Automatic update of the Knowledge Base with sensory inputs

Gerard Canal, Research Assistant

KING'S College LONDON

Previously in ROSPlan...

- Actions have a single outcome

Previously in ROSPlan...

- Actions have a single outcome
  - Knowledge Base is updated by the ActionInterface

Previously in ROSPlan...

- Actions have a single outcome
  - Knowledge Base is updated by the ActionInterface
- However…
  - What if unexpected outcomes happen?

Previously in ROSPlan...

- Actions have a single outcome
  - Knowledge Base is updated by the ActionInterface
- However…
  - What if unexpected outcomes happen?
  - What if parts of the state are exogenous and come from outside?

Planning in Robotics

- Uncertain domains → Chance of failure!

Planning in Robotics

- Uncertain domains → Chance of failure!
- Two approaches:

Planning in Robotics

- Uncertain domains → Chance of failure!
- Two approaches:
  - Deterministic planning + replanning on failure

Planning in Robotics

- Uncertain domains → Chance of failure!
- Two approaches:
  - Deterministic planning + replanning on failure
  - Probabilistic planning optimizing success probability

Planning in Robotics

- Uncertain domains → Chance of failure!
- Two approaches:
  - Deterministic planning + replanning on failure
  - Probabilistic planning optimizing success probability
    - More on that tomorrow...

Non-deterministic effects

- Deterministic effects always happen (if action succeeds)

Non-deterministic effects

- Deterministic effects always happen (if action succeeds)
- Non-deterministic effects create divergent paths

Non-deterministic effects

- Deterministic effects always happen (if action succeeds)
- Non-deterministic effects create divergent paths
  - Need to update KB accordingly!
  - How to detect what happened?

Non-deterministic effects

- Deterministic effects always happen (if action succeeds)
- Non-deterministic effects create divergent paths
  - Need to update KB accordingly!
  - How to detect what happened?
    - SENSORS!

Keeping KB up-to-date

- To overcome potential issues, KB must be kept up-to-date… Manually!

Keeping KB up-to-date

- To overcome potential issues, KB must be kept up-to-date… Manually!
- But the use case in most cases is the same:
    1. Get sensor data / state of the world

Keeping KB up-to-date

- To overcome potential issues, KB must be kept up-to-date… Manually!
- But the use case in most cases is the same:
    1. Get sensor data / state of the world
    2. Compute predicate values

Keeping KB up-to-date

- To overcome potential issues, KB must be kept up-to-date… Manually!
- But the use case in most cases is the same:
    1. Get sensor data / state of the world
    2. Compute predicate values
    3. Update KB

Keeping KB up-to-date

- To overcome potential issues, KB must be kept up-to-date… Manually!
- But the use case in most cases is the same:
    1. Get sensor data / state of the world
    2. Compute predicate values
    3. Update KB
    4. Repeat

Doing it automatically

- Hassle-free update of the Knowledge Base

Doing it automatically

- Hassle-free update of the Knowledge Base
- With only **3** lines!

Doing it automatically

- Hassle-free update of the Knowledge Base
- With only **3** lines!
- Automatic subscription to topics and calling of services

Doing it automatically

- Hassle-free update of the Knowledge Base
- With only **3** lines!
- Automatic subscription to topics and calling of services
- We call it the "ROSPlan's Sensing Interface"

The sensing interface

- Update action result needs sensor processing

The sensing interface

- Update action result needs sensor processing
- Tedious to do by hand…

The sensing interface

- Update action result needs sensor processing
- Tedious to do by hand…
- So, we propose an automatic sensing interface:

```
docked:
    - params kenny
    - /mobile_base/sensors/core
    - kobuki_msgs/SensorState
    - msg.charger != msg.DISCHARGING
```

Setting up the sensing interface

- Needed files:
  - rosplan_sensing.launch
  - config_file.yaml
  - predicate_scripts.py (optional)

Adding topics: config_file.yaml

- Syntax:

```
topics:
  predicate_name:
    params:
      - p1/’*’
    topic: /topic_to_subscribe
    msg_type: topic_msg_type
    operation: python string
```

Adding topics: config_file.yaml

- Syntax:

```
topics:
  predicate_name:
    params:
      - p1/*
    topic: /topic_to_subscribe
    msg_type: topic_msg_type
    operation: python string
```

Adding services: config_file.yaml

- Syntax:

```
services:
  predicate_name:
    params:
      - p1/*
  service: /service_to_call
  srv_type: topic_msg_type
  time_between_calls: 10
  request: python string
  operation: python string
```

Adding services: config_file.yaml

- Syntax:

```
services:
  predicate_name:
    params:
      - p1/'*'
  service: /service_to_call
  srv_type: topic_msg_type
  time_between_calls: 10
  request: python string
  operation: python string
```

More complex set-ups

- Sometimes a single line is not enough!

