Teleo-Reactive Planner On ROS

Hai Wei [haiwei@stanford.edu]

Introduction

With the rapid development in AI, robots nowadays have a number of basic capabilities, e.g. navigation, opening the door, grasping objects, placing/dropping objects, object recognition etc. However, to further exploit of robots, we (or the robot) need to come up with a sequence of actions that will achieve a goal. This is also referring as planning. Traditionally, STRIPS (Stanford Research Institute Problem Solver) language could be used to represent the planning problems in terms of states, actions and goals. Generally, this kind of method will induce searching in the state-space, which is too complicated to be feasible for agent control.

Other than that, people can manually specify all possible conditions. However, in reality, the number of possible things that could happen, and thus the number of cases that such a system needs to handle, is very large. While it is possible to try to "hand code" vast amounts of software to handle every special case that we can think of, in practice this is extremely difficult to do, and would not only be extremely time-consuming, but would also lead to a system that is unlikely to be robust (because of special cases that the programmer missed).

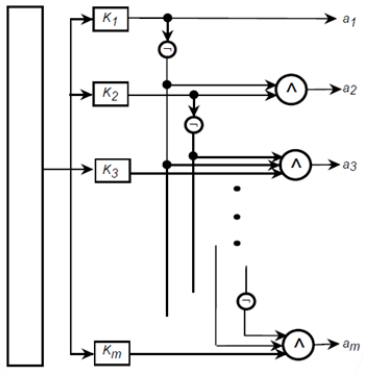
In this project, I built a TR framework based on TR algorithm [1] proposed by Nils Nilsson in 1994, to enable complex applications to be built easily. TR programming represents a systematic way of organizing the high-level decision making problem to cover essentially all possible cases/all possible contingencies that the robot could possibly handle using its basic capabilities. Further, it allows us to ensure that in all of these cases, the robot can select some reasonable action to take to move forward towards the goal.

TR Algorithms and concepts

TR Concepts

TR program consists of a sequence of conditions as their corresponding actions, shown in the figure below. While running, the TR program will check all the conditions (from K_1 toK_m) listed on the left hand side, and searches for the first condition (for example, K_i) that is true. Then, the corresponding action (A_i in this case) will be executed respectively as long as K_i is true. A requirement for user is that the action sequence should be defined in such a way that the durative execution of A_i will eventually lead to at least one condition K_j ($j < i$) being true.

$$
\begin{aligned}
K_1 &\rightarrow a_1 \\
K_2 &\rightarrow a_2 \\
&\cdots \\
K_i &\rightarrow a_i \\
&\cdots \\
K_m &\rightarrow a_m
\end{aligned}
$$



Moreover, TR program can be hierarchical. That is to say, certain action for one TR program could be another TR program. With this feature of recursion, people can start reuse the

code done before. For example, once the tasks, such as making coffee, grabbing stapler and greeting visitor, are accomplished. People can easily achieve the "function", like office assistant, based on the previous medium level tasks.

Implementation on ROS

ROS's communication system: topic and service

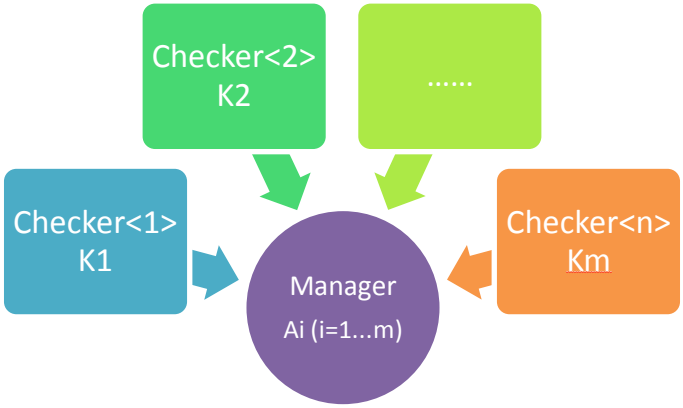
ROS (Robot Operating System) is a framework that supports the development of robotics software. In the ROS, there are a bunch of discrete nodes. The communication between different nodes is done in two different ways: topic and service [2].

Topic: Messages are routed via a transport system with publish/subscribe semantics. A node sends out a message by publishing it to a given Topic. A node that is interested in a certain kind of data will subscribe to the appropriate topic. Logically, a topic is a typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages.

Service: The publish/subscribe model is a very flexible communication paradigm, but its many-to-many one-way transport is not appropriate for request/reply interactions, which are often required in a distributed system. A providing node offers a service under a string name and a consuming node uses the service by sending the request message and awaiting the reply.

TR Manager's structure

The structure of TR framework is shown below. There is one manager, m checkers for the whole framework. Each checker <i> is responsible for condition K_i. In the run time, checkers will check the all the conditions continuously and report to the manger repeatedly. Once the manger gets the information (true/false of all different conditions) from checkers, it will decide which action (the one with highest priority) to take and execute that action as well. Instead of asking manger to check all different conditions sequentially, I use m different nodes to check conditions. The major reason is that some condition checking can be extremely time consuming. Thus, it is better to assign a different node to do that job independently. Besides, all conditions can be checked simultaneously and the robot can observe the world faster.

