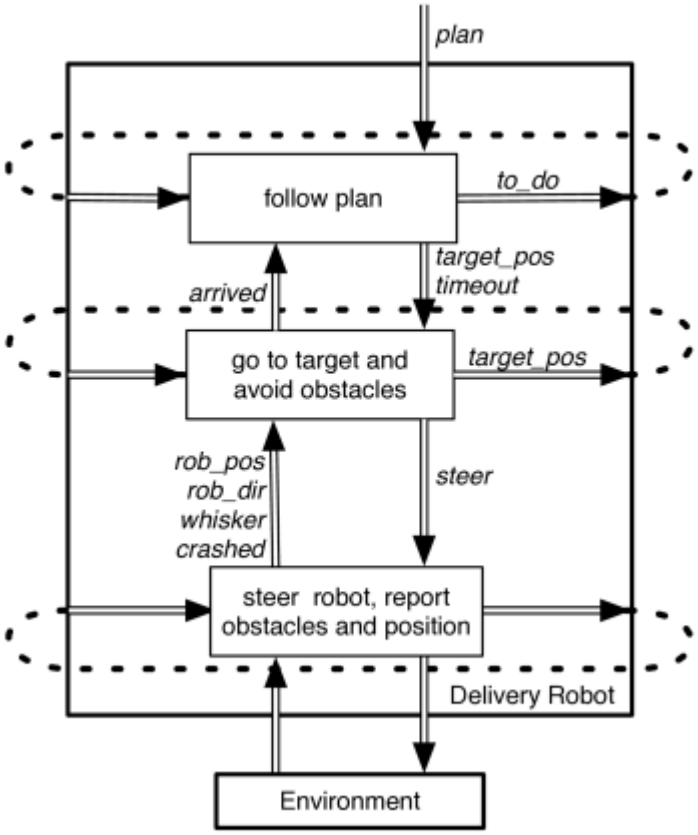
Delivery Robot Example (display.py, agents.py, agentEnv.py, agentMiddle.py, agentTop.py):



A hierarchical decomposition of the delivery robot

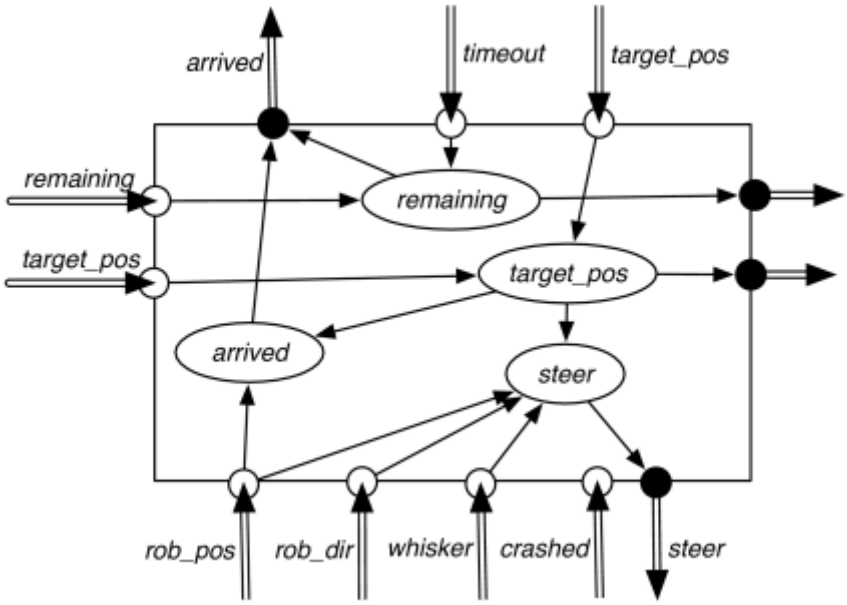
Consider a Delivery Robot able to carry out high-level navigation tasks while avoiding obstacles. The delivery robot is required to visit a sequence of named locations in the environment, avoiding obstacles it may encounter.
Assume the delivery robot has wheels, like a car, and at each time can either go straight, turn right, or turn left. It cannot stop. The velocity is constant and the only command is to set the steering angle. Turning the wheels is instantaneous, but turning to a certain direction takes time. Thus, the robot can only travel straight ahead or go around in circular arcs with a fixed radius.

