Application of Reinforcement Learning in Multi-Sensor Fusion Problems with Conflicting Control Objectives

Shichao Ou, Andrew Fagg, Prashant Shenoy
Department of Computer Science,
University of Massachusetts,
Amherst, MA - 01003, USA
chao,fagg,shenoy@cs.umass.edu

Keywords: Multi-sensor fusion, conflicting objectives, reinforcement learning, adaptive power management

Abstract

Robots or other smart agents are equipped with sensors that enable them to be sensitive to the surrounding, including human beings in the environment. However, the mapping of multiple raw streams of sensory data to the appropriate actions is not an easy problem, especially when multiple conflicting objectives are involved. This paper studies this type of multi-sensor fusion problem through the domain of power management of smart mobile devices. In such application domain, users would prefer their mobile devices to stay on as long as possible such that just-in-time services (such as reminder announcements) can be provided. However, this conflicts with the mobile device’s inherent goal - turn off to conserve power. These are two opposing objectives. Due to the stochastic nature of human behavior, a hand-coded fixed strategy may not be the best solution. We present a learning control approach to the problem. Through experiments, we show that our approach learns the appropriate mapping between multiple streams of raw sensory data to power management actions and produces policies that can outperform the hand-crafted policies. The learned policies are also shown to be more robust in handling unscheduled events.

1. Introduction

When smart agents, physical (robots) or virtual (software agents) work to assist humans, they need sensors to help them detect changes in the environment, in order to be useful. However, the mapping of multiple raw streams of sensory data to the appropriate actions is a difficult task, especially when multiple conflicting objectives are involved. This paper studies this type of multi-sensor fusion problem through the domain of smart mobile devices.

Smart assistant digital agents running on a PDA, a wearable computer or a capability-rich cell phone provide digital services to an user that are based on the current activity that the user is engaged in [1, 13]. Since such devices are always with the user, the smart assistant can be useful in tasks such as reminding user relevant upcoming events [15], autonomously handling incoming phone calls under different context [2, 8], or precaching web pages [16], emails of the user’s interest when it anticipates the user will be off the network soon. These proactive assistive actions often occur in the background when the user is not physically interacting with the device. Therefore, the device must stay ON (even though the user is not actively using it) in order to collect information and assist the user. Ideally, it should be on all the time such that no context change is missed. However, staying on all the time is not possible for any capability-rich mobile device, with the current battery technology. Therefore, for the device,
there exists two conflicting objectives: to stay ON for as long as possible to assist the user, and to take power conservation actions without causing annoyance to the user or missing important events. Due to the stochastic nature of human behavior, a hand-coded fixed strategy may not be the best solution.

The approach taken by this work is a learning control algorithm, Reinforcement Learning (RL) [21]. With various physical sensors (e.g., location sensors [5, 3, 10, 12]) as well as data streams from software applications (e.g., calendar events, email notifications, application usage patterns), states of the world (from the user’s perspective) can be captured. Any control actions taken by the system has consequences, some positive, some negative. They are referred to in this paper as *costs*. By associating control actions with *costs*, and designing an appropriate cost model for the application, the RL system can autonomously learn the mapping between multiple sensory streams and the correct sequence of system actions such that total cost can be minimized, balancing between the two opposing objectives.

2. A Learning Control Approach

To address the issue of mapping sensor data to the appropriate action, we turn to the feedback control domain. In the form a control problem, as shown in figure 1, we treat the actions that the system takes as control actions that change the state of the world and of the user: in figure 1, after action $a_t$, state $s_t$ is changed to a new state $s_{t+1}$ and of the user. Consequently, the system receives an updated state estimate as well as a signal that represents the cost associated with taking the last action. However, unlike conventional control problems, where control decisions are made at regular intervals, in the domain of context-aware application, the intervals between events can be irregular. As a result, the control decisions also occur at irregular intervals. For example, the user may receive emails consecutively and then wait a long time before the next email arrives. In this case, the system may choose to take actions to reduce the frequency of checking for emails and direct the available resources for other more urgent tasks. The approach SMDP Q-learning approach handles the irregularity issue, and allows for learning the system change behavior as a function of user context.

