Complex Cobot

Zih-Min Wang
Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan(R.O.C.)
b05902062@ntu.edu.tw

You-Yang Hu
Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan(R.O.C.)
huyyoo@outlook.com

Chun-Yang Wang
Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan(R.O.C.)
wangchunyanghit@gmail.com

Chun Hui Hung
Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan(R.O.C.)
asd768999@gmail.com

***Abstract*—Collaboration between humans and machines nowadays is mostly limited to a predefined part of a workflow. In such a system, a user is forced to adjust his behavior to accommodate the system. And as a consumer, people sometimes wouldn't want to do that because that would require some extra work that would not trouble us when we are collaborating with a human partner. In this paper, a framework of physical Human-Robot interaction (pHRI) that goes beyond this limitation is proposed. This framework allows a robot to behave as a human would expect from a human partner, thus no more extra work and more user friendly. We are sure this concept would be incorporated in more and more products in the future.**

Keywords—physical Human-Robot interaction (pHRI), directed acyclic graph (DAG), depth first search (DFS).

I. Introduction

Nowadays, the use of robotic arms and humanoid robots has mostly been limited to organizations and industries. There are very few cases that make their entrances into our houses like a robotic vacuum. Apart from their expensive price tags, our team thinks the reason also lies in the purposes those robots serve. Many of the robots in the market has been designed to do one specific job only, for example preparing a meal all by itself. And even the ones that were labeled as cooperative robots can do only one fixed part of a job, either.

When a person needs help, there are cases he would want his partner to do the entire job. This is what the above-mentioned robots are designed for. But, for most of the time, the person might expect his partner to cooperate with him. An example for cooking a dish of tomato egg is shown in Fig. 1. The arrows in the figure represent the dependencies between jobs. The person may wish to do job 0 first, and his partner would possibly choose job 1. Then, the person may or may not do job 2. In either case, his partner would have to be able to counteract.

In this paper, we would propose a robotic system that behaves and cooperates in a way that a regular person would expect from a human, that is to check whether there is an available job in real-time and if so do it. Thus, we create a system that will not be restricted to only a part of a job but a system that would cooperate throughout a job. Besides, the system can be theoretically applied to almost any scenario after some definitions are modified and an extra program is provided. Section II introduce an example scenario that we would use through out this paper. Section III talks about the framework of the system and provide some algorithms we designed. Section IV described the result, limitation, and applications of our system. Concluding remarks are made in the final section.

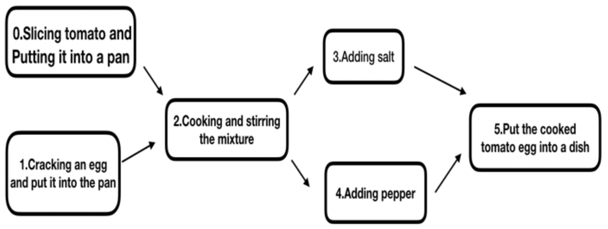
0.Slicing tomato and Putting it into a pan
1.Cracking an egg and put it into the pan
2.Cooking and stirring the mixture
3.Adding salt
4.Adding pepper
5.Put the cooked tomato egg into a dish

Fig. 1. An example for cooking a dish of tomato egg.

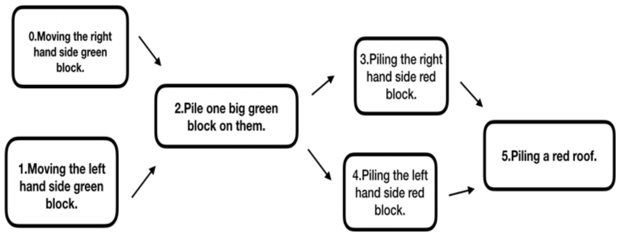
0.Moving the right hand side green block.
1.Moving the left hand side green block.
2.Pile one big green block on them.
3.Piling the right hand side red block.
4.Piling the left hand side red block.
5.Piling a red roof.