The robot has a position sensor that gives its current coordinates and orientation. It has a single whisker sensor that sticks out in front and slightly to the right and detects when it has hit an obstacle. In the example below, the whisker points 30∘ *to the right of the direction the robot is facing. The robot does not have a map, and the environment can change with obstacles moving.*
A layered controller for the delivery robot is shown in the figure above. The robot is given a high-level plan to execute. The robot needs to sense the world and to move in the world in order to carry out the plan. The details of the lowest layer of the controller are not shown in this figure.
The top layer, called follow plan. *That layer takes in a plan to execute. The plan is a list of named locations to visit in sequence. The locations are selected in order. Each selected location becomes the current target. This layer determines the* x-y *coordinates of the target. These coordinates are the target position for the middle layer. The top layer knows about the names of locations, but the lower layers only know about coordinates.*

The top layer maintains a belief state consisting of a list of names of locations that the robot still needs to visit. It issues commands to the middle layer to go to the current target position but not to spend more than timeout steps. The percepts for the top layer are whether the robot has arrived at the target position or not. So the top layer abstracts the details of the robot and the environment.
The middle layer, which could be called go to target and avoid obstacles*, tries to keep traveling toward the current target position, avoiding obstacles. . The target position, target_pos, is received from the top layer. The middle layer needs to remember the current target position it is heading towards. When the middle layer has arrived at the target position or has reached the timeout, it signals to the top layer whether the robot has arrived at the target. When arrived becomes true, the top layer can change the target position to the coordinates of the next location on the plan.*

The middle layer can access the robot's position, the robot's direction and whether the robot's whisker sensor is on or off. It can use a simple strategy of trying to head toward the target unless it is blocked, in which case it turns left.
The middle layer is built on a lower layer that provides a simple view of the robot. This lower layer could be called steer robot and report obstacles and position. *It takes in steering commands and reports the robot's position, orientation, and whether the whisker sensor is on or off.*
Inside a layer are features that can be functions of other features and of the inputs to the layers. In the graphical representation of a controller, there is an arc into a feature from the features or inputs on which it is dependent. The features that make up the belief state can be written to and read from memory.
In the controller code in the following two examples, do C means that C is the command for the lower level to do.

The middle go to location and avoid obstacles *layer steers the robot towards a target position while avoiding obstacles. The inputs and outputs of this layer are given in the figure below*
The layer receives two high-level commands: a target position to head towards and a timeout, which is the number of steps it should take before giving up. It signals the higher layer when it has arrived or when the timeout is reached.
The robot has a single whisker sensor that detects obstacles touching the whisker. The one bit value that specifies whether the whisker sensor has hit an obstacle is provided by the lower layer. The lower layer also provides the robot position and orientation. All the robot can do is steer left by a fixed angle, steer right, or go straight. The aim of this layer is to make the robot head toward its current target position, avoiding obstacles in the process, and to report when it has arrived.



The middle layer of the delivery robot

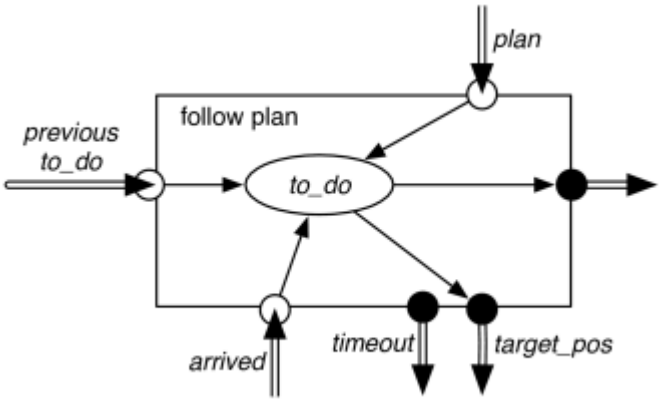
This layer of the controller needs to remember the target position and the number of steps remaining. The command function specifies the robot's steering direction as a function of its inputs and whether the robot has arrived.
The robot has arrived if its current position is close to the target position. Thus, arrived is function of the robot position and previous target position, and a threshold constant:
arrived()≡distance(target_pos,rob_pos)<threshold

where distance is the Euclidean distance, and threshold is a distance in the appropriate units.
The robot steers left if the whisker sensor is on; otherwise it heads toward the target position. This can be achieved by assigning the appropriate value to the steer *variable, given an integer timeout and target_pos:*

```
remaining:=timeout
while not arrived() and remaining≠0
        if whisker_sensor=on
                then steer:=left
        else if straight_ahead(rob_pos,robot_dir,target_pos)
                then steer:=straight
        else if left_of(rob_pos,robot_dir,target_pos)
                then steer:=left
        else steer:=right
        do(steer)
        remaining:=remaining−1
tell upper layer arrived()
```