![Figure 1: Control Problem Formulation](image)

In the formulation of SMDP (figure 2), the world model is that from a given state, the system makes a choice of an action. Then at some *nondeterministic* time later, the state changes and system receives a report of the new state, plus the immediate cost. The goal is to take the sequence of actions that minimizes the discounted sum of these costs. In particular, the system would choose an action $a_t$ to minimize the expected discounted return (total cost starting from time $t$):

$$C_t = c_{t+1} + \gamma c_{t+2} + \gamma^2 c_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k c_{t+k+1}$$
where $\gamma < 1$, is a discount factor that causes costs for further away (time-wise) actions to have less impact on the value of the current state-action pair. To achieve minimizing the discounted sum of costs, we need to estimate the 'value' of taking an action from a given state. The value of an action is defined as the negative discounted sum of costs that result from selecting the action now, and then selecting the (estimated) highest value action for the rest of the sequence. SMDP Q-learning is an algorithm that incrementally estimates these action values, optimizing the parameters to minimize the discounted sum of the future costs. The discrete SMDP Q-learning value update function is given as below [21]:

$$
Q_{t+\tau}(s, a) \leftarrow Q_t(s, a) - \alpha[c_{t+1} - \gamma c_{t+2} - \gamma^2 c_{t+3} - \ldots - \gamma^{\tau-1} c_{t+\tau} + \gamma^\tau \max_{a_{t+\tau}} Q_t(s_{t+\tau}, a_{t+\tau}) - Q_t(s, a)]
$$

where $Q_t(s, a)$ is the estimated action value for action $a$ when state is $s$. We assume the duration of the action $a$ is $\tau$, which varies with different actions. The cost associated with the selected action at time $t$ is represented as $c_t$.

In general, the state representation of each context-aware application varies on the case-by-case basis. However, there are some commonalities among the state representations. For instance, for many applications, time is often an important variable that should be part of the state description. Due to the continuous nature of time, the state space then becomes continuous. Therefore, we have chosen to use function approximation (FA). Instead of learning an enormous action-value lookup table as in the discrete case, with FA, an approximate continuous multi-dimensional surface function is learned to best-fit the real action-value function.

Furthermore, when facing unscheduled human behaviors that lead to unforeseen situations (or unvisited states), continuous FA allows for extrapolation on the past learned action-values that are similar, and thus informed guesses of the state-action value can be made. Later, these guesses can be improved and updated when new experiences arrive. Therefore, using function approximation, a mapping from multiple sensor readings to an appropriate system action can be learned despite the continuous state space. More importantly, it can do so robustly with respect to unscheduled human behavior.

### 3. Adaptive Power Conservation

To demonstrate the application of learning control optimization approach for multi-sensor fusion with conflicting objectives, we chose the mobile device power conservation problem as an example. It should be noted that the approach is not limited the example domain. For instance, the same problem as well as the solution apply to the domain of mobile robots. Under the context of mobile devices, the system needs to take the appropriate sequence of actions that makes the trade-off between two conflicting goals of both conserving power to stay on for longer and minimizing missed events. Clearly, this is a control optimization problem.
### Table 1: Cost Model

<table>
<thead>
<tr>
<th>Case</th>
<th>action taken</th>
<th>events</th>
<th>$c_{power}$ (joules)</th>
<th>$c_{event}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OFF</td>
<td>user event</td>
<td>279</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>OFF</td>
<td>non user event</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>OFF</td>
<td>none</td>
<td>279</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>KEEP ON</td>
<td>any</td>
<td>186</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 3.1. The Learning Agent

