A Real-time Ubiquitous System: Real-Time Indoor Tracking of Humans and Objects for Assisted Living*

Min-Young Nam
Dept. of CS
University of Illinois
Urbana, IL 61801, USA
mnam@uiuc.edu

Mhd Zaher Al-Sabbagh
Dept. of ECE
The Ohio State University
Columbus, OH 43210, USA
als@ece.osu.edu

Jung-Eun Kim, Man-Ki Yoon, Chang-Gun Lee
School of CSE
Seoul National University
Seoul, 151-742, Korea
{deep0314,ymk3337,cglee}@snu.ac.kr

Eun Yong Ha
Dept. of CSE
Anyang University
Anyang, 430-714, Korea
eyha@anyang.ac.kr

Abstract

As the elderly population increases, the elderly care using inexpensive technological means becomes critical. This paper presents our prototype system that provides real-time indoor tracking of elderly residents and their belongings, which is essential to assist and secure their independent living. For high-fidelity real-time tracking, we propose novel scheduling algorithms. Our scheduling algorithms are designed by harmonizing both sensing and communication signals and leveraging location-awareness and mobility-consciousness, in order to improve the tracking accuracy while reducing the energy consumption. We performed extensive experiments through both simulation and actual implementation. Our experimental result says that our scheduling algorithms can provide real-time tracking of residents within 20 cm error bound in the typical range of human mobility.

*This work was supported in part by NSF-CSF 0509175 and in part by Research Settlement Fund for the new faculty of SNU. A part of this work has been done when the authors were in OSU. A preliminary version of this paper has appeared in the proceedings of IEEE Real-Time Systems Symposium (RTSS), 2006. The corresponding author is Chang-Gun Lee.
1 Introduction

As the baby boomer generation ages, the elderly care has become a social and economical challenge worldwide. Therefore, it is very important to assist the elderly living by inexpensive technological means in order to enable the reasonably healthy elderly to independently live at their own homes instead of expensive nursing home facilities\(^1\).

This paper presents a technology-based assisted-living environment where we can provide real-time indoor tracking of the elderly residents and their belongings. The real-time indoor tracking of humans and objects is an important baseline service for realizing many useful high-level services such as finding objects for “where is” type inquiries, reminding necessary actions like taking medicine, and detecting emergency situations. Visual tracking is not attractive because it requires expensive image processing and also it costs too high to cover the entire home. There have been other types of location systems using different active sensing technologies [6, 27, 11, 5, 21, 9, 1, 14, 20]. However, they rather focus on reducing physical sensing error than sound scheduling algorithms of active sensing signals. Thus, they cannot provide high-fidelity real-time tracking.

Our goal is to implement a system that can provide high-quality real-time tracking for assisting the elderly living. Due to economical reasons, our prototype system uses a combination of ultrasonic and RFID technologies as the underlying sensing mechanism. Each of the ultrasonic transmitters called beacons mounted on the ceiling occasionally sends out an ultrasonic pulse and a short RF message at the same time. Due to the speed difference of an RF message (speed of light) and an ultrasonic signal (speed of sound), the listener on the wristband of each elderly resident can infer its distance from the beacon using the TDOA (Time Difference of Arrivals) of the two signals. This distance measurement

\(^1\)The current average for a semi-private room in a nursing home is $70,000 annually and is expected to increase to $190,000 by 2030 [19].
can be reported to the host computer for real-time tracking of the user. Also, the RFID (radio frequency identification) reader attached on the same wristband can read RFID tags of objects touched by the user. Thus, we can track the objects as well, whenever the user carries them. Since the beacons use active probing signals, i.e., ultrasonic pulse and RF message, if two beacons within the interference range transmit the active signals at the same time, they collide. The state-of-the-art ultrasonic based location system, Cricket [21], addresses the collision problem based on carrier sensing and random arbitration. However, it cannot ensure high-quality and predictable tracking performance.

This paper proposes collision-free scheduling algorithms by adopting the following ideas step-by-step:

• Harmonizing sensing and communication: The system should schedule active ultrasonic signals (i.e., sensing) while exchanging RF messages (i.e., communication) among beacons, listeners, and the host computer. We maximally overlap the duration of ultrasonic signal and RF messages, since they do not collide, to allow a higher sampling rate.

• Location-aware scheduling: Two beacons may or may not collide depending on listener’s location. We form a feedback loop between the scheduler and tracking tasks and utilize listener locations given by tracking tasks so that the scheduler can build a better schedule.

• Mobility-conscious scheduling: We adaptively control the sampling frequencies of beacons depending on the mobility of users—low rate for low mobility and high rate for high mobility. This way we can significantly save the energy consumption without compromising the tracking accuracy.

The rest of this paper is organized as follows: The next section briefly surveys the related work. In Section 3, we present the overall system configuration and then formally describe the scheduling problem for real-time indoor tracking. Section 4 proposes our novel scheduling algorithms for orchestrating both ultrasonic and RF signals. Section 5 addresses the practical issues of 3D real-life environments.
Section 6 presents the experimental results through both simulations and actual implementation. Finally, Section 7 concludes this paper.

2 Related Work

There have been a number of indoor location systems targeting different applications. A typical example is visual tracking [17, 23]. Although it can provide an acute tracking of human faces and motions, it requires complex image processing and also costs too much to cover the whole assisted living environment. To avoid complex signal processing, other location systems use different active probing signals, for example, IR (Infra Red) signals in Active Badge [6], RF signal strengths in Microsoft RADAR [1] and SpotON [9], RFID tags in the systems of Intel and Univ. of Washington [20] and Utah State Univ. [14]. However, due to the inherent limitation of their sensing technologies, their location accuracy is several meters at best. For our assisted living environment, one option that gives an acceptable localization accuracy is an UWB (Ultra Wide Band) location system like Ubisense [27] and Multispectral Solution Inc.’s PAL650 [11]. Although they provide location accuracy of tens of centimeters, their current unit price is more than U.S. $10,000, which counters one of our important objectives—inexpensive deployment of our assisted living system.

