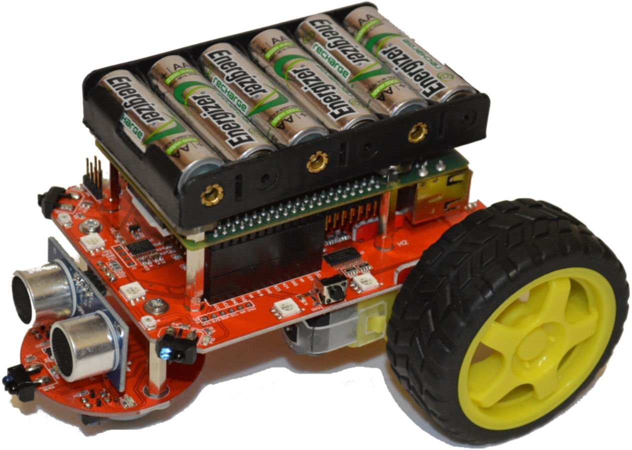
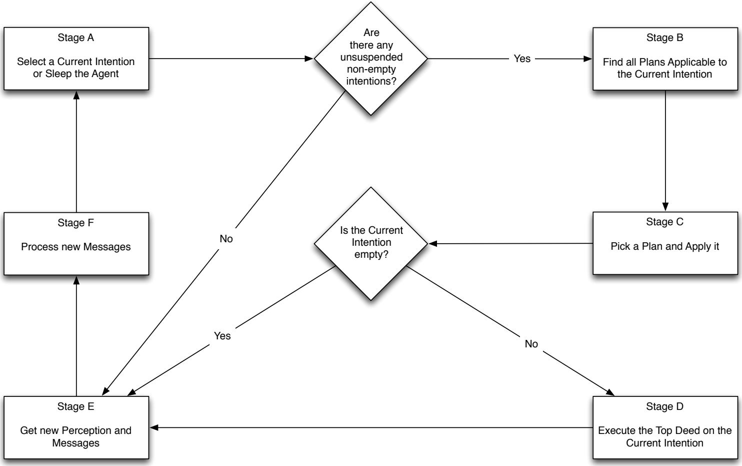
BDI Programming in Python: Louise Dennis

October 2, 2017

MOTIVATION



BDI Programming Languages



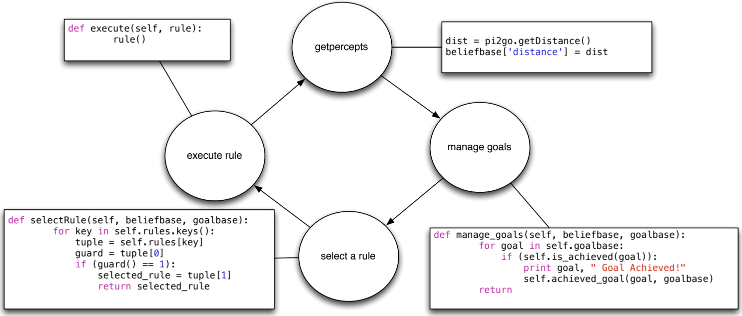
A BDI Program

```
:Initial Beliefs:

possible_rubble(1, 1), possible_rubble(3, 3), possible_rubble(5

:Reasoning Rules:

square_to_check(X, Y) :- possible_rubble(X, Y), ~no_rubble(X, Y
done :- holding(rubble);
done :- ~ (possible_rubble(X, Y), ~no_rubble(X, Y));

:Initial Goals:

done [achieve]

:Plans:

+!done [achieve] : {B square_to_check(X, Y)} ← move_to(X, Y);
+at(X, Y) : {~B rubble(X, Y)} ← +no_rubble(X, Y);
+rubble(X, Y): {B at(X, Y)} ← lift_rubble;
+holding(rubble): {True} ← print(done);
```

BDI Programming for Python

```
import pi2goagent

agent = pi2goagent.Agent()

def print_obstacle_rule():
   print ("Obstacle: ", agent.sensor_value('obstacle_centre'))
   return

def stop_rule():
   print("Stopping Agent")
   agent.done()
   return

agent.add_condition_rule(agent.believe('switch_pressed'), stop_rule)
agent.add_rule(print_obstacle_rule)
agent.run_agent()
```

Reasoning Cycle



POPULATING THE BELIEF BASE

```
def getpercepts(self, beliefbase):
        dist = robohat.getDistance()
        beliefbase['distance'] = dist
        irR = robohat.irRight()
        beliefbase['obstacle_right'] = irR
        irL = robohat.irLeft()
        beliefbase['obstacle_left'] = irL
        irC = robohat.irCentre()
        beliefbase['obstacle_centre'] = irC
        irLL = robohat.irLeftLine()
        beliefbase['no_line_left'] = irLL
        irRL = robohat.irRightLine()
        beliefbase['no_line_right'] = irRL
        switch = robohat.getSwitch()
        beliefbase['switch_pressed']= switch
        ....
        time.sleep(0.1)
        return
```