To formulate this as a SMDP Q-learning problem, a cost model must first be established. The costs for staying ON and for missing events has to be combined into the same cost model such that the value of for an action can be computed (using equation (1)) for evaluation. We denote the cost for missing events as $C_{event}$, and cost for power consumption as $C_{power}$. There are two types of power consumption: keeping the PDA running and switching from OFF to ON. Using the Sharp Zaurus as an example (whose power consumption is 6.2 watts), we can compute the power needed to keep the PDA ON per unit time (30 seconds) as $C_{power,run} = 6.2 \times 30 = 186$ joules. We assume that more energy is required to switch the PDA ON than to keep it running. Therefore, we define $C_{power,on} = 1.5C_{power,run} = 1.5 \times 186 = 279$ joules. For missed events, there are also two types: user events and non-user events and the associated cost for both is denoted as $C_{event}$. However, due to the fact that the units for $C_{power}$ and $C_{event}$ are different, that they cannot be combined directly. Therefore, a regularization parameter $\alpha$ is introduced for the purpose of costs aggregation. Specifically, using $\alpha$, $C_{event}$ can be converted to units in joules, denote as $C'_{event} = \alpha C_{event}$ joules. Finally both costs can be combined to compute the total cost:

$$c_t = c_{power} + c'_{event} = c_{power} + \alpha c_{event}$$

For any action $a$ taken, assuming the action execution time is $\tau$, at any given time instance $t$ on the time-line, the costs $c_{power}$ and $c_{event}$ are defined as follows:

In case 1, the system decides to turn OFF for time $\tau$, and a user event occurred during the sleep time. The user was forced to manually turn on the PDA since he needed to use the PDA. Thus, the total cost $c_t$ includes both the cost of switching ON the PDA, and the cost of missing a user event. For case 2, again, the OFF action was chosen, and we assume for this duration, only non-user events (such as changing rooms, or incoming emails) were missed. Compared with missing user events where the user would be forced turn PDA ON repeatedly, missing non-user events causes less annoyance to the user. Therefore, we define $C_{event} = 0.5$ in this case to reflect lesser annoyance. Since the PDA does not turn ON until the current OFF action ends, no cost for power consumption is needed. For case 3, since nothing happened during the OFF period, no event is missed and the cost only involves power spent for switching ON at the end of the OFF period. For the last case, where the system chose to stay ON to anticipate for future events, since no events will be missed during this time, the only cost is the accumulative cost of staying ON for $\tau$ time, which is the length of the selected action. These cover all the cases for all possible actions.

We assume the state of the world (with respect to the user) can be captured using the available sensors. The state representation in this application includes the following state variables: location, application usage, time of events, scheduled events and recent activities. Due to the continuous nature of some of the variables, function approximation is used to represent the state-action value function. Function approximation allows for generalization of data such that can robustly handle unforeseen situations. A Linear neural network was used as the function approximator. For the available action set, we chose 4 ON actions and 4 OFF actions. The durations of these actions are: 30 seconds, 1 minute, 2 minutes and 5 minutes, both for the ON and OFF set.
3.2 User Activity Simulator

To allow for analysis the SMDP Q-learning algorithm under various situations, a large data set of diverse, interesting and repeatable user activities need to be collected. An user activity simulator was implemented to generate simulated user experiences. This guarantees repeatability, which is crucial for in-depth analysis.

The simulator contains five different probabilistic activity models for the same user: each one corresponds to one day of the weekdays. In the model, the user has a different schedule for each day. For instance, for Monday (figure 3), the user has scheduled two classes in the morning and one lab meeting in the afternoon and with some time in between to work in the lab, as well some free time during the lunch break. Throughout the day, the user context transitions from one state to another based on the defined the activity duration model and the probabilistic transition model. Each activity has a predefined a duration, e.g. classes are all 75 minutes long. However, each person has different habits of arrival time for different events, some early, some late. Also, unpredictable events (buses late, weather conditions) also add randomness to the start and end time for these events. To model this variability, we introduce an user action uncertainty parameter $\sigma$. Now, the arrival and departure time is defined by the nominal transition time $t$ and a random offset generated from a Gaussian distribution: $t + \mathcal{N}(0, \sigma_1)$, where $\sigma_1$ is proportional the $\sigma$ and is different under different context. The relationship between $\sigma$ and $\sigma_1$ is given in Table 2 in the appendix. As for the transition probabilities, by design, the scheduled events have a high transition probability (e.g. 90%) such that they are more likely to occur. Unscheduled events are also modeled with transition probabilities, e.g. with 5% probability, class 1 may run late and the user may skip the free time and directly transition to class 2.