An ultrasonic signal can be an inexpensive alternative. Both Active Bat [5] and Cricket [21] use active ultrasonic signals and RF messages to localize a target based on the TDOA (time difference of arrivals). They can provide location accuracy of tens of centimeters for a stationary target. One difference is that Active Bat uses the “active mobile” paradigm in which the mobile target actively sends out the signals while Cricket uses “passive mobile” paradigm in which the mobile target passively listens to probing signals. We prefer the cricket configuration because of the energy issue of the mobile user device—we do not want to make the battery operated user device actively generate ultrasonic signals.
In terms of spatial accuracy, the existing ultrasonic-based location systems are acceptable. However, for real-time tracking of mobile users, multiple devices should frequently send out the active probing signals and hence their scheduling to avoid collisions becomes an essential issue. Even the state-of-the-art cricket tracking system [21] handles the scheduling problem using a naive approach based on carrier-sensing and random arbitration, i.e., CSMA-CA. Although such mechanism works for a low sampling rate, it stops working when we do need more frequent sampling for real-time tracking, due to the increased probability of collisions.

There have been many efforts to use graph coloring approaches to schedule sensor nodes avoiding collisions in the wireless and sensor network area [12, 15, 16, 7]. However, they focus on only the communication aspect, i.e., scheduling RF messages among sensor nodes, without addressing the combined aspect of active sensing and communications. Also, there are many MAC protocols such as traditional PRMA (Packet Reservation Multiple Access) [8] and recent ZigBee/IEEE 802.15.4 [13] for collision-free scheduling among competing nodes. Although the scheduling principle for avoiding collisions has similarities to our approach, they are again targeting for only communications. Thus, there is no consideration about combined scheduling of sensing and communication and dynamically adapting the schedule depending on listener’s location and mobility.

3 System Overview and Problem Description

We first give a big picture of the entire system configuration of our assisted living prototype. As shown in Figure 1, the system consists of one host computer (i.e., \(HS\)), a number of ultrasonic transmitters called \textit{beacons} (i.e., \(B_i\)) mounted on the ceiling, a number of wristband devices (i.e., \(L_j\)) carried by residents, and a number of RFID tags attached to all objects in the home including furniture and small stuffs like a teapot, a remote controller, and eye-glasses. We use \(N_B\) to denote the total number of
beacons and $N_L$ to denote the total number of residents or wristband devices. Each wristband device $L_j (1 \leq j \leq N_L)$ as shown in Figure 1 consists of an ultrasonic listener and an RFID reader connected through a serial cable. We will use the same notation $L_j$ to denote the combined wristband device or to denote its ultrasonic listener only.

For a beacon $B_i$ mounted on the ceiling, we use a Crossbow Cricket mote [4] as in Figure 2(a) that has a small processing power, wireless RF communication capability, and ultrasonic transmission and reception capability. It costs less than U.S. $10$ [21]. For a wristband $L_j$, we use the same Crossbow Cricket mote as the ultrasonic listener and a Skyetek M1-Mini [24] in Figure 2(b) (costs less than U.S. $10$) as the RFID-reader. The ultrasonic listener and RFID reader are connected through a custom-made serial cable and also the RFID reader’s internal antenna is hooked-up to the custom-built external antenna to extend the RFID read range. Figure 2(c) shows such a combined wristband device $L_j$. Note that the size of the current wristband is a little bit larger than that of wrist watch, since it is built using
the COTS hardware components. However, once we validate the system with COTS components, we plan to make a specialized device including ultrasonic listener and RFID reader in a single optimized hardware component and minimizing the overall size. With such optimized hardware component, we expect a small wristband that an elderly user feels comfortable to wear all the times. For all the objects, we attach TI RFID tags [24] as in Figure 2(d), less than 10 cents per each.

With this configuration, the scenario for tracking both humans and objects is as follows: Each beacon $B_i$ occasionally transmits an ultrasonic signal denoted by $US(B_i)$ together with a short RF message denoted by $RF_{us}(B_i)$, marked as ➀ in Figure 1. Since an RF message flies much faster at the speed of light than an ultrasonic signal that flies at the speed of sound, the wristband ultrasonic listener $L_j$ will receive the RF message before the ultrasonic signal from the same beacon. Using such time gap called TDOA (time difference of arrivals), the listener of $L_j$ can infer its physical distance from the beacon $B_i$. Since $US(B_i)$ and $RF_{us}(B_i)$ are used for sensing the distance, we call them sensing signals. At the same time, the RFID reader of $L_j$ searches for RFID tags in its proximity if any and reads their IDs. Then, $L_j$ sends an RF message to $B_i$, denoted by $RF(L_j)$ that includes the inferred distance information and the most recent RFID readings, marked as ➁ in Figure 1. Then, $B_i$ relays it using a high power RF message $RF(B_i,H_S)$ to directly reach the host computer, marked as ➂ in Figure 1. The host computer feeds the new distance measurement to the EKF (Extended Kalman Filter) tracking algorithm so that it filters out the distance measurement noise and estimates the target position by triangulation. Associated with
this new location estimate, we can use the RFID readings in the same RF message to localize the RFID tags and their attached objects within the range of RFID reader—less than 10 cm for the short-range RFID reader we are using in the prototype. After that, the host computer can broadcast a high power RF message denoted by $RF(HS, B)$ as needed for global clock synchronization and global schedule update of all beacons, marked as ④ in Figure 1.

In order to make such tracking scenario work, the challenge we tackle in the paper is how to schedule ultrasonic signals, i.e., $US(B_i)$ from multiple beacons and RF messages, i.e., $RF_{us}(B_i)$, $RF(L_j)$, $RF(B_i, HS)$, and $RF(HS, B)$ from multiple beacons, listeners and host computer.\(^2\)

Since RFID devices work in a separate frequency band, they do not interfere with the RF messages. Also, positioning accuracy of RFID tags (and hence objects) depends on the accuracy of tracking user’s wristband device $L_j$. Thus, for the simplicity of explanation, we will move out the RFID part from our consideration and focus on tracking user’s wristband device using ultrasonic signals and RF messages.

If the ultrasonic signals and RF messages are not properly orchestrated, there are two types of collisions: (1) If the duration of ultrasonic signals from two close beacons overlap, they become a garbage signal. Thus, a listener cannot use it for distance measurement. (2) If the duration of RF messages from interfering beacons, listeners, and host computer overlap, they cannot successfully be received. Thus, our goal is to realize collision-free scheduling of the sensing resource, i.e., ultrasonic band and the communication resource, i.e., wireless RF band, for the best performance of real-time tracking.