Where straight_ahead(rob_pos,robot_dir,target_pos) *is true when the robot is at rob_pos, facing the direction robot_dir, and when the current target position, target_pos, is straight ahead of the robot with some threshold (for later examples, this threshold is* 11∘ *of straight ahead). The function left_of ests if the target is to the left of the robot.*

The top layer, follow plan*, is given a plan – a list of named locations to visit in order. These are the kinds of targets that could be produced by a planner. The top layer must output target coordinates to the middle layer, and remember what it needs to carry out the plan. The layer is shown in the figure below.*



The top layer of the delivery robot controller

This layer remembers the locations it still has to visit. The to_do feature has as its value a list of all pending locations to visit.

Once the middle layer has signalled that the robot has arrived at its previous target or it has reached the timeout, the top layer gets the next target position from the head of the to_do list. The plan given is in terms of named locations, so these must be translated into coordinates for the middle layer to use. The following code shows the top layer as a function of the plan:

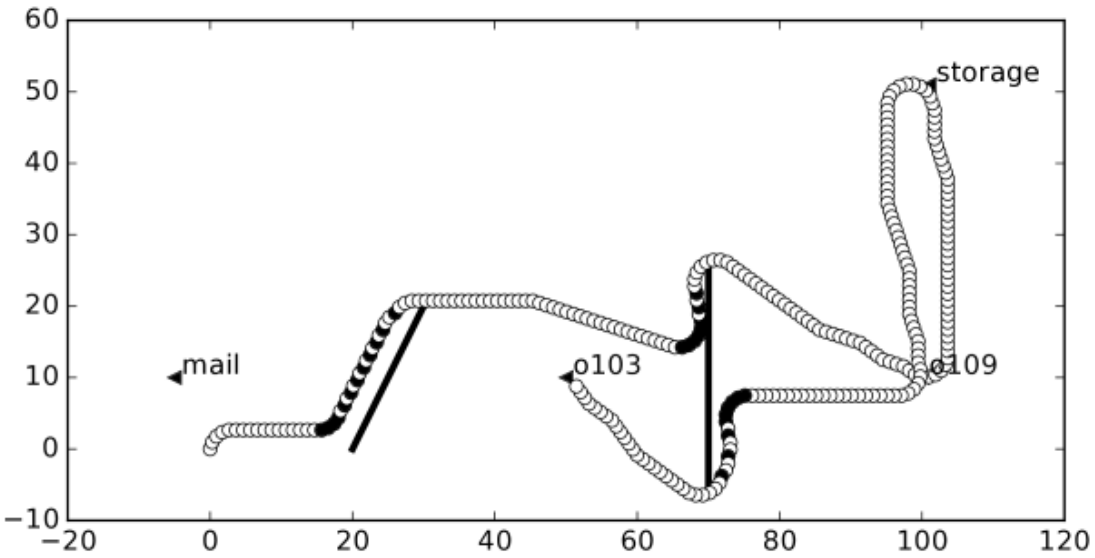
```
to_do:=plan
timeout:=200
while not empty(to_do)
        target_pos:=coordinates(first(to_do))
        do(timeout,target_pos)
        to_do:=rest(to_do)
```

where first(to_do) *is the first location in the to_do list, and* rest(to_do) *is the rest of the to_do list. The function* coordinates(loc) *returns the coordinates of a named location loc. The controller tells the lower layer to go to the target coordinates, with a timeout here of 200 (which, of course, should be set appropriately).* empty(to_do) *is true when the to_do list is empty.*

This layer determines the coordinates of the named locations. This could be done by simply having a database that specifies the coordinates of the locations. Using such a database is sensible if the locations do not move and are known a priori. If the locations can move, the lower layer must be able to tell the upper layer the current position of a location.

A simulation of the plan [goto(o109), goto(storage), goto(o109), goto(o103)] *with two obstacles is given in the figure down. The robot starts at position* (0,0) *facing North, and the obstacles are shown with lines. The agent does not know about the obstacles before it starts.*

Each layer is simple, and none model the full complexity of the problem. But, together they exhibit complex behavior.



This figure shows a *simulation of the robot carrying out the plan mentioned above. The black lines are obstacles. The robot starts at position* (0,0) *and follows the trajectory of the overlapping circles; the filled circles are when the whisker sensor is on. The robot goes to* o109, *storage,* o109, *and* o103 *in turn.*