Once the simulator transitions into a state, user events will be simulated using actions from the behavior model, such as note-taking, email-checking behaviors defined under the current context. For example, during class the user exhibits a frequent note-taking behavior. When the user works in the lab, much longer rest intervals are more common since the user tends to use the desktop during this period, with only occasional checks of calendar schedules. In essence, the activity model uses the scheme shown in figure 4: it transitions between the active working state and inactive idle state, with a nominal idle interval period, defined by $\mathcal{N}(\mu_2, \sigma_2)$, where $\mu_2$ is the mean transition time, and $\sigma_2$ again is the user action uncertainty parameter. The relationship between $\sigma_2$ and $\sigma$ is also given in Table 2. As a result, the higher the value of $\sigma$, the more randomness is introduced in idle time between events and arrival time. Upon each simulated event, the event is logged as well as the corresponding sensor readings (such as time, or scheduled activity) are generated and recorded at the same time. These form the simulated sensory stream to be observed by the learning agent.

Figure 3: Monday Schedule User Activity Transition Probabilistic Model
4. Experiments and Discussions

The goal of the following experiments is to explore the effects of the control optimization technique: (1) with our approach, the agent can learn to translate multiple raw sensory inputs into appropriate sequence of system actions (for which we will use the term *policy* in the remaining discussion), optimizing on two conflicting goals; (2) our learned policy can outperform the user-defined policies; (3) by combining multiple sensor information, the learned policy handles unscheduled events reliably.

4.1 Learning appropriate action policies from multiple sensor streams

The first experiment examines the feasibility of the learning control approach to produce appropriate action policies from multiple raw sensor data streams. The agent is trained on a series of simulated experiences drawn from the Monday activity model. One hundred daily experiences are then drawn separately for evaluation. Performance for a single trial is measured in terms of the amount of power consumed and the number of missed events.

The top panel of figure 5 shows how the system behaves as the context of the changes throughout the day. Note that when the user is in a class, the agent prefers to turn the PDA off for 2 minutes over the other off actions. Occasionally, 5-minute off action were used. This is the case because the user exhibits a frequent note-taking behavior, with idle intervals drawn from Gaussian distributions with a mean value of either 2 or 5 minutes. When the user works in the lab, much longer rest intervals are more common since the user tends to use the desktops during this period, with only occasional checks of calendar schedules. Likewise for the lab meeting, although the user also takes notes during lab meetings, a different usage pattern is used. Therefore, the system chooses a distinctively different on/off pattern compared with the class sessions. Most of the user events are anticipated since the agent is able to turn on the system at the right times. Notice that although the agent can select 1-minute or 30-second off actions, these are never chosen. This is because the cost associated with turning on the PDA after an off action is higher than simply staying on for same the duration.

The bottom panel of figure 5 summarizes the maximum values of the on action and off actions over a single trial. At any given time, the action selection policy is greedy with respect to the action values, i.e. the action with the highest value is chosen. The shapes of the $Q$ action value function are noticeably different when the context of the user switches from one activity to the next. For instance, the greedy action switches between on and off more frequently in the class period than the work period in the lab. As a result, the agent using the learned policy wakes up much more frequently in anticipation for user events. On the other hand, the off period is much longer when the user is working in lab. The choices the learned policy made are reasonable because during classes, the user works in an active note-taking mode with some intermittent idle time when listening to lectures. Therefore, more frequent wake-up actions are needed. When working in the lab, the user tends to use the desktop instead of the PDA. During this period, the PDA is occasionally used for checking upcoming appointments. As a result,
Figure 5: Learned behavior over a single trial ($\alpha = 8$). The top panel illustrates how the system behaves under different user contexts. The horizontal axis is the time line, from 9:15 to 18:00. The top 5 rows show which actions were selected at any given time. The 6th row shows the duration of each scheduled activity (e.g., in a class, in a meeting). Anticipated and missed are shown as tick marks in the two remaining rows. The bottom panel summarizes the maximum values of the on (dotted line) and off (solid line) actions over time. The agent has learned to adapt to the user’s habits by switching to the appropriate on/off pattern when the user context changes.