For more formal description of our scheduling problem, we define two ranges for each beacon, coverage and interference range. The coverage of a beacon $B_i$ is defined as a disk within which a listener can successfully receive both $US(B_i)$ and $RF_{us}(B_i)$ to infer the distance. Note that if a listener $L_j$ is under

\(^2\)We ignore the RF interference from neighbor apartments by assuming that the installation phase can automatically configure the system with a free RF frequency channel where no activity is observed from neighbor apartments. The US interference from neighbor apartments is not likely to happen because the US signals can be well contained by the walls.
the coverage of a beacon $B_i$, its RF message $RF(L_j)$ can also reach $B_i$. In contrast, the interference range of a beacon is defined as a disk within which either $US(B_i)$ or $RF_{us}(B_i)$ may interfere with the same type signals from other beacons. The interference range is usually larger than the coverage.

We can obtain those ranges through actual experiments [22]. Just for the pictorial simplicity, all the following figures will use the same disk for the two ranges. However, we will clearly state when we have to consider which range. Also, note that the interference range of two types of high power RF messages, i.e., $RF(B_i, HS)$ and $RF(HS, B)$ covers the entire home space to allow direct communications between the beacons and host computer. Thus, only one instance of such high-power RF messages can be scheduled at a time.

With the above notations and definitions, we can now formally define our combined ultrasonic and RF scheduling problem as follows:

**Problem Description:** Find a sequence of times denoted by $t_k$ when the above signals can be transmitted, i.e.,

1. $t_k(US(B_i)) = t_k(RF_{us}(B_i)), k = 1, 2, \cdots, \text{for all beacons } B_i(1 \leq i \leq N_B)$,
2. $t_k(RF(L_j)), k = 1, 2, \cdots, \text{for all listeners } L_j(1 \leq j \leq N_L)$,
3. $t_k(RF(B_i, HS)), k = 1, 2, \cdots, \text{for all beacons } B_i(1 \leq i \leq N_B)$, and
4. $t_k(RF(HS, B)), k = 1, 2, \cdots, \text{for the host computer } HS$, such that no two ultrasonic signals and no two RF messages collide.

A naive approach to this problem is to schedule the signals in sequence as shown in Figure 3: $B_1$ transmits $US(B_1)$ and $RF_{us}(B_1)$, $L_j$ under $B_1$’s coverage infers the distance and reads RFID tags and delivers this information to $B_1$ using $RF(L_j)$, $B_1$ relays the information to $HS$ using high-power $RF(B_1, HS)$, $HS$ broadcasts high-power $RF(HS, B)$ for global time-synchronization and schedule update, and then $B_2$ takes turn. However, this approach is not necessarily good. First, it allows only one beacon to send signals at a time even if two beacons are far away and thus never collide. Second, for the duration while waiting for an ultrasonic pulse to propagate and be silenced out, which is much
longer than an RF message transmission time, the RF resource is idle. Finally, it may over-sample the
listener positions even when the user is moving very slowly or not moving at all, which is just a waste
of energy.

Understanding these problems, our scheduling algorithm is designed with the following two goals:

- **Maximize the affordable sampling rate for the case of fast moving users.** The *affordable
  sampling rate* of each beacon is defined as the maximum possible frequency with which each
  beacon can transmit sensing signals without collision. This goal is achieved by allowing concurrent
  executions of non-colliding multiple signals and pipelining ultrasonic signals and RF messages.

- **Minimize the energy consumption of mobile user device, i.e., listener, without com-
  promising the tracking accuracy.** This goal is achieved by adaptively controlling beacon’s
  sampling rate depending on the user mobility.

4 Combined scheduling of ultrasonic signals and RF messages

In this section, we first explain a *static scheduling* method that pipelines ultrasonic signals and RF
messages to minimize the resource idle duration (Section 4.1). Using the static scheduling as a baseline,
we present a *location-aware scheduling* method that actively uses up-to-date user location information
to dynamically build a better schedule (Section 4.2). Finally, we present one more step improvement,
i.e., a *mobility-conscious scheduling* method that adjusts the sampling frequency adaptively to the user mobility to save the energy consumption without compromising the tracking accuracy (Section 4.3).

### 4.1 Static pipeline scheduling of ultrasonic signals and RF messages

In contrast to the sequential scheduling as in Figure 3, the idea of pipelining ultrasonic signals and RF messages is motivated by a long duration of an ultrasonic signal. Once a beacon sends out an ultrasonic pulse, although the pulse itself is short, it lives for a quite long time until it propagates and is silenced out. For this duration, which we call *ultrasonic duration* (or *sensing interval*) and denote by $\text{Len}(US)$, it can confuse another ultrasonic pulse created within the interference range. This ultrasonic duration is determined by physical factors such as the required ultrasonic reaching range, strength of pulse power, and speed of sound. For the cricket mote [4], $\text{Len}(US)$ is tuned as 50 ms [22]. On the other hand, the duration of RF message is relatively short. Using the cricket mote that works with 19.2 kbps data transmission rate, the time for transmitting a short RF message $RF_{\text{us}}(B_i)$ of 32 bytes is approximately 15 ms. Thus, the *left-over time* of 35 ms can be used for other RF messages between listeners, beacons, and the host computer. The left-over time will increase as the wireless data transmission rate increases with advanced technology and thus a increasing amount of data can be exchanged using the left-over time.

Motivated by this, our scheduling algorithm pipelines the two stages of a single sampling; (1) the sensing stage using $US(B_i)$ and $RF_{\text{us}}(B_i)$ and (2) the reporting stage using $RF(L_j)$ and $RF(B_i, HS)$. Figure 4 shows an example of such pipelining. Each beacon $B_i$ takes turns to transmit its sensing signals, i.e., $US(B_i)$ and $RF_{\text{us}}(B_i)$. In the example, $B_1$ first sends out $US(B_1)$ and $RF_{\text{us}}(B_1)$ for $i$-th sensing. At the end of the ultrasonic duration of $US(B_1)$, a listener $L_j$ under $B_1$'s coverage can calculate its distance from $B_1$. This distance report, called the $i$-th reporting, can be made in the left-over time.
of $i+1$-th sensing—$L_j$ sends $RF(L_j)$ to $B_1$ and then $B_1$ relays it by sending $RF(B_1, HS)$ with a high-power directly to the host computer. The last part of each left-over time can be reserved for the host computer to broadcast the global synchronization and schedule updates, i.e., $RF(HS, B)$, as needed.

This two stage pipelining of sensing and reporting (i.e., communication) can reduce the resource idle period and thus make beacons take turns more frequently for higher rate sampling without collision.