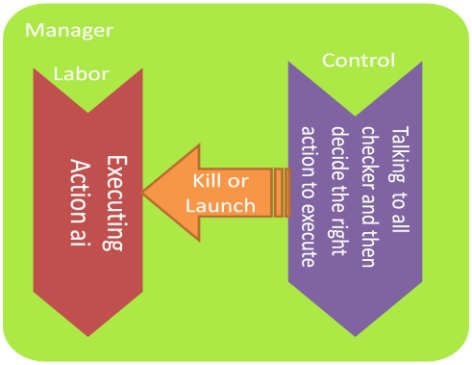


Choice of communication mechanism

From the chart shown above, it is obvious that the one-way communication (from checker to the manager) is sufficient. Hence the topic is used in this case. Other than that, we can also have each check set up a service and let manager loop around (from K_1 toK_m) to query each checker for result. However, there could be a case that certain condition will take hours to do. Consequently, the manager will get stuck, since the manger will wait for the result after sending the querying requirement.

Process control within manager

Since the condition function as well as the action function are all define by the user, which could be compiled executables, process is need to run those executable. Inside the manager, there are two processes. One is the real manager, doing the control job (talking to all checkers, decide the right action to take, etc). The other one actually serves as the labor, who executes the specific action decided by the manager. When starting to take an action, the manager process will launch a labor process and ask it to execute the action. Upon the transition of actions, the manger process, which is also the parent process, will terminate the old action by killing the original old labor process. After the older one is killed, manager will start a new process to take the new action.



TR wrapper

As described above, one needs to run m different checkers and one manager to make the whole system working. This could be tedious to user, when number of conditions is large. To be more user-friendly, I further coded a TR wrapper program to do everything automatically. The user only needs to provide a script file with all conditions and actions. A sample script is like this:

```
Condition#1: /home/haiwei/ros/ros/hai/fCond
Action#1: /home/haiwei/ros/ros/hai/HelloWorld hai wei
Action#2: /home/haiwei/ros/ros/hai/HelloWorld
Condition#2: /home/haiwei/ros/ros/hai/tCond
```

TR Robo simulator for trp on ros

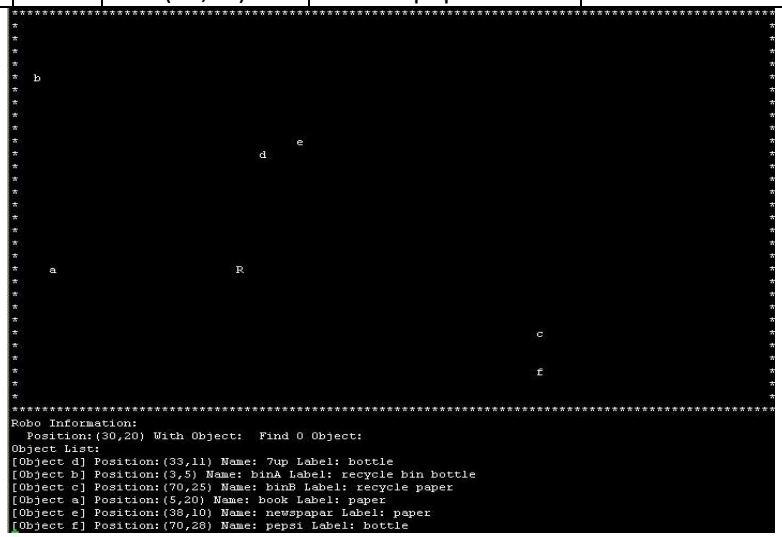
To verify the functionality of the TR framework, a simulator called Robo is built on ROS to simulate a robot in the real world. Robo has four capabilities: Navigation (go to the specified coordinate in the world); Grasping the object (as long as Robo is at the specified object); Placing/dropping the object; Object recognition (find the characteristics of the object). Furthermore, one sensor is used as well: Radar (find the position of unknown objects remotely and check whether the Robo is at some object or not).

With the capabilities designed above, the Robo is asked to do the recycle task with the control from one TR program. Shown in the figure below, in the virtual world, there are six objects, some of which are bins, while others are garbage items. Each bin has a label (paper or bottle), so does the garbage item. The Robo need to find all the garbage items and bins, and most important, put each garbage item into the right bin.

Initially, the world is also plotted in the figure below.

