Robust Human-Computer Interaction System Guiding a User by Providing Feedback

M. S. Ryoo and J. K. Aggarwal
Computer and Vision Research Center
Department of Electrical and Computer Engineering
The University of Texas at Austin, Austin, TX 78712
{mryoo, aggarwaljk}@mail.utexas.edu

Abstract

We introduce a human-computer interaction system which collaborates with a user by providing feedback during user activities. The goal of the system is to help a user complete a high-level activity that has been represented hierarchically. While a user is performing the high-level activity, our system analyzes what sub-events a user has already completed and what sub-events are needed next in order for the user to finish the activity. The representations of human activities are constructed using a previously developed context-free grammar based representation scheme. We focus on a game named 'pentagram game' to illustrate our system. In the experiments, our system shows the ability to guide the user to complete the 'pentagram game' by providing explicit feedback. The feedback not only contains atomic level instructions but also describes higher-level long-term goals of the composite activities.

1 Introduction

Humans understand the structure of activities, and they take advantage of this knowledge of structure when performing high-level activities. In the middle of performing a complex activity, humans know exactly what sub-events have already occurred and what sub-events need to be performed next. They explicitly or implicitly know what to do next, considering what they have done in order to achieve the goal in the past. For example, during a dance of an expert dancer, the dancer explicitly or implicitly knows what his/her next move should be. The dancer knows all his/her moves and their order in the dancing activity, and uses this knowledge during the dancing activity to decide various moves.

Motivated by the ability of human experts, we design a human-computer interaction (HCI) system which collaborates with a user to help the user complete high-level human activities. Similar to human experts, the system has knowledge on the structure of the activity that the user wants to perform. Our system has a representation of an activity which describes how the correct activity should look like temporally, spatially, and logically, and uses it to automatically guide the user to perform the represented activity. While the user is trying to perform the activity he/she wants to do, our HCI system makes its own analysis on the current status of the activity being performed, and produce feedback. If the user does not perform the correct sub-event, the computer interface alerts the user to the mistake. The goal is to enable the system to provide feedback (or advice) to the user based on its action representation and visual observations.

A HCI system helping a novice in the task of tightening nuts while installing a car tire is a typical example that motivated the construction of our algorithm. The task is composed of multiple sub-events (tightening a single nut) which must follow a particular order: nuts must be tightened in the order that is similar to drawing a star. Also, in this type of tasks, a user can undo sub-events (un-tightening a nut) to recover from his/her mistake. The detailed structure and characteristic of this activity is discussed in the section 4. Our system has the ability to guide a user not only to perform sub-events in a correct order but also to undo sub-events when he/she made a mistake. Our algorithm is designed to provide feedback to accomplish those activities as well as other general activities.

Previous HCI systems had mainly action-reaction based models. Those systems recognize human actions and perform particular reactions already encoded in the system. Research on intelligent environments is a typical example of these reactive systems. The goal of our system is a little different. The goal of previous reaction based HCI systems is to provide correct reactions for corresponding actions, while that of our system is to guide and instruct how to make a whole high-level activity successful by providing feedback. Reaction based HCI systems are not suitable in our case. In order for reaction based HCI systems to guide a user to perform represented activities containing concurrent as well as sequential sub-events, the system must specify all possible steps and branches of the order of sub-events; the number of cases becomes very large. In addition, the feedback provided by our system is hierarchical, which is difficult for reaction based systems to deal.

The main technical contribution made in this paper is on the algorithm to provide feedback, which estimates the internal states of the incomplete composite activity and predicts what sub-event is needed in order to complete the action.

In the case of simple sequential models, such as finite state machines or hidden Markov models, estimating which state and what to do is a simple task since they inherently contain the information on the next occurring sub-event. However, in the case of activities that can only be represented by highly complex models, such as full first-order logic, this is no longer a simple task. We use the human activity representation framework developed by Ryoo and Aggarwal [2006]. In activities consisting of concurrent sub-events, it is not easy to determine which point the system has reached, and what sub-event is urgently needed in concurrent activities. In addition, in highly complex activities, some sub-events might have an identical sequence in the beginning, confusing the system in its decision as to the status of the activity.