More complex set-ups

- Sometimes a single line is not enough!
- Possibility to add own full methods:

```
functions:
    - $(find rosplan_sensing_interface)/example.py
```

- When the “operation” line is ignored, the method is looked into the python script.

Custom python script

- Three types of methods:
  - Topic msg processing
  - Service request creation
  - Service response processing

Custom python script

- Topic msg processing
  - Define a method with the same name as the predicate.
  - Parameters:
    - Message received
    - Parameters defined in the config file
  - The method returns the result for the predicate

Custom python script

- Service request creation
  - Define a method starting with “_req” and the predicate’s name.
  - No parameters are accepted
  - The method should return the service request creation

Custom python script

- Service response processing
  - Define a method with the same name as the predicate.
  - Parameters are the response and the defined action parameters.
  - The method should return the the predicate’s assignment result.

Extra features...

- Message types are automatically imported.
- Helper functions available to use inside the methods:
  - rospy
  - get_kb_attribute

Let’s play!

- Create a new package and add this files:
  - rps_tutorial.yaml
  - rps_tutorial.py
  - rps_tutorial.launch
- Download the test sensor node:
  - `wget https://bit.ly/3214T12 -O test_client_service.py`

rps_tutorial.launch

```
<?xml version="1.0"?>
<launch>

    <arg name="main_rate"   default="10"/>

    <node name="rosplan_sensing_interface" pkg="rosplan_sensing_interface"
type="sensing_interface.py" respawn="false" output="screen">
        <rosparam command="load" file="$(find
rps_tutorial)/config/rps_tutorial.yaml" />
        <param name="main_rate"  value="$(arg main_rate)" />
    </node>

</launch>
```

rps_tutorial.yaml

- robot_at predicate from topic.
- docked predicate from service.

rps_tutorial.yaml

```
topics:
  robot_at:
    params:
        - kenny
        - wp0
    topic: /chatter
    msg_type: std_msgs/String
    operation: "int(msg.data.split(' ')[-1])%2 == 0"
```

Launching ROSPlan...

```
roslaunch rosplan_planning_system interfaced_planning_system.launch
domain_path:=$(rospack find
rosplan_demos)/common/domain_turtlebot.pddl problem_path:=$(rospack
find rosplan_demos)/common/problem_turtlebot.pddl
```

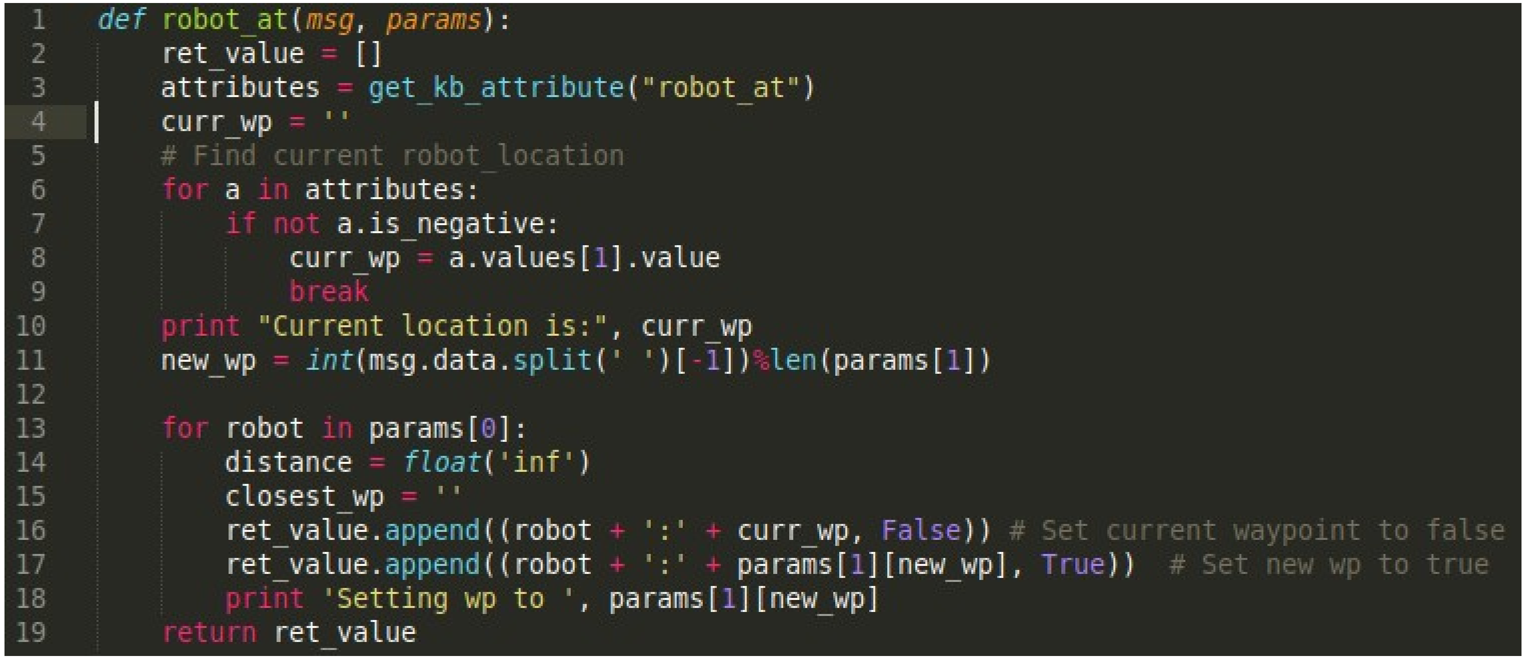
Launching the Sensing Interface...

```
roslaunch rps_tutorial rps_tutorial.launch
```