For this pipelining idea to work, we have to address the following two detail issues:

- How to make the order of beacons so that they can take turns?
- If multiple listeners hear the sensing signals in the same sensing interval, how their reports can be arbitrated in the next left-over time?

For the first question, a simple minded solution is a round-robin where only one beacon is allowed to transmit its sensing signals. Although it guarantees a collision-free schedule of sensing signals, it cannot take advantage of concurrent scheduling of multiple beacons even when they are far away and hence never collide. Figure 5 shows an example. In Figure 5 (a), we use 11 beacons such that all the points of the rectangular space can be covered by at least three beacons. Such beacon placement is well studied in [21, 22, 26, 3] and is beyond the scope of this paper. If we use a simple round-robin method, it takes $11 \times \text{Len}(US)$ for one complete cycle of all beacons. However, if we consider the beacon positions and

\[3\] At least three beacons are required for triangulation.
the interference ranges, we can notice that some beacons never collide even if we schedule their signals at the same time. For example, $B_1$ and $B_4$ never collide and thus can be concurrently scheduled.

In order to take advantage of such possible concurrent scheduling, our first step is to draw a conflict graph considering the beacon positions and their interference ranges. The conflict graph $CG = (N, E)$ is formally defined as follows: $N$ is the set of nodes where each node is a beacon $B_i$. $E$ is the set of conflict edges. Each edge between a pair of nodes $B_i$ and $B_j$ represents that their interference ranges overlap. If there is no edge between $B_i$ and $B_j$, it is safe to schedule both at the same time. Figure 5 (b) shows such a conflict graph for Figure 5 (a).

With the conflict graph, the second step is to assign a color code to each beacon such that any two beacons with a conflict edge cannot have the same color. We can use a simple coloring heuristic [2] to find the coloring solution with the minimum number of colors. Figure 6 (a) shows such a coloring solution for the example conflict graph in Figure 5 (b).

Once we find the coloring solution, we can build a master schedule, which is a complete cycle of different colors. Figure 6 (b) shows the master schedule for the example coloring solution of Figure 6 (a).
Figure 6: Coloring and static schedule of beacons

The duration of each color in the master schedule is equal to the ultrasonic duration $Len(US)$. The master schedule is repeated. At each color of the master schedule, all beacons with the same color can transmit its sensing signals, $US(B_i)$ and $RF_{us}(B_i)$ without any collision. This way, we can maximize the concurrent scheduling of non-interfering beacons and hence reduce the time for completing a cycle of all beacons. For the example, we can complete a cycle with $7 \times Len(US)$. Thus, the affordable sampling rate of each beacon, which is the inverse of master schedule length, is $1/(7 \times Len(US))$. This is 57% improvement compared to the simple round-robin method.

The color codes and master schedule is statically calculated by the host computer only once in the beginning. Then, the host computer broadcasts the color code for each beacon and the number of colors. Once such information is given, each beacon $B_i$ can wake-up at the pre-determined schedule times as designated by the master schedule and send out its sensing signals $US(B_i)$ and $RF_{us}(B_i)$ without any collision with others.

Now, the remaining question is how to arbitrate them if multiple listeners hear the sensing signals in the same sensing interval. The worst case scenario is that all $N_L$ residents or listeners hear $i$-th sensing signals and thus all of them have to report their distances and RFID readings (i.e., the $i$-th reporting) using the left-over time of the $i+1$-th sensing interval. For this, we use a simple deterministic method. Specifically, we divide the left-over time into $K$ regular slots and one special slot for a $HS$ message as shown in Figure 7. Each regular slot is reserved for delivering a report from each listener.
to the host computer. Thus, the size of a slot should be the twice of an RF message length: one for a message from a listener \( L_j \) to a beacon \( B_i \), i.e., \( RF(L_j) \), and the other for relaying the message from \( B_i \) to the host computer, i.e., \( RF(B_i, HS) \). The number of slots, i.e., \( K \), that can be packed into the left-over time is usually large enough compared to the number of elderly residents, i.e., \( N_L \), in a home environment, because of the short RF message length compared to the ultrasonic duration. In our prototype implementation, we can pack three slots into the left-over time, which means that our system is applicable to the home environment where up to three elderly residents live. By using more advanced wireless technology, the time for transmitting an RF message can be significantly reduced and thus even the above simple method can easily be applicable to a larger scale home environment to a certain extent. Also, if we use wired connections between beacons and the host computer, we can eliminate the \( RF(B_i, HS) \) part from each slot and thus we can double the number of affordable residents.

### 4.2 Location-aware scheduling

One problem of the aforementioned static scheduling is the conservative definition of the conflict graph. As an example, consider two beacons in Figure 8 whose interference ranges overlap. According to our conservative definition of conflict, the two beacons have a conflict edge and thus cannot be concurrently scheduled. However, if there is no listener in the intersection of the two interference ranges,
they do not actually collide. In the figure, even if the two beacons transmit the sensing signals (i.e., \{US(B_1), RF_{us}(B_1)\} from B_1 and \{US(B_2), RF_{us}(B_2)\} from B_2) at the same time, they can be successfully received by L_a and L_b, respectively—no conflict edge between B_1 and B_2. Thus, considering listener locations allows us to build a more accurate conflict graph. Also, using the listener locations, we can selectively turn on only beacons necessary to cover the listener locations, which can reduce the number of nodes in the conflict graph and in turn reduce the master schedule length.

Motivated by this observation, our location-aware scheduling approach forms a feedback loop from tracking tasks to the scheduler so that the latter can use the location information from the former to dynamically update the schedule. We update the schedule periodically with period I which is an integer multiple of a sensing interval, i.e., \( I = m \times \text{Len}(US) \)—update the schedule at every m sensing intervals. For the duration of I, the same schedule will be used.

At the beginning of each update period, we use the location information of all listeners,

\[
(\text{Loc}(L_1), \text{Loc}(L_2), \cdots, \text{Loc}(L_{N_L})).
\]

Using the location of each listener and maximum stride speed \( v_{\text{max}} \), we can define the area the user can move within the next update period I. The area called a proximity area of a listener \( L_i \) is denoted by \( A(L_i) \) and defined as follows:

\[
A(L_i) = \text{circle centered at Loc}(L_i) \text{ with radius of } I \times v_{\text{max}}.
\]
Figure 9 (a) shows the proximity areas (shaded circles) of two listeners $L_1$ and $L_2$.