2 Related Works

The reactive systems mentioned in the introduction have similar aspects as our system. Since Coen [1998] presented design principles of intelligent environments, several research projects constructed HCI systems for intelligent environments, rooms, and workspaces [Hassens et al., 2002; Kulkarni, 2002]. They introduced the concept of reactive behavioral systems, extending previous rule-based reactive systems. Their main goal is to make the HCI correctly respond to relatively simple user behaviors or actions within the activity context.

On the other hand, several works on surveillance and children monitoring systems attempted to recognize hierarchical human activities [Minnen et al., 2003; Park and Aggarwal, 2006; Siskind, 2001]. Most of those works only focused on after-the-fact detection of activities, and did not attempt to provide feedback to the user who is performing the activities. In the work done by Ryoo and Aggarwal [2006], high-level human activities were represented hierarchically using context-free grammar (CFG) syntax, specifying temporal, spatial, and logical relationships among sub-events. Their representation was similar to the first-order logic using Allen's temporal predicates [Allen, 1994], and recognition was done hierarchically.

3 Feedback

The objective of our algorithm to provide feedback is as follows: Given a situation where the user is in the middle of performing an action, the system must analyze which sub-events are already done and which sub-events need to be done. The explicit feedback must be provided finally, stating which sub-events are immediately needed and what is their appropriate starting time and ending time.

We assume that the system initially has the language-like representation of activities. We start with converting the representation of activities into a directed graph representation, which makes the system able to apply our algorithm. We then discuss principle theorems justifying our approach. Finally, we present a general algorithm for providing feedback to hierarchical activities. The system must convey hierarchical structure of activities and generate hierarchical feedback.

3.1 Language-like Representations

In order for our HCI system to provide feedback, the system must have a formal and complete representation of the activity that a user is trying to accomplish. We use Ryoo and Aggarwal's CFG-based representation syntax to construct a formal representation of activity by listing necessary temporal and spatial conditions. It is always assumed that the system has a programming language-like representation of the activity it wishes to recognize.

Example 1 Let's look into a spinning move in figure skating as an example. Spinning moves that figure skaters do are complex human activities, which involve both arm and leg movements. Assume that our system wants to provide feedback (advice) to a skater to perform a common combination spin, a *camel-sit* spin. In a *camel* spin, a skater must spin with his/her leg and arm fully outstretched horizontally, into the general shape of a 'T'. In a *sit* spin, the skater must sit while spinning. The *camel-sit* spin is a sequence of two spins: the skater must initially spin with a 'T' shape, and then sit while spinning. Our system must have the following CFG-based representation before applying our feedback providing algorithm:

```
Camel-Sit_Spin(person1) = (
  list( def('a', Spin(person1)),
        list( def('b', Sit(person1)),
              list( def('c', Stretch_Arm(person1)),
                    def('d', Stretch_Leg(person1))) )),
  and( equals(a, this),
       and( and( during(c, a), during(d, a) ),
            and( during(b, a),
                 and( meets(c, b), meets(d, b) ) ) ))  );
```

3.2 Directed Graph Representations

The language-like representation of activities is human-oriented. In order to estimate ongoing status of the activity and to generate proper feedback, our system converts a programming language-like activity representation into an internal directed graph representation automatically using the following algorithm.

In our directed graph representation, a vertex is a time point (either starting time or ending time of a sub-event), and an edge from vertex *t1* to vertex *t2* implies $t1 < t2$. The purpose of this conversion is to calculate the necessary temporal ordering between times associated with an activity's sub-events. Our feedback providing algorithm uses this directed graph representation of human activities, since the directed graph representation enables us to calculate the order of two time points easily.

The procedure to convert our CFG-based representation into a directed graph representation is presented below.

First, the system must convert Allen's temporal predicates for time intervals into equalities and inequalities among time points. In our CFG-based representation, temporal relationships are specified as a logical formula of Allen's temporal predicates. Following the definition of temporal predicates, the representation can be converted into equalities and inequalities among time points as follows:

Let *a* and *b* be the time intervals, (a_{start}, a_{end}) and (b_{start}, b_{end}).

equals(a, b) => $a_{start} = b_{start}$ *and* $a_{end} = b_{end}$
before(a, b) => $a_{end} < b_{start}$
meets(a, b) => $a_{end} = b_{start}$
overlaps(a, b) => $a_{start} < b_{start}$ *and* $b_{start} < a_{end}$
starts(a, b) => $a_{start} = b_{start}$ *and* $a_{end} < b_{end}$
during(a, b) => $b_{start} < a_{start}$ *and* $a_{end} < b_{end}$
finishes(a, b) => $a_{end} = b_{end}$ *and* $a_{start} > b_{start}$

Also, we add one trivial inequality $a_{start} < a_{end}$ for all time intervals *a*. As a result, logical concatenations of Allen's temporal predicates are converted into logical concatenations of equalities and inequalities among time points.