	ID	Position	Name	Label
Object a	1	(5,20)	book	Paper

Object b	2	(3,5)	binA	Recycle bin bottle
Object c	3	(70,25)	binB	Recycle paper
Object d	4	(33,11)	7up	Bottle
Object e	5	(38,10)	Newspaper	Paper
Object f	6	(78,28)	pepsi	Bottle

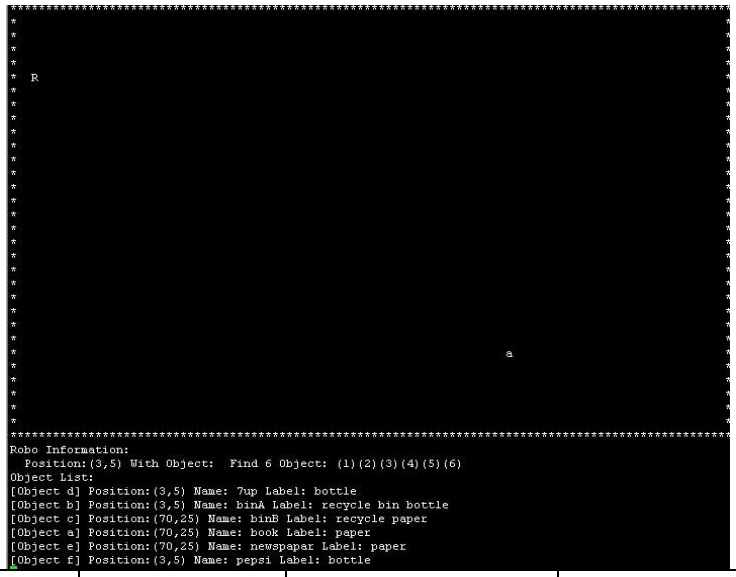


Under the control of TR progam, the action sequence of Robo is:

[NAVI] I go to (29,20)
... ...
[NAVI] I go to (5,20)
[NAVI] Destination (5,20) Arrived!
[Find] I found 1 objects so far, they are (1)
[Robo] I picked up book at (5,20)
[NAVI] I go to (4,19)
... ...
[NAVI] I go to (3,5)
[NAVI] Destination (3,5) Arrived!
[Find] I found 2 objects so far, they are (1)(2)
[NAVI] I go to (4,6)
... ...
[NAVI] I go to (70,25)
[NAVI] Destination (70,25) Arrived!
[Find] I found 3 objects so far, they are (1)(2)(3)
[Goto] I go to binB[Robo] I dropped book at (70,25)
[NAVI] I go to (69,24)
... ...
[NAVI] I go to (33,11)
[NAVI] Destination (33,11) Arrived!
[Find] I found 4 objects so far, they are (1)(2)(3)(4)
[Robo] I picked up 7up at (33,11)
[NAVI] I go to (32,10)
[Goto] I go to binA[NAVI] I go to (31,9)
... ...
[Goto] I go to binA[NAVI] Destination (3,5) Arrived!
[Find] I found 4 objects so far, they are (1)(2)(3)(4)
[Goto] I go to binA[Robo] I dropped 7up at (3,5)

[NAVI] I go to (4,6)
... ...
[NAVI] I go to (38,10)
[NAVI] Destination (38,10) Arrived!
[Find] I found 5 objects so far, they are (1)(2)(3)(4)(5)
[Robo] I picked up newspapar at (38,10)
[NAVI] I go to (39,11)
[Goto] I go to binB[NAVI] I go to (40,12)
... ...
[Goto] I go to binB[NAVI] I go to (70,25)
[Goto] I go to binB[NAVI] Destination (70,25) Arrived!
[Find] I found 5 objects so far, they are (1)(2)(3)(4)(5)
[Goto] I go to binB[Robo] I dropped newspapar at (70,25)
[NAVI] I go to (70,26)
... ...
[NAVI] I go to (70,28)
[Find] I found 6 objects so far, they are (1)(2)(3)(4)(5)(6)
[Robo] I picked up pepsi at (70,28)
[Goto] I go to binA[NAVI] I go to (70,28)
... ...
[Goto] I go to binA[NAVI] I go to (3,5)
[Goto] I go to binA[NAVI] Destination (3,5) Arrived!
[Find] I found 6 objects so far, they are (1)(2)(3)(4)(5)(6)
[Goto] I go to binA[NAVI] Destination (3,5) Arrived!
[Goto] I go to binA[Robo] I dropped pepsi at (3,5)
Congratulations. Succeed!!

When the task is accomplished, the world looks like



	ID	Position	Name	Label
Object a	1	(70,25)	book	Paper
Object b	2	(3,5)	binA	Recycle bin bottle
Object c	3	(70,25)	binB	Recycle paper
Object d	4	(3,5)	7up	Bottle
Object e	5	(70,25)	Newspaper	Paper
Object f	6	(3,5)	pepsi	Bottle

Conclusion

Agent planning is needed for robot to make high level decision. In this project, I implemented the TR framework based on the TR algorithm proposed by Nilsson. With the demonstration of recycle task, it is shown the TR framework make it easy to synthesize single actions into a complex task. In future, I will continue to implement more applications with TR framework. Also, some learning algorithms will be explored to assist user to better program in TR.

Acknowledgement

Thank Adam Coates for discussing the problem with me and providing a variety of useful suggestions. Also, thank Prof. Andrew Ng for teaching the learning algorithms. Though no machine learning algorithm is directly used in my project. I do benefit from the methodology part of how to applying ML.

Reference

[1] **Teleo-Reactive Programs for Agent Control**, Nilsson, N. *Journal of Artificial Intelligence research*, 1:139-158, 1994.

[2] **ROS Wiki :** http://pr.willowgarage.com/wiki/ROS