Once we have the proximity areas for all listeners, our next step is to select the minimal subset of beacons such that all proximity areas can be covered by at least three beacons. Our greedy heuristic algorithm to find such minimal subset is as follows:

**ActiveBeacons**: Find a subset of beacons to be activated

<table>
<thead>
<tr>
<th><strong>Input:</strong> Coverage of each beacon $C(B_i)(1 \leq i \leq N_B)$ and proximity area of each listener $A(L_j)(1 \leq j \leq N_L)$</th>
<th><strong>Output:</strong> $ActiveBeaconSet$ (subset of beacons to be activated)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>begin procedure</strong></td>
<td><strong>begin procedure</strong></td>
</tr>
<tr>
<td>1. $ActiveBeaconSet = \emptyset$</td>
<td>1. $ActiveBeaconSet = \emptyset$</td>
</tr>
<tr>
<td>2. for each listener $L_j$ in ${L_1, \ldots, L_{N_L}}$</td>
<td>2. for each listener $L_j$ in ${L_1, \ldots, L_{N_L}}$</td>
</tr>
<tr>
<td>3. make a preference list of beacons $preferBeaconList$</td>
<td>3. make a preference list of beacons $preferBeaconList$</td>
</tr>
<tr>
<td>4. repeat</td>
<td>4. repeat</td>
</tr>
<tr>
<td>5. Take out the first beacon $B_i$ from $preferBeaconList$</td>
<td>5. Take out the first beacon $B_i$ from $preferBeaconList$</td>
</tr>
<tr>
<td>6. Increase $A(L_j)$ coverage with $C(B_i)$</td>
<td>6. Increase $A(L_j)$ coverage with $C(B_i)$</td>
</tr>
<tr>
<td>7. if $B_i \notin ActiveBeaconSet$</td>
<td>7. if $B_i \notin ActiveBeaconSet$</td>
</tr>
<tr>
<td>8. Add $B_i$ to $ActiveBeaconSet$</td>
<td>8. Add $B_i$ to $ActiveBeaconSet$</td>
</tr>
<tr>
<td>9. end if</td>
<td>9. end if</td>
</tr>
<tr>
<td>10. until the entire area of $A(L_j)$ is covered by at least three beacons</td>
<td>10. until the entire area of $A(L_j)$ is covered by at least three beacons</td>
</tr>
<tr>
<td>11. end for</td>
<td>11. end for</td>
</tr>
<tr>
<td>12. return $ActiveBeaconSet$</td>
<td>12. return $ActiveBeaconSet$</td>
</tr>
<tr>
<td><strong>end procedure</strong></td>
<td><strong>end procedure</strong></td>
</tr>
</tbody>
</table>

The above procedure considers the coverage of each beacon $C(B_i)$ and proximity area of each listener $A(L_j)$ as inputs and then produces the set of beacons $ActiveBeaconSet$ to be activated as an output. In Line 1, it initializes $ActiveBeaconSet$ as the empty set. Then, the following for loop from Line 2 to Line 11 adds beacons to cover each proximity area $A(L_j)$. For each $A(L_j)$, Line 3 makes a preference list of all beacons to cover $A(L_j)$ with minimum addition of new beacons into $ActiveBeaconSet$. Our heuristic is to (1) first place all beacons already in $ActiveBeaconSet$—first try to cover $A(L_j)$ with no addition of new beacons, and (2) then sort all other beacons as the decreasing order of overlap size between $A(L_j)$ and $C(B_i)$—cover the remaining part of $A(L_j)$ with as small as possible number of new beacons. Once we make such preference list, the repeat-until loop from Line 4 to Line 10 incrementally take out beacons from the list until $A(L_j)$ is covered by at least three beacons. For this three-coverage check, we can use a polynomial time $k$-coverage test [10]. A new beacon can be added
(a) Beacons and proximity areas

(b) Minimal subset of beacons

(c) Location-aware conflict graph and coloring

(d) Location-aware schedule

Figure 9: Location-aware conflict graph and schedule

into ActiveBeaconSet in Line 8 only if it is not already there.

For the example in Figure 9 (a), the algorithm returns ActiveBeaconSet = \{B_1, B_2, B_3, B_5, B_6\} as shown in Figure 9 (b), which means that it is enough to activate only the five beacons to cover every point of \(A(L_1)\) and \(A(L_2)\) with at least three beacons.

With such found beacons in ActiveBeaconSet, we can draw a location-aware conflict graph as in Figure 9 (c) by considering interference ranges. This time, however, we use a less-conservative conflict relation being aware of listener locations as follows:

Definition 1 (Location-aware conflict) Two beacons \(B_i\) and \(B_j\) are defined to have a conflict relation only if there exists at least one listener \(L_a\) whose proximity area \(A(L_a)\) overlaps with the intersection of beacon interference ranges of \(B_i\) and \(B_j\).

If we use this rule for the example in Figure 9 (b), \(B_1\) and \(B_3\) has no conflict-edge in Figure 9 (c), since
there is no listener in the intersection of their interference ranges. In contrast, if we use the previous conservative rule, \( B_1 \) and \( B_3 \) should have an edge since their interference ranges overlap.

With this location-aware conflict graph, which is a significantly simplified one because of our minimal subset finding and less-conservative rule of conflict, we apply the same coloring method to find the master schedule as in Figure 9 (d). Since we use the simplified conflict graph with less number of nodes and edges by leveraging location-awareness, the number of colors can be significantly reduced and hence the master schedule length can be as well, compared to the static schedule. This allows even higher sampling rate for real-time tracking.

The above process for computing the location-aware master schedule will be repeated at every schedule update period \( I \) to update the schedule adapting to user location changes. The updated schedule information is broadcasted using the RF message slot reserved for the host computer.

### 4.3 Mobility-conscious scheduling

By repeating the location-aware master schedule, we can sample listener locations with a high sampling rate. However, it is not always required to sample a listener with such a high rate. When a listener is moving slowly or steady, over-sampling the listener just consumes its battery power for triggering the RFID reader and sending out the report messages, without improving the tracking accuracy.

In this section, we present how we can adaptively use the master schedule considering the user mobility to save listener’s energy consumption without compromising the tracking accuracy. The idea is to use the property of tracking algorithms to mathematically find the minimum required sampling rate (i.e., the inverse of the maximum tolerable inter-sampling distance \( MSD \)) under the tracking accuracy criteria. The tracking accuracy criteria for each listener are given in the following form:
Definition 2 (Tracking requirement) The gap between the actual and estimated positions should be less than a given value $\text{err}_{th}$ (e.g., 20 cm) with a probability higher than $\text{Prob}_{th}$ (e.g., 90%).