Next, the system removes the predicate *not*, as follows.

not(t1<t2) => *t2 < t1 or t1 = t2*
not(t1=t2) => *t1 < t2 or t2 < t1*

The system then converts a logical formula into a disjunctive normal form (DNF). The end product is the disjunction of conjunctive clauses of pure equalities and inequalities. This suggests that the activity representation can be divided into several conjunctive clauses, where each clause presents necessary temporal conditions for the activity. There is no semantic difference between the DNF representation and the directed graph representation we plan to construct. For each clause, we formulate one directed graph to help the user visualize the representation. The system first calculates time points that are equal, checking the equalities in a clause. The system assigns one vertex for a set of time points which are equal. For example, if *t1=t2 and t2=t3*, only one vertex is assigned for a set *{t1, t2, t3}*. Then, an edge is constructed from a vertex v1 to a vertex v2, if *∃t1, t2 such that (t1 ∈v1 and t2 ∈v2)*. What this directed graph representation suggests is that the activity can be completed if and only if an integer value is correctly assigned for each vertex while satisfying the temporal order of the graph.

Example 2 The language-like representation of the *camel-sit spin*, which we discussed in example 1, may be converted into a directed graph. First, the temporal predicates are converted into equalities and inequalities.

$(a_{start} < a_{end}$ *and* $b_{start} < b_{end}$ *and* $c_{start} < c_{end}$ *and* $d_{start} < d_{end}$ *and* $a_{start} = this_{start}$ *and* $a_{end} = this_{end}$ *and* $a_{start} < c_{start}$ *and* $c_{end} < a_{end}$ *and* $a_{start} < d_{start}$ *and* $d_{end} < a_{end}$ *and* $a_{start} < b_{start}$ *and* $b_{end} < a_{end}$ *and* $c_{end} = b_{start}$ *and* $d_{end} = b_{start})$

This already is a DNF composed of only one conjunctive clause. Figure 1 shows a final directed graph representation of the example.

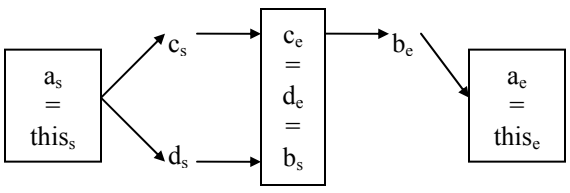


Figure 1: The directed graph representation for 'Camel-Sit_Spin'. Edges (a_s, a_e), (a_s, b_s), (c_e, a_e), and (d_e, a_e) are omitted in the graph.

3.3 Consistent State and Feedback

This subsection describes principle definitions and theorems which justifies for our algorithm theoretically.

In our directed graph representation, analyzing the status of ongoing activity is identical to finding up to which vertices the user has assigned a correct value. Since vertices correspond to the starting or ending times of sub-events, assigning values for vertices means that a user started and ended sub-events in those time points. We denote a set of assigned vertices as a *state* of the graph.

The system can calculate more than one possible assignment, i.e. state, in one situation. There might be multiple candidate assignments for each vertex, since sub-events can occur multiple times. In addition, the system might leave some vertices unassigned, leaving them to be assigned in the future. Among many possible states, few of them satisfy the temporal order specified in the graph. We call those states *consistent states*.

Definition 1 (Assigned Vertex) *We define a vertex* assigned vertex*, if the system has assigned an identical value to all time points associated with the vertex. We define a vertex* unassigned vertex, *if the system did not assign any value to the time points associated with the vertex. We define a vertex* half-assigned vertex*, if the sub-set of time points in the vertex are assigned while the others are unassigned.*

Definition 2 (Consistent State) *Let S be a set of assigned vertices in a graph G. Let H be a set of half-assigned vertices in G. We say the graph G is in the* consistent state *with the assignment S and H if and only if*

1. *∀ v, w: v, w ∈ S ∪H, v<w if there exist a path from v to w, and*
2. *∀v: v ∈ S, (parents of v) ⊂ S, and*
3. *∀v: v ∈ H, (parents of v) ⊂ S, and*

The consistent state implies that there is no contradiction among assigned values of the graph. The consistent state clearly shows up to which point the activity has been processed. From Definition 2, we know that a graph without any assignment is also in a consistent state. Among consistent states, the states which we are particularly interested in are *maximum consistent states*. If additional assignments based on currently provided time intervals are not possible for a consistent state, it is a maximum consistent state.