Our sensors...

- Run the sensor node:
  - `rosrun rps_tutorial test_client_service.py`
  - Check the topics and services available…

Adding complexity...



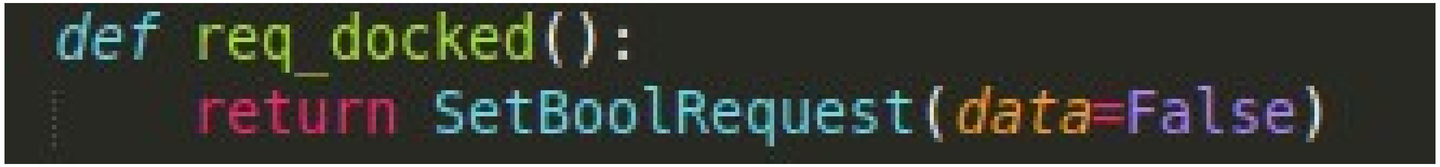
```
def robot_at(msg, params):
    ret_value = []
    attributes = get_kb_attribute("robot_at")
    curr_wp = ''
    # Find current robot_location
    for a in attributes:
        if not a.is_negative:
            curr_wp = a.values[1].value
            break
    print "Current location is:", curr_wp
    new_wp = int(msg.data.split(' ')[-1])%len(params[1])

    for robot in params[0]:
        distance = float('inf')
        closest_wp = ''
        ret_value.append((robot + ':' + curr_wp, False)) # Set current waypoint to false
        ret_value.append((robot + ':' + params[1][new_wp], True))  # Set new wp to true
        print 'Setting wp to ', params[1][new_wp]
    return ret_value
```

rps_tutorial.yaml

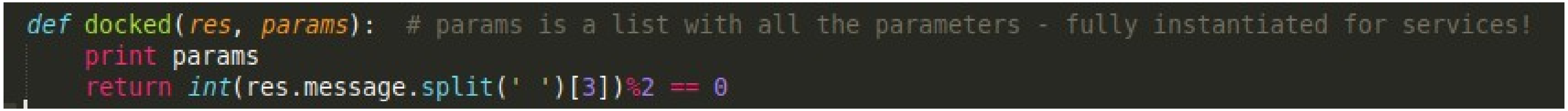
```
services:
  docked:
      params:
          - kenny
      service: /test_service # Service
      srv_type: std_srvs/SetBool # Srv type
      time_between_calls: 1 # Time between calls in seconds
      request: SetBoolRequest(data=True) # Request creation
      operation: "int(res.message.split(' ')[3])%2 == 0" # operation
```

Adding complexity...



```
def req_docked():
    return SetBoolRequest(data=False)
```

rps_tutorial.yaml



```python
def docked(res, params):  # params is a list with all the parameters - fully instantiated for services!
    print params
    return int(res.message.split(' ')[3])%2 == 0
```

Thank you for your attention!

Questions are welcomed!