Let us use a simplified KF (Kalman Filter) tracking algorithm to explain how we can find the maximum tolerable sampling interval under the above criteria. The similar idea can be used in the EKF (Extended-Kalman-Filter) tracking algorithm. At each iteration (say $k$-th iteration) of Kalman filter, the following equations are solved:

\[
\begin{align*}
\dot{s}(k) &= A(t) \cdot \hat{s}(k-1), \text{ state prediction from step } k-1 \\
\dot{P}(k) &= A(t) \cdot \dot{P}(k-1) \cdot A(t)^T + Q, \text{ state-error covariance prediction from step } k-1 \\
\hat{s}(k) &= \hat{s}(k) + G(k) \cdot (z(k) - H \cdot \hat{s}(k)), \text{ state update, and} \\
\hat{P}(k) &= \dot{P}(k) - G(k) \cdot S(k) \cdot G(k)^T, \text{ state-error covariance update.}
\end{align*}
\]

The first equation predicts the target state, i.e., $\dot{s}(k)$ from the previous state estimation $\hat{s}(k-1)$ using the motion model matrix $A(t)$ where $t$ is the sampling time interval between step $k-1$ and $k$. The second equation similarly predicts the error covariance matrix, i.e., $\dot{P}(k)$, which represents the accuracy of predicted state $\hat{s}(k)$ \(^4\). Then, the third equation calculates updated state estimation, i.e., $\hat{s}(k)$ by combining the predicted state $\hat{s}(k)$ and $k$-th measured state $z(k)$ \(^5\). Although the equation looks sophisticated, the idea of state update is quite intuitive. The first term $\hat{s}(k)$ is the state prediction if we did not have a measurement. The second term is called the corrector term, and it represents

\(^4\)The superscript $T$ represents the transpose operation. $Q$ is process noise covariance matrix, which represents the correctness of the motion model.

\(^5\)In this equation, $H$ is the measurement matrix and $G(k) = \dot{P}(k) \cdot H^T \cdot S(k)^{-1}$ is the filter gain, and $S(k) = H \cdot \dot{P}(k) \cdot H^T + R$ is the covariance of the innovation, and $R$ is the measurement noise covariance. Since designing the filter is beyond the scope of this paper, we use the same model for filter coefficients $A(t)$, $Q$, $H$, and $R$ as the one used in the original cricket system [25]. This model is based on a piecewise constant velocity linear motion with a white Gaussian acceleration noise and a position-only measurement with a white Gaussian measurement noise.
how much to correct the first term due to the measurement. If the measurement noise is much greater
than the process noise, the filter gain $G(k)$ will be small (that is, we do not give much credence to the
measurement). On the other hand, if the measurement noise is much smaller than the process noise,$G(k)$ will be large (that is, we give a lot of credence to the measurement). Similarly, the final equation
updates the predicted error covariance $\hat{P}(k)$ to $\bar{P}(k)$ using the filter gain.

The tracking algorithm maintains not only the target state vector $\bar{s}(k)$ but also its confidence level
represented by the error-covariance matrix $\bar{P}(k)$. A small error-covariance implies a small variation of
the estimated state $\bar{s}(k)$ from the actual target state, whereas a large error-covariance implies a large
variation. The error-covariance $\bar{P}(k)$ becomes small when the target is moving slowly or steady while
it becomes large when the target starts moving fast and maneuvering.

Thus, we can use the error-covariance $\bar{P}(k)$ as a metric of user mobility and find out the maximum
tolerable inter-sampling distance considering the user mobility to meet the tracking criteria. Also,
the error-covariance $\bar{P}(k)$ is given in proportional to the inter-sampling distance [18]. Note that the
updated error-covariance $\bar{P}(k)$ is proportional to the predicted error-covariance $\hat{P}(k)$ as in Eq. (4) and
in turn the predicted error-covariance $\hat{P}(k)$ is proportional to the inter-sampling distance $t$ as in Eq. (2).
Thus, as the sampling interval $t$ becomes longer, the error-covariance $\bar{P}(k)$ becomes larger. Figure 10
conceptually depicts this with one dimensional state $x$. As the inter-sampling distance $t$ increases ($t_1,$
$t_2,$ and $t_3$ in the figure), the error-covariance increases and hence the probability distribution of the
actual state $x$ more widely spreads from the estimated position $\bar{x}(k)$. Since the error-covariance $\bar{P}(k)$
is given as a function of $t$, we can also calculate the probability that the actual position $x$ is within
$[\bar{x}(k) - err_{th}, \bar{x}(k) + err_{th}]$ as a function of $t$. Thus, we can find the maximum $t$ such that the calculated
probability is less than or equal to the requirement $Prob_{th} = t_2$ in Figure 10. Such maximum $t$ is the
maximum tolerable inter-sampling distance $MSD$ under the tracking requirement of Definition 2.
Figure 10: Increase of estimation uncertainty according to inter-sampling distance $t$

Once we find $MSD$ for each listener, we can adaptively use the master schedule by turning on beacons only when necessary. The idea can be best explained with the example in Figure 9 (a) and its corresponding schedule in Figure 11 for one schedule update period $I$. Suppose that $L_1$ is moving slowly and thus its $MSD_1$ is 15 units where one unit is $Len(US)$. Also, suppose that $L_2$ is moving faster and thus $MSD_2$ is 10 units. For $L_1$, it is enough to activate the covering beacons, i.e., ($B_1, B_2, B_5$)—see Figure 9, only once at every 3 master schedules. For $L_2$, we can activate the covering beacons, i.e., ($B_2, B_3, B_6$) only once at every 2 master schedules. In general, for each listener’s $MSD$, the number of master schedules that can be skipped, which we call a *skip factor*, is given by

$$skip\ factor = \left\lfloor \frac{MSD}{\text{Master Schedule Length}} \right\rfloor - 1.$$ 

For $L_1$ and $L_2$ of the above example, their skip factors are 2 and 1, respectively. Now, the skip factor for a beacon is defined as the minimum skip factor of all listeners under its coverage. For example, the skip factor for $B_1$ is the same as the skip factor of $L_1$ since $L_1$ is the only listener covered by $B_1$. Thus, $B_1$ can skip 2 master schedules as shown in Figure 11. On the other hand, for $B_2$ covering both $L_1$ and $L_2$, its skip factor is the minimum skip factor of $L_1$ and $L_2$, which is one. Thus, $B_2$ can skip only one master schedule as in Figure 11. As a result, the actual schedule in Figure 11 activates beacons only when necessary, which can significantly less trigger listeners and save their power consumption.
5 Practical considerations for real-life 3-D environments

Note that the aforementioned scheduling algorithms consider 2-D projected interference ranges and listener locations to build a schedule. However, in the real-life 3-D environments, their performance may be negatively affected by the *facing* and *blocking* problems due to the line-of-sight limitation of ultrasonic sensing. This section sketches the idea on how to address the 3-D issues.