Calculating the maximum consistent states of an ongoing activity is the key component not only for analyzing the status of ongoing activity, but also for providing feedback. We introduce two theorems which will lead us to use an iterative approach to find maximum consistent states.

Theorem 1 *Let U be a set of all vertices in a graph G in a consistent state. Let v be a vertex, where v ∈U-H-S and ∀ (parents of v) ∈ S. If we assign a value larger than (max of parents of v) to the entire time points in v, then G is still in the consistent state. If we assign the same value only to the sub-set of time points in v, making the v half-assigned, then G is still in the consistent state.*

Theorem 2 *Let v be a vertex, where v ∈H. If we make vertex v an assigned vertex by assigning identical values for all time points in v, then G is still in the consistent state.*

Iteratively updating consistent states using Theorem 1 and 2 will lead us to the maximum consistent states. Assuming that the system knows all starting times and ending times of sub-events, the system can check whether further update is possible or not. If possible, the system can assign the value of detected sub-events for corresponding vertex. The only constraint is that the starting time and the ending time of the same sub-event must match. For example, if there are two time intervals (1, 5) and (3, 8) for the sub-event A, the system cannot assign 1 for A_s and 8 for A_e. This can be checked by applying either a dynamic programming algorithm or an algorithm which traverses back when adding the ending time of a sub-event. The resulting time complexity is either linear time or quadratic time in each case.

Algorithm to calculate maximum consistent states

```
1  Initialize all vertices in the graph as unassigned vertices
2  do
3      Apply Theorem 1 and 2 to find a vertex v
4      Make v an assigned vertex or a half assigned vertex
       accordingly
5      Update v.value as a list of possible assignments
6      for all i,
7          Update v.value[i].p as a list of assignments of
           parent vertex which is less than v.value[i].
8          if v.value[i] is an ending time of a sub-event,
9              Traverse back the graph using value and
               value[i].p of ancestors of v to check the
               starting of the sub-event was assigned
10             if not, remove v.value[i] from v.value
11 while v ≠ NIL
12 return assignments of the graph
```

Even though maximum consistent states do not have any contradiction among their assignments on vertices, it does not imply that the future assignments without any contraction are possible. Therefore, we must calculate *maximum valid consistent states* based on the maximum consistent states.

Definition 3 (Valid Consistent State) *We call a consistent state as a* valid consistent state*, if it has at least one possible combination of future assignments that satisfies temporal relationships among vertices. The* maximum valid consistent states *are the valid consistent states which do not have other valid consistent states containing them.*

Algorithm to calculate maximum valid consistent states

```
1  for each maximum consistent state with the assignment
   S and H,
2      S' = S ; H' = H
3      do
4          Find a_s∈v such that
               {v∈S' ∧ ((a_e is not assigned ∧ a_e<= current
               time point) ∨ (a_e is assigned for v'∈H' ∧
               a_e<current time point))} ∨
               {v∈H' ∧ a_s<current time point}
5          S' = S' - (a_s ∪ descendent of v)
6          H' = H' - (a_s ∪ descendent of v)
7      while a_s ≠ NIL
8  return set of all S' and H'
```

After calculating maximum valid consistent states, the system can provide feedback to the user. The feedback is one-step look-ahead information of the activity. By applying Theorem 1 and 2 once more to calculated maximum consistent states, the system is able to see what sub-events are needed next in what time interval.

3.4 Hierarchical Feedback Algorithm

The actual algorithm for feedback providing is described in this subsection, especially focusing on the hierarchical aspect of our feedback.

Most high-level human activities have hierarchical structures; sub-events also have their own sub-events. This implies the system needs a hierarchical algorithm to recognize time intervals of sub-events, and suggests that hierarchical feedback is needed.