It is interesting to note that the action value for the off action dramatically decreases shortly prior to the occurrence of an event (allowing the on action to take over). This is a reasonable strategy since the cost function assigns a cost to those actions that result in missed events. On the other hand, the choice of an on action immediately following an event has more value than an off action, since the likelihood of an event remains high sometime after the previous event. Staying on allows the system to avoid receiving the high cost of missing an user event. Therefore, over time, the agent learns that the values for the off actions is much lower than the on action when an activity occurs. Similarly, there are also costs associated with on actions. Therefore, the phenomenon is reversed when during idle periods such that the value of on action decreases and the value of off action recovers. Thus, off mode takes over during idle periods. The training experiences and the design of the cost model dictate the expected action values. In turn the action values dictate the resulting policy that uses different on and off patterns under different user contexts.

4.2 Varying the relative cost of power consumption and missed events

Since the cost model is one of the important factors that dictates the resulting policy, it is conceivable that changing the cost model will result in a series of different policies and thus a variety of system behaviors (figure 6). Compared with the top panel of figure 5, the action selection behavior change most occurred in the lab meeting session: due to a different cost model, less 5-minutes off actions, more 2-minute off actions and on actions were selected. As a result, fewer events were missed. In design of the cost model, a regularization power/correctness trade off parameter $\alpha$ was introduced, to linearly combine the costs associated with power consumption and missed events.
As a side-effect, changing $\alpha$ also changes the relationship between the two conflicting costs in the model, thus in essential altering the cost model. The effects of varying $\alpha$ are studied in the following experiments. Under low $\alpha$ value conditions, less cost is incurred for missing events, the agent learns a policy that is prone to conserving energy than staying on longer to wait for possible events. As the $\alpha$ increases, the cost for missing an event is raised, thus policies will be tuned to be more and more sensitive to missing events, and eventually approaches some policy that minimizes missed events. Trial performance of three learned policies are shown in figure 7. Each data point represents the performance of the learned control policy over a single trial. For each learned policy, the performance on 30 distinct trials is shown. As shown in the figure, there is a distinct distribution associated with each $\alpha$ value (put supporting p-test results here). This suggests that for a given cost model ($\alpha$), the agent learned a control policy that balances the goals of both conserve power and minimize missing events with respect to the cost model. Also, since the policy captured the activity model, it was able to reliably produce similar performance over a hundred trials drawn from the same probabilistic activity model. As $\alpha$ increases, better performance is achieved with respect to the number of missed events, this comes at the cost of increased power consumption. To view the trend with a wider range of $\alpha$ values, the overall performance of an individual control policy is measured as an average over one hundred trials of the number of missed events and of the consumed power. Figure 8 shows the overall performance for policies resulting from N-values of $\alpha$. A Least-Mean-Squared-fit inverse function ($E = \frac{a}{P} + b$: $E$ denotes the number of missed events and $P$ denotes the amount of power consumed) is superimposed on the data, and shows a general trend of trading power consumption for missed events as $\alpha$ is increased.

It is interesting to note that adjustment of the $\alpha$ parameter can be used as an intuitive mechanism for the user to alter the system behavior in order to suit his needs. If the user feels the system be more attentive to his activities, then he can increase the cost for missing events by increasing $\alpha$. If he is willing to sacrifice missed events in exchange for longer battery life, then $\alpha$ should be lowered. The control optimization formulation of the problem allows us to translate cost functions into a specific prescription for action under different contexts, and thus an appropriate performance trade-off solution for a certain behavior value can be computed. The end result is a much simpler and more flexible way to adjust system behavior than the multi-variate rule set system that needs to be manually predefined by the user.

For comparison purposes, we designed two heuristic policies. The Power-saving heuristic is defined to ag-
Figure 7: Trial performance under three learned policies ($\alpha = 8, 16, 22$). For each learned policy the performance on 30 distinct trials is shown. As $\alpha$ is increased, better performance is achieved with respect to the number of missed events. This comes at the cost of increased power consumption.