Figure 12(a) depicts the facing problem. The ultrasonic signal can be reached to the listener with the minimal noise when a beacon and a listener directly face each other, however it becomes more noisy as the facing angle becomes larger [22]. The aforementioned algorithms may select three beacons, $B_2$, $B_3$, and $B_4$ as the minimal subset for triangulation, considering the 2-D projected position of the listener $L_j$. However, when the listener $L_j$ is facing like “facing A”, the ultrasonic signal from $B_4$ is very noisy or even cannot be received by the listener. In this case, selecting $B_1$ instead of $B_4$ can perform better. When the listener changes its facing like “facing B” due to arm movements, $B_4$ performs better than $B_1$. Similarly, Figure 12(b) depicts the blocking problem; Selecting $B_2$, $B_3$, and $B_4$ as the minimal subset is not good since the direct path from $B_4$ to $L_j$ is blocked by the human body.

In order to address the 3-D problems, unlike the aforementioned algorithms where the color codes are assigned to beacons by the central host computer based on 2-D projections, our idea is to make each beacon $B_i$ become self activated and select a color code only when it is not experiencing the facing and
(a) Facing problem  
(b) Blocking problem

Figure 12: Facing and blocking problems in the 3-D environment

Figure 13: Globally synchronized color code schedule

blocking problems. We call such a mechanism “self-coloring.”

The self-coloring mechanism uses the same pipelining as in Figure 4. It also uses the same master schedule of $N_{\text{color}}$ color codes, where $N_{\text{color}}$ should be at least three for triangulation. Figure 13 shows an example of such a master schedule when $N_{\text{color}}$ is three. The difference is the way that the beacons are assigned with color codes. In the self-coloring mechanism, a beacon captures and releases a color code by itself depending on the situations. In the example of Figure 12(a), suppose that the listener’s facing is “facing B” and $B_2$, $B_3$, and $B_4$ are working with the color codes $A$, $B$, and $C$, respectively. When the listener’s facing is changed from “facing B” to “facing A”, $B_4$ starts failing to receive the reply $RF(L_j)$ from $L_j$ since $L_j$ receives an ultrasonic signal $US(B_4)$ with too much noise and hence its inferred distance has an untrustful jump compared to the previous sampled distances. Thus, $B_4$ automatically goes to the standby mode. Then, $B_1$ can observe no-activity in the color $C$ slots and thus capture the color code. As a result, $B_1$, $B_2$, and $B_3$ can continue the successful triangulation for the real-time tracking.
For this self capturing and releasing of the color codes, each beacon works with two states, “Contention State” and “Colored State” as in Figure 14. After power-on, it starts with the contention state (“No Free Color” substate). In the contention state, it performs two operations. The first operation is to maintain the free color list by monitoring the listener responses $RF(L_j)$. For this, the RF messages sent back and forth between the beacons and listeners, that is, $RF_{us}(B_i)$ and $RF(L_j)$, include two additional fields, the beacon ID and the color ID. If a beacon $B_i$ receives a listener response $RF(L_j)$ for a color ID with any other beacon ID, it can recognize that the color is already occupied by another beacon. On the other hand, if it does not receive any response message for a color ID, say color $A$, for $N_{free}$ consecutive master schedules, it considers that the color $A$ is not occupied by any other beacons in the proximity area and thus adds the color to the free color list. If at least one free color is added, the beacon moves to the “Free Color” substate.

While performing this free color monitoring operation, the second operation in the “Contention State” is to try to capture a color if one or more colors are free. That is, in the “Free Color” substate, the beacon randomly picks one color, say $A$, from the free color list and selects a random time $I_R$ for random contention of the free color $A$. $I_R$ is a random integer from $(1, maxI_R)$, which specifies how many master schedules the beacon has to wait before contending the color $A$. This random wait helps avoiding collisions among beacons that pick the same free color $A$. After waiting for $I_R \times Master\_Schedule\_Length$, the beacon sends its sensing signals $RF_{us}(B_i)$ and $US(B_i)$ using the selected color $A$ time slot and moves to the “Color Trial” substate. If it receives a listener response $RF(L_j)$ with its beacon ID in the left-over time of the next US duration, it is the successful capture of the color and moves to the “Colored State.” Other close beacons can also hear the same listener response. Thus, they can stop contending for the color $A$. If it does not receive any listener response, it moves back to the “Free Color” substate for retry.

--

\textsuperscript{6}We use $N_{free}$ larger than one to prevent a beacon from misunderstanding a color code as free by a single failure of response reception simply due to RF errors.
if a free color is still available.

In the “Colored State”, the beacon $B_i$ performs the normal sensing and relaying operations as in Section 4. Only when it does not receive any response for $N_{\text{free}}$ consecutive master schedules, which is because of bad facing, blocking, and moving away of the listener, it releases the captured color and moves to the “Contention State”. Then, another beacon with good facing and no-blocking will recapture the released color. In normal situations, three beacons with good facing and no-blocking for a listener work in the “Colored State”. Thus, the schedule works in the exactly same way as in Section 4 providing a high collision-free sampling rate of 20 samples/sec \(^7\). When a person changes its facing direction, the beacons may temporarily loose their colors but other beacons can shortly recapture the colors and continue providing the real-time tracking.

Note that the self-coloring mechanism works in a decentralized way and thus beacons may have inconsistent view of free colors for many reasons. However, such inconsistency can happen only temporarily and can be resolved soon. We explain this with one typical scenario with multiple listeners that can derive the system inconsistent.