Therefore, a recursive algorithm is designed, where the base case is the feedback providing for atomic actions (directly telling the user to do the atomic action). For composite activities, the system analyzes sub-events' temporal structures using our directed graph representation of the activity, and tells the user to start or end a particular sub-event, by applying the algorithm presented in subsection 3.3. If that sub-event is an atomic action, then the system can simply tell the user to do the action. If that sub-event itself is a composite activity, the system must tell the user how to complete that sub-event also. That is, the system must examine sub-events of the sub-event to provide hierarchical feedback. This process continues until the atomicity is gained.

Example 3 Let's look into the spinning moves in figure skating once more. Assume that a user wants to perform the *camel-sit/sit* spin. We already defined what the *camel-sit* spin is and what the *sit* spin is. The *camel-sit/sit* spin is a combination of the *camel-sit* spin and the *sit* spin, where the skater changes his/her axis feet between two spins. We can set the sub-events of the *camel-sit/sit* spin to be the *camel-sit* spin and the *sit* spin. Assume that the system wants to provide feedback when the user just started to spin at time 1. Applying the approach presented in subsection 3.3 will result with the system telling the user to 'start the *camel-sit* spin, after 1+'. However, that information is not sufficient. The system must explicitly specify how to start the sub-event also, as follows:

1. In order to do a *camel-sit/sit* spin at (1, 1+), do the *camel-sit* spin at (1, 1+)
2. In order to do a *camel-sit* spin (1, 1+), do a *stretch_arm* at (1+, 1+) and do a *stretch_leg* at (1+, 1+).
3. *Stretch_arm* and *stretch_leg* are atomic actions, so do them now.

We always assume that the system knows time intervals of sub-events already occurred by using the recognition algorithm for the CFG-based representations. Further, we also assume that the system knows the starting time of a sub-event as soon as the sub-event started. That is, for each sub-event, the system has a list of time intervals which specifies previous occurrences of the sub-event, and has candidate starting times of the sub-event that are not completed.

Algorithm to provide hierarchical feedback:

1 Using previous recognition algorithms for the CFG-based representation, find all possible starting times and ending times of sub-events
2 Convert the CFG-based representation into directed graph representations
3 Find the maximum valid consistent state closest to goal
4 Apply Theorem 1 and 2 once more to construct feedback. When to start or end the corresponding sub-event is specified
5 If the sub-event from step 4 is a composite itself, apply the same procedure from step 1 to the sub-event
6 **return** the concatenated feedback

4 Experiments

We focus on the activity named 'pentagram game' for the experiments. The 'pentagram game' is a composite action, a sequence of moving or removing stones in a particular place, similar to the activity of tightening the tire of a car. For our experiments, we define three types of atomic actions that take two parameters: *move_stone(person1,place1), remove_stone (person1, place1),* and *basic_safe(person1, place1)*. The first atomic action describes a person1 moving a stone to place1. The seconding atomic action describes a person1 removing a stone from place1 and placing it to somewhere else. The last atomic action describes that a person1 did not perform any action related to place1. For the notational convenience, we will simply write *move_stone(person1, place#)* as '*move#*', since only one user is moving the stones. Our system detects the starting time and ending time of the atomic actions using the computer vision techniques.

In order to make the high-level activity 'pentagram game' successful, the user must place all stones in the locations following a particular order. Figure 2 describes the order of the activity 'pentagram game'. One important characteristic of the 'pentagram' is that the user can always remove stones to go back to previous the state, which makes the activity more complex than it seems to be. For example, following the sequence of atomic actions satisfies the 'pentagram game' activity: *move1-move3-remove3-move2-move3-move 4-move5*. Even though *move3-remove3* was interrupted in the original sequence *m1-m2-m3-m4-m5*, this also is a 'pentagram game' activity since the person removed the stone from *place3* before other moves. Before defining the composite activity 'pentagram game', we need to define *move* followed by *remove*, such as *m4-m5-rm4-rm5*. We name this activity as a composite activity *Safe(person1, stone1, place1, place2, place3, place4, place5)*. When playing 'pentagram game', 'safe' activity can be inserted between correct moves.

Therefore, the 'pentagram game' will be composed of five 'move_stone' atomic actions, and five 'safe' composite activities: *Safe-m1-Safe-m2-Safe-m3-Safe-m4-Safe-m5*. Our language-like representation for the activity 'PentagramGame' is as follows. The actual representation of the 'Safe' is omitted due to the limitation of space.