Goals, More Complex Rule Conditions

```
def b_in_the_light():
        lightFL = agent.sensor_value('lightFL')
        lightFR = agent.sensor_value('lightFR')
        if (lightFL > 250 and lightFR > 250):
                return 1
        return 0
...
agent.goal_is_achieved_when('in_the_light', b_in_the_light)
...
b_started = agent.B('started')
...
b_can_move = agent.AND(b_started, agent.G('in_the_light'))
b_can_turn_left = agent.AND(b_can_move, agent.NOT(b_turning_left))
...
agent.add_condition_rule(agent.AND(b_can_turn_left, b_light_on_left),
agent.add_condition_rule(agent.AND(b_can_turn_right, b_light_on_right
agent.add_condition_rule(agent.AND(b_in_the_light, b_moving), stop_mot
```

PROBLEM: PREDICATE LOGIC

+!done [achieve] : B square_to_check(X, Y) <− move_to(X, Y);

Possible Syntaxes: Implicit argument passing (like map, filter, etc.)

add_rule(agent.believe(square_to_check), move_to)

- Pros: Familiar to people used to map and filter.
- Cons: Ambiguous: B father(X, Y), B widowed(X) becomes agent.AND(agent.believe(father), agent.believe(widowed))

POSSIBLE SYNTAXES: BELIEF EXPLICITY RETURN UNIFIER

add_rule(agent.believe(square_to_check, 'X', 'Y'), move_to('X', 'Y')

- Pros: Still mostly uses python.
- Pro/Con: Forces beliefs to distinguish between parameters and output variables.
- Con: Looks strange to everyone.

Possible Syntaxes: Predicate logic in a string

```
add_rule(logic("EXISTS X, Y. square_to_check(X, Y)") , move_to('X', 'Y'))
```

- Pros: Can express unambiguously what we mean.
- Con: Abandons any pretence that this is Python (but only needed when predicate logic is used).

Possible Solution: Bespoke Functions for Common uses of Predicate Logic

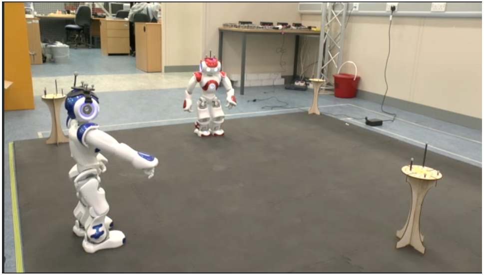
```
import logictestagent

agent = logictestagent.LogicAgent()

def mycmp(c1, c2):
    scores = agent.belief_value(agent.B('scores'))

    if scores[c1] > scores[c2]:
        return 1;
    else:
        return 0;


def print_choice_rule(choice):
    print (choice)
    return


agent.add_pick_best_rule(agent.B('choice'), mycmp, print_choice_rule)
agent.run_agent();
```

Application: A Verifiable Ethical Engine

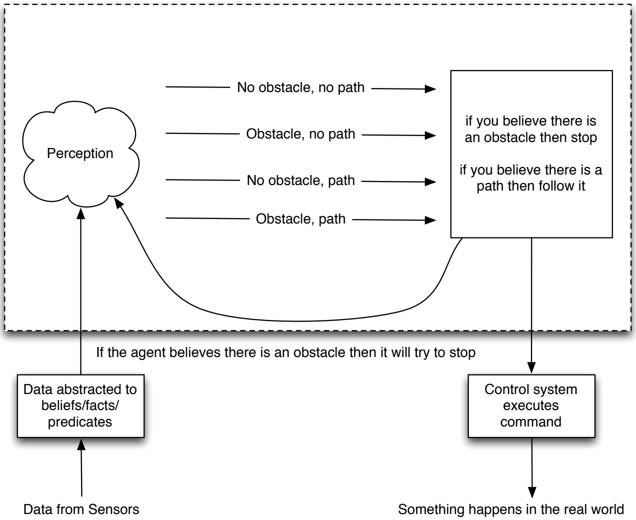
BDI Agent reasons about scores and picks a path

Path Planning, Machine Learning used to pick paths to score

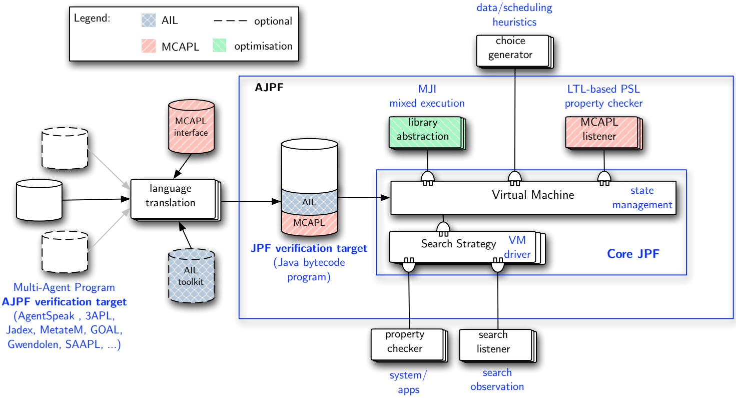
Control, NAOqui Interface, Vicon System



HOW DO YOU VERIFY A ROBOTIC SYSTEM?



How do you Verify a Python Agent?



CURRENT STATUS

```
import pi2goagent, pi2go

agent = pi2goagent.Agent()

def stop_rule():
print("Stopping Agent")
agent.done()
return

def forward_rule():
print("Going Forward")
pi2go.forward(20)
return

agent.add_condition_rule(agent.B('switch_pressed'), stop_rule)
agent.add_rule(forward_rule)
agent.run_agent()
```

QUESTIONS?