Fig. 2. An example scenario we would use in our paper.

II. A Scenario We Use To Model The Process Of Cooking A Dish of Tomato Egg

We have chosen a simple scenario of piling blocks to model the process of cooking a dish of tomato egg, as is shown in Fig. 2. One can see that the dependencies in Fig. 2 are exactly the same as the ones in Fig. 1 and see the tasks, which are numbered from 0 to 5, as one to one mapping from cooking to piling. Dependencies are shown in the concept of piling. The upper blocks cannot be piled before the lower blocks are put into places. Fig. 3 and Fig. 4 together show the block piling example that we will use and eventually implement in this paper.

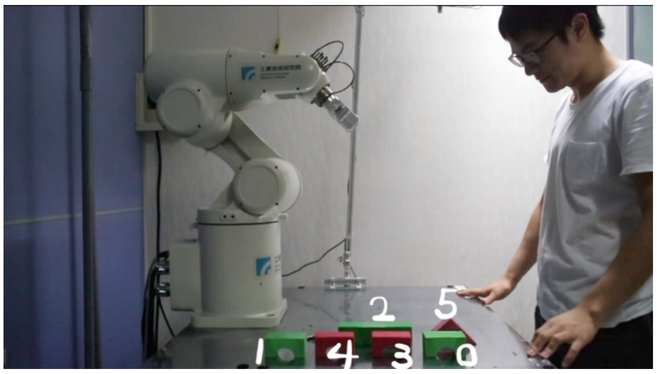

Fig. 3. Blocks in their original positions.

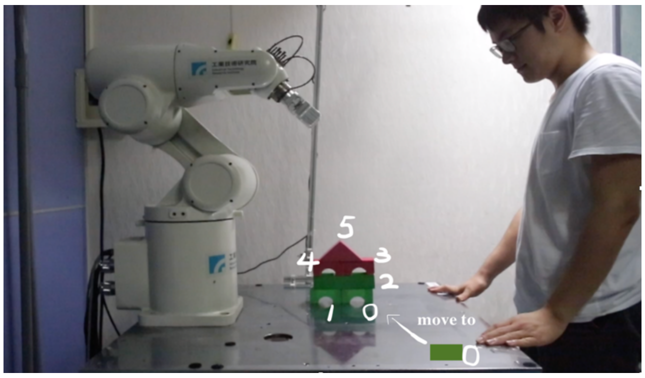

Fig. 4. Blocks in their final positions.

III. FRAMEWORK AND MYTHOLOGY

A. Framework Overview

There are two programs in our cooperative system, as is shown in Fig. 5. A state generating program would continuously generate a file containing states of each individual job in Fig. 2. A robotic arm program would take a look at the state file when the robotic arm is available and then determine what command to send to the robotic arm, if any.

B. Definition of Jobs

A directed acyclic graph (DAG) is chosen to represent the structure of jobs. Nodes are used to represent jobs. Directed links are used to represent dependencies. The time complexity analyses below are performed under the assumption that the DAG is implemented using linked lists.

C. State File

To enable a robot to decide what to do autonomously, we need to let it know the states of all jobs first. We think any job can be classified into three types of states: not yet started, started but yet finished, and finished. A state file containing states of all jobs will be accessible by the robotic arm program. Algorithms, with the state file and the DAG as its input, that would enable the robot to choose a proper job are provided below. Plus, because of the dependencies between jobs, some combinations of states of jobs are actually invalid.

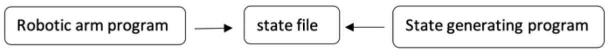

Fig. 5. The framework of Complex Cobot.

D. Modified Depth First Search (DFS) Algorithms and Some Preprocessing

Because we only allow our definition of jobs to be a DAG, firstly, we shall check whether it is indeed a DAG. To check whether a graph is a DAG, we only need to check whether it yields any back-edge. This can be achieved simply by running a regular DFS algorithm [1].