PentagramGame(person1, place1, …, place5) = (
list(def('x1', move_stone(person1, place1)),

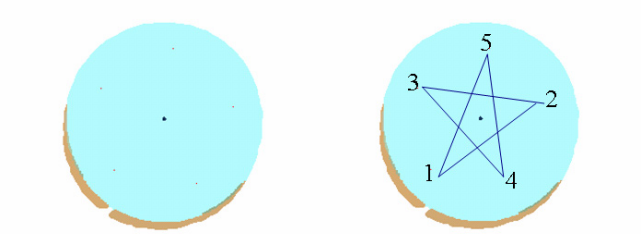


Figure 2: The illustration of the setup for 'pentagram game'. There are five locations: place1, place2, …, and place5. Black stones are to be placed on each location. If the placement is done in correct order from 1 to 5, then the activity 'pentagram game' is complete. Otherwise, if in an incorrect order, the activity is incomplete. Note that the user can freely remove placed stones when he/she thinks the placement is wrong.

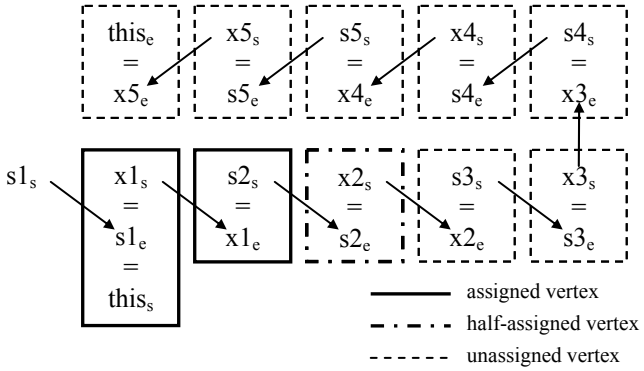


Figure 3: The directed graph representation for 'PentagramGame'. The dotted vertices are unassigned vertices or half-assigned vertices.

def('x2', move_stone(person1, place2)),
def('x3', move_stone(person1, place3)),
def('x4', move_stone(person1, place4)),
def('x5', move_stone(person1, place5)),
def('s1', Safe(person1, place1, …, place5)),
def('s2', Safe(person1, place1, …, place5)),
def('s3', Safe(person1, place1, …, place5)),
def('s4', Safe(person1, place1, …, place5)),
def('s5', Safe(person1, place1, …, place5))),
and(
starts('this', 'x1'), finishes('this', 'x5'),
meets('s1', 'x1'), meets('x1', 's2'), meets('s2', 'x2'),
meets('x2', 's3'), meets('s3', 'x3'),meets('x3', 's4'),
meets('s4','x4'), meets('x4', 's5'), meets('s5', 'x5'))
);

We shot a video of users doing the 'pentagram game'. Some of them performed a complete sequence of successful activities, while others were stopped in the middle of an activity. For some sequences, mistakes were intentionally made, such as inserting *move4-move3* after initial *move1*. The purpose of these erroneous insertions was to test whether our feedback providing algorithm can correctly guide the user to complete the overall activity. For example, if we fed the input sequences of *move1-move4-move3*, the system must guide user to start *remove3* and start *remove4*.

Videos taken by the Sony VX-2000 were converted to a sequence of frames with a frame rate of 10 per second. Each

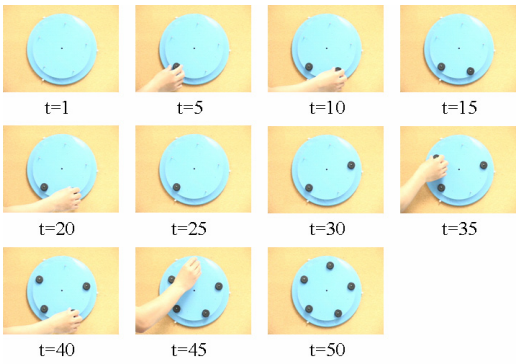
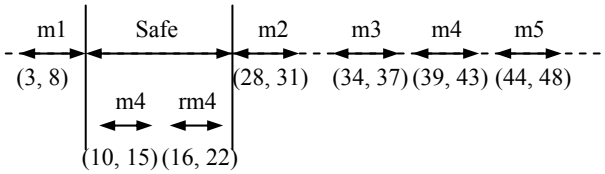


Figure 4: The raw sequence of input images. The 'pentagram game' activity is completed in the following order: *move1-safe(move4-remove4)-move2-move3-move4-move5.*.

