    _b[x] *= normalizer;
  }
}
```

Test it

- Run

  grid_localizer <map>,

  and move the robot with I,j,k,l.
- The program implements also a simulator that will introduce noise to your controls/observations
- You will notice that issuing a motion command has not deterministic effects
- Observe the “belief” window, and see the probability mass changing



Exercise

What if Grid-Orazio has also an orientation, and its commands become

- forward,
- backward,
- rotate left,
- rotate right?

How does the state change?

What about the observation and transition model?

Summary

- ....