Secondly, though we can reasonably assume that the states in the state file are valid, this relies on how we write the state generating program, it is worth it to check it again. This would check whether the combination of states of jobs is invalid as described earlier. Given the original graph (V, E), that is the definition of jobs, and the state file. For all n ∈ V, we use n.set to represent the set to which node n belongs, and n.state to represent the state of node n. We devise an algorithm with DFS in it and can perform this job in O(V+E).

```
Check_invalid ():
    Iterate n over V
        Initialize n.set to V
        Initialize n.state according to the state file
    Iterate n over V
        If n.state is equal to finished then do nothing
        else
            set n.set to T
            DFS(n)
    return

DFS (n):
    Iterate over directed link starting from n to n'
        if n'.set is equal to D then continue
        if n'.state is not equal to not yet started then error
        else
            set n'.state to D
            DFS(n')
    return
```

Because there are dependencies between jobs, some jobs cannot be chosen until the jobs on which they depend are all finished. Thus, it is necessary to do some preprocessing before the robot makes its final decision. An algorithm to rule out those jobs is provided below. This Rule_out() algorithm also would finish in O(V+E).

```
Rule_out ():
    Iterate n over V
        Initialize n.set to V
        Initialize n.state according to the state file
    iterate n over V
        if n.state is not yet started then do nothing
        else if n.state is finished then set n.set to U
        else if n.state is started then
            set n.set to U
            DFS(n)
    return;

DFS (n):
    Iterate over directed link starting from n to n’
        If n’.set is equal to V
            set variable d to more
            set n’.set to U
            DFS(n’)
    return
```

The robot is now finally ready to choose a proper job. It would simply run a topology sort algorithm, we use the regular DFS [1] again, on the nodes in V whose n.set is equal to V after running the above Rule_out(). The above variable d can tell us whether there are more jobs, which we shall choose one or in some cases just wait, or there is no more job and the system should terminate. Since these four parts of the robotic arm program all finish in O(V+E), the overall time complexity for the robotic arm program is also O(V+E).

E. state file program

We need to have a state generating program to generate the state file. Our cooperation system depends hugely on the correctness of the state file so this program should be as accurate as possible. The design of this program depends on where the cooperation system will be used. For example, if we want to use the system to prepare a dish of tomato egg as described previously. With those jobs in mind, the program can be written to produce a state file as closely as possible. Using machine learning (ML) to train a model that generates state files for general cases is possible and may be considered as our future work.

IV. Result

This framework is meant to fit into all cooperation scenarios. A system for our example scenario has been implemented. A video of our life demo is at https://youtu.be/3nmcs4G_KLw. And it yields very positive results under certain limitations. These limitations include at least the following.

- Though the state generating program is almost always accurate in our scenario, it doesn’t make sense to assume it would always be correct because, in reality, the environment is extremely complex and the system may be applied to other scenarios. So, the system should actually include some mechanisms to allow it to address the problems that is related to this sort of error.
- Even if the state generating program is always correct, there can be situations where a robot and its human partners contest for an available job. In this case, sometimes we may hope the robot would choose another job instead.
- Interactions between robots and humans can be dangerous. So, if they do share a workspace, it is necessary to introduce some algorithms to avoid collisions.

Though we have only implemented a special case, what we try to do is to demonstrate the concept of cooperation. And the positive results do prove some of the concept in our system may be valuable.

V. Conclusion

Cooperation between humans is complex, so building a robot that cooperates with us in a way that we would expect from a human is difficult. Despite the hardness, we still propose this framework because we think this complex cooperative robot is the type of robot that is most likely to appear in our houses in the future. Our model is not complete, but it can be a fundamental structure of a more complex system. We hope more research can be done about it, and one day we will be able to write papers with electronic partners.

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, “Introduction to Algorithms,” The MIT Press, Cambridge, Massachusetts London, England, pp. 550–551, 2001.