Figure 8: Performance of a range of learned policies trained under 10 cost models, compared against the performance of the user-defined policies. For the learned policies, as $\alpha$ increases, better performance is achieved with respect to the number of missed events, this comes at the cost of increased power consumption. Comparing against the user-defined policies, a learned policy can be found such that it performs significantly better than the user-defined policy in one dimension if the performance over the other dimension is matched.
gressively conserve power with no regard for missing events: it turns off the PDA whenever the system is idle. This is the lower-bound for power conservation because it represents the minimal power necessary to address the user’s requests, with no regard for missing user events. The balanced heuristic consists of a set of hand-designed rules that are intended to balance power consumption against the number of missed events (the rule set is given in Appendix A). As with the learned control policies, performance of these heuristics is measured by averaging the power consumption and the number of missed events over one hundred trials.

As shown in figure 8, the power consumption of the balanced heuristic is roughly equivalent to that of the learned policy when $\alpha = 16$. However, the learned policy significantly outperforms the heuristic with respect to the number of missed events. Likewise, the number of missed events of the balanced heuristic is roughly equivalent to that of the learned policy when $\alpha = 8$. However, the learned policy significantly outperforms the heuristic with respect to the power consumption (show paired-t-test values here). For the power saving heuristic, by taking advantage of the curvature of the performance function with respect to $\alpha = 8$, we can find an $\alpha$ that is similar in power-consumption, and yet the performance in the events missed category is significantly better than that of the heuristic (show paired-t-test values here).

4.3 Sensitivity Analysis

Although human behavior exhibits patterns that have strong correlation with scheduled activities, there are times when the user will deviate from the typical schedule. Some of these deviations are dramatic, e.g. canceling meetings, changing locations, or even moving scheduled events to an earlier time and date. Some deviations are minor, e.g. arriving late for classes. We examine the robustness of the learned control policies against both of these classes of unscheduled human activities. For scheduled activities, both the training and test trial experiences are drawn from the same activity model. We simulate the unscheduled activities by drawing training and test trials from different activity models. Since each policy is trained for a specific day’s activity model, presenting trials drawn from a different day model to the agent creates the effect of drastic deviation from original schedules. In the first experiment, both scheduled, and unscheduled trials were presented to the agent for evaluation. Overall performance for a single learned control policy is measured as an average over the trials of the same type (scheduled vs. unscheduled). As with the previous results, a series of cost models (different $\alpha$) were used for training in each case. Results are shown in figure 9. Although performance is consistently made worse by the introduction of the unscheduled activities, the control policies in the latter case still demonstrate a clear trade-off between power consumption and the number of missed events that is controlled by the selection of a particular value of $\alpha$. This shows that the agent is able to learn a policy that handles unscheduled activities robustly.

In designing context-sensitive devices (and in particular those that are calendar-sensitive), we expect some users to adhere closely to their scheduled activities, whereas we expect others to demonstrate a significant amount of variability in their arrival times to some activities. It is critical that any approach to learning context-sensitive behavior be robust to this range of users. Here, we model the latter user with the user action uncertainty parameter $\sigma$ such that more different random idle intervals between user events can be generated. Experiences were drawn from this model for both training and test trials. Evaluations show (figure 10 ) that as randomness of the activities increases, the performance of the learned policy (performance trend with respect to $\alpha$ (the ALL case) slightly dropped. The drop is more apparent toward the power saving end. However, the important thing is that the trend of performance trade-off with respect to the the corresponding $\alpha$ value is retained. This demonstrates the robustness of the learned policy obtained using the current state representation.