Recognition results:



Feedback provided at time point 15:

```
Objective:
  PentagramGame (3, 15+)
  Safe (8, 15+)
    Safe4 (8, 15+)
      Undoing4 (10, 10+)
        Remove4 (15+, 15+)
  Therefore, do Remove(person1, place4)
  now.
```

Figure 5: The recognition results of the input shown in Figure 5. Output of feedback providing system at time point 15 is also provided. The system clearly states hierarchical feedback.

frame has a resolution of 320*240. From the raw visual input sequences, starting times and ending times of the atomic actions were recognized automatically. Recognized atomic actions were coupled with time intervals, making the recognition and feedback providing of composite activities possible. Because of the simple domain, the recognition rate for atomic actions *'move_stone'* and *'remove_stone'* were 1. The stone blobs were clearly detected and tracked. As a result, the system was able to correctly recognize the activity 'pentagram game' regardless of the number of atomic actions.

Based on the perfect recognition of atomic actions, our new feedback providing algorithm was tested. Figure 4 and 5 shows the feedback generated as the sequence of input frames were given. The feedback contains hierarchical information. The feedback starts from telling the objective time intervals of a high-level activity that the user wants to accomplish. Then, the system states what sub-event must be started or ended in order to accomplish the activity. For example, in Figure 5, the system informs the user in what time interval the composite activity 'pentagram game' must be done. Then, the system mentions the composite activity 'safe' is needed in time interval (8, t>15), in order to perform 'pentagram game' in the time interval (3, t>15). The 'undoing' activity is needed in time interval (10, t>10), and finally, the atomic action 'remove' is needed in time intervals (t1>15, t2>15) to complete 'undoing'.

5 Conclusions and Future Works

The contribution of this paper is the framework and the algorithm to provide feedback for ongoing activities. The feedback contains simple instructions to start or end one sub-event of the activity, and it also contains instructions on how to start or end the sub-event itself. Based on the previous CFG-based representation and recognition of the activity, the system estimates the state of activity correctly, and calculates what sub-event must happen next. This is a novel human-computer interaction system based on the computer vision. The ability to provide feedback for complicated activities represented in terms of time intervals distinguishes our work from other works on temporal planning using temporal networks. In future, we plan to test our system on more complex domains.

References

[Allen, 1994] J. F. Allen, Maintaining knowledge about temporal intervals, *Communications of the ACM*, 26(11):832-843, 1983.

[Bobick *et al.*, 2000] A. Bobick, S. Intille, J. Davis, F. Baird, C. Pinhanez, L. Campbell, Y. Ivanov, A. Schutte, and A. Wilson. The KidsRoom. *Communications of the ACM*, 43(3). 2000.

[Coen, 1998] M. Coen, Design principles of intelligent environments, *Proceedings of AAAI'98*, 1998.

[Hassens *et al.*, 2002] N. Hanssens, A. Kulkarni, R. Tuchinda, and T. Horton, Building Agent-Based Intelligent Workspaces. *Agents for Business Automation Conference Proceedings*.

[Kulkarni, 2002] A. Kulkarni, "Design Principles of a Reactive Behavioral System for the Intelligent Room," in *Bitstream: The MIT Journal of EECS Student Research*, 2002.

[Minnen *et al.*, 2003] D. Minnen, I. Essa, T. Starner, Expectation Grammars: Leveraging High-Level Expectations for Activity Recognition, *CVPR'03*.

[Park and Aggarwal, 2006] S. Park and J. K. Aggarwal, Simultaneous tracking of multiple body parts of interacting persons, *CVIU 102(1)*, pp. 1-21, April 2006.

[Ryoo and Aggarwal, 2006] M. S. Ryoo and J. K. Aggarwal, Recognition of Composite Human Activities through Context-Free Grammar based Representation, *CVPR*, 2006, pp. 1709-1719, New York, NY.

[Siskind, 2001] J. M. Siskind, Grounding the Lexical Semantics of Verbs in Visual Perception using Force Dynamics and Event Logic, *JAIR*, 15(2001) 31-90.