\(^7\)Remember that the length of one sensing interval, i.e., $Len(US)$ is 50 ms.
If multiple listeners are far apart and covered by disjoint set of beacons, there will obviously be no problem. Thus, let us consider a scenario where two listeners $L_1$ and $L_2$ are located close to each other as in Figure 15(a). Suppose that $B_1$, $B_2$, and $B_4$ have successfully captured the colors $A$, $C$, and $B$, respectively, to track the listener $L_1$. Also, suppose that $B_3$, $B_5$, and $B_6$ have captured the colors $B$, $C$, and $A$, respectively, to track $L_2$. Note that $B_2$ and $B_5$ have captured the same color $C$. This is possible because there exist no listener in the intersection of $B_2$ and $B_5$ interference ranges and thus this is not a collision from the perspective of listeners. Thus, $B_2$ can successfully get a response from $L_1$ and $B_5$ can from $L_2$ although both beacons send their sensing signals at the same time. This fact also applies to $B_3$ and $B_4$ working with the same color $B$. If $L_2$ moves left, it can enter into the intersection of $B_2$ and $B_5$ interference ranges as in Figure 15(b). In this situation, $L_1$ can still successfully receive $B_2$ signals since they are not interfered by $B_5$ signals. Thus, $B_2$ can continue to receive the response messages from $L_1$. This makes $B_2$ keep the color $C$. However, the signals from $B_2$ and $B_5$ collide at $L_2$ and thus $L_2$ does not respond. Therefore, $B_5$ does not receive response messages for $N_{\text{free}}$ consecutive master schedules and hence releases the color $C$ moving to “Contention State”. As a result, the system settles down in the consistent state as in Figure 15(b) where $L_2$ is covered by $B_2$, $B_3$ and $B_6$. As $L_2$ moves further entering into the intersection of $B_3$ and $B_4$ interference ranges as in Figure 15(c), the signals from $B_4$ and $B_3$ start colliding at $L_2$. Thus, $B_3$ cannot receive any response for $N_{\text{free}}$ consecutive master schedules and eventually release its color. However, $L_1$ continue receiving the signals from $B_4$ and hence $B_4$ can continue receiving the response messages from $L_1$. Thus, $B_4$ can keep the color $B$. At the same time, $B_6$ loses its color $A$ since it does not receive any response from $L_2$ anymore. Instead, $B_3$ and $B_5$ can detect that no responses with the color code $A$ are ongoing and thus add the color $A$ into their free color list. Then, they contend for the color $A$ in the “Contention State” and the winner, for example, $B_5$, will capture the color $A$. As a result, the system automatically becomes consistent again as in Figure 15(c) where $L_1$ is covered by $B_1$, $B_2$, $B_4$ and $L_2$ is sampled by $B_2$, $B_4$, $B_5$ without
Figure 15: Inconsistency by multiple listeners

any collision.

The inconsistency can happen also by failure of RF reception at one beacon but successful reception by another. However, such inconsistency can similarly be resolved within a short time. In our future work, we plan to engineer the control parameters, i.e., $N_{free}$ and $maxIR$ to minimize the inconsistency period and hence optimize the real-time tracking quality.

6 Experiments

This section compares five scheduling methods; (1) the original cricket method based on carrier-sensing and random arbitration, (2) static pipeline scheduling presented in Section 4.1, (3) location-aware scheduling in Section 4.2, (4) mobility-conscious scheduling in Section 4.3, and (5) self-coloring in Section 5. We first present the simulation results and then the results of implementation. Table 1 summarizes the parameters used in the simulation study. The parameter values are derived based on parameters of cricket motes [4]. In all simulations, the same EKF tracking algorithm as in [22] is used. Also, we use the same distance measurement noise model as in [22]. All the simulation results are the average of 20 runs of 1000 sec simulation unless specified, which corresponds to under 1 cm confidence interval width of the average tracking error at the 95% level.
Table 1: Simulation parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius of beacon coverage</td>
<td>300 cm</td>
</tr>
<tr>
<td>radius of beacon interference range</td>
<td>300 cm</td>
</tr>
<tr>
<td>time length of $US(B_i)$</td>
<td>50 ms</td>
</tr>
<tr>
<td>time length of $RF_{wa}(B_i)$</td>
<td>15 ms</td>
</tr>
<tr>
<td>time length of $RF(L_j)$</td>
<td>5 ms</td>
</tr>
<tr>
<td>time length of $RF(B_i, HS)$</td>
<td>5 ms</td>
</tr>
<tr>
<td>time length of $RF(HS, B)$</td>
<td>5 ms</td>
</tr>
</tbody>
</table>

Figure 16 shows the fundamental limitation of the original cricket scheduling method. For easy understanding of the results, we use a simple beacon configuration, i.e., four beacons in a 300 cm by 300 cm room. We simulate a person circulating inside the room with radius 100 cm at three different speeds; Low (20 cm/sec), Medium (60cm/sec), and High (100 cm/sec). Figure 16(a) shows the average tracking error as we increase the average random wake-up frequency of each beacon. As expected, the error decreases in the beginning but starts increasing again as we further increase the wake-up frequency. This is because of increased collision probability as shown in Figure 16(b) fundamentally caused by the random-nature of the scheduling method.\(^8\) Considering the whole range of frequency, the best possible tracking errors for medium and high mobility are 50 cm and 80 cm, which are too high considering the 100 cm radius circular motion. This implies that simply increasing the wake-up frequency of each beacon cannot provide an acceptable tracking quality especially when the user moves fast.

In order to see how much improvement we can make using the proposed scheduling methods, we use a larger scale room 1400 cm by 1200 cm covered by 61 beacons. We simulate a person circulating the entire area with radius of 500 cm at different speeds ranging from 20 cm/sec to 100 cm/sec.

Figure 17 shows the average tracking error as we increase the user mobility. For the original cricket method, we try four different random wake-up frequencies $f_1 = 1.0Hz$, $f_2 = 1.5Hz$, $f_3 = 2.0Hz$, $f_4 = 2.5Hz$.

\(^8\)The collision probability sharply increases in the beginning but it is stabilized at the end because of carrier-sensing effect.
and $f_4 = 3.0Hz$. When the wake-up frequency is low, say $1.0Hz$, the error ranges from 10 cm to 70 cm depending on the user mobility. Increasing the frequency up to $2.0Hz$ improves the error, but any further increase to $3.0Hz$ starts degrading the performance. This is because of the fundamental limitation of random-based scheduling method. On the other hand, the static pipeline scheduling can guarantee a higher rate of collision-free sampling for each beacon and thus can further improves the tracking error. For location-aware scheduling, we use 1 sec schedule update period, i.e., $I = 1sec$, and thus the radius of the proximity area of the user $A(L_j)$ is 100 cm considering the worst-case speed 100 cm/sec. The location-aware