Finally, we examined the sensitivity of the system with respect to the chosen state representation. This is important since choosing the appropriate state variable (where a feature or a set of features is used to encode a variable) to comprise our state representation is crucial to the performance of the learning agent. Although using our current representation, the learned policy can outperform the example heuristics, the question that whether these are right
5. Conclusions

This paper presented a Reinforcement Learning approach to multi-sensor fusion problems with conflicting objectives. An example application of the approach was given in the domain of adaptive power management for mobile devices. We showed that by designing an appropriate cost model for the application, a cost can be associate with each system action. Over time, the system can learn to choose a sequence of actions that makes trade-off between the two conflicting goals, and minimizes the total costs. Through experiments with simulated user activities, we have shown that (1) with our approach, the agent can learn to translate multiple raw sensory inputs into appropriate sequence of system actions, optimizing on two conflicting goals; (2) our learned policy can outperform the user-
Figure 10: Sensitivity analysis. Performance of learned policies when the *user action uncertainty* parameter is varied ((a) $\sigma = 0.5$ and (b) $\sigma = 4$). The solid line represents the policy learned using *full state representation*. The long dotted line is the policy learned using a state representation that *includes recent activity* only, and the short dotted line is one that only *excludes* the *recent activity*.

defined policies; (3) by combining multiple sensor information, the learned policy handles unscheduled events reliably.

## A Balanced Heuristic

**Pseudo Source code**

```plaintext
if(state[E_ACTIVITY]==RUN_APPLICATION){
    action = STAY_ON_30_SECS;
```
```c
}
else{
    if(LastSeenActivity == WITHIN_3_MINS ){
        action=STAY_ON_1_MIN;
    }
    else if(iLastSeenAct == WITHIN_5_MINS){
        if(state[E_EVENT]==CLS_ROBOTICS ||
            state[E_EVENT]==CLS_NETWORKING){
            action=SLEEP_5_MINS;
        }
        else if(state[E_LOCATION]==LOC_HOME ||
            state[E_LOCATION]==LOC_HELL144 ||
            state[E_LOCATION]==LOC_CS150){
            action=SLEEP_3_MINS;
        }
        else{
            action=SLEEP_1_MINS;
        }
    }
    else if(iLastSeenAct == WITHIN_8_MINS){
        if(state[E_EVENT]==CLS_ROBOTICS ||
            state[E_EVENT]==CLS_NETWORKING){
            action=1;
        }
        else if(state[E_LOCATION]==LOC_HOME ||
            state[E_LOCATION]==LOC_HELL144 ||
            state[E_LOCATION]==LOC_CS150){
            action=SLEEP_5_MINS;
        }
        else{
            action=SLEEP_1_MINS;
        }
    }
    else if(iLastSeenAct == WITHIN_12_MINS){
        if(state[E_EVENT]==CLS_ROBOTICS ||
            state[E_EVENT]==CLS_NETWORKING){
            action=STAY_ON_1_MIN;
        }
        else if(state[E_LOCATION]==LOC_HOME ||
            state[E_LOCATION]==LOC_HELL144 ||
            state[E_LOCATION]==LOC_CS150){
            action=SLEEP_5_MINS;
        }
        else{
            action=SLEEP_3_MINS;
        }
    }
}
B Activity model - activity duration and transition time model

<table>
<thead>
<tr>
<th>Event</th>
<th>t(start/end)</th>
<th>$\sigma_1$</th>
<th>$\mu_2$</th>
<th>$\sigma_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class(1)</td>
<td>9:30am/10:45am</td>
<td>2$\sigma$</td>
<td>10</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>Class(2)</td>
<td>11:15am/12:30pm</td>
<td>2$\sigma$</td>
<td>10</td>
<td>$\sigma$</td>
</tr>
<tr>
<td>Work(1)</td>
<td>1:00pm/3:15pm</td>
<td>4$\sigma$</td>
<td>5</td>
<td>$\sigma^*$</td>
</tr>
<tr>
<td>LabMeeting</td>
<td>3:30pm/5:00pm</td>
<td>2$\sigma$</td>
<td>15</td>
<td>1.5$\sigma$</td>
</tr>
<tr>
<td>Work(2)</td>
<td>5:00pm/7:00pm</td>
<td>2$\sigma$</td>
<td>5</td>
<td>$\sigma^*$</td>
</tr>
</tbody>
</table>

Table 2: activity duration and transition time model. (*) The idle intervals for the Work period depends on a different transition model: a work duration between 20 minutes and 1 hour is drawn from a uniform distribution. The PDA will be idle during the work duration. Between the work periods, the user uses the PDA to download files or check schedules and thus the similar activity model used in other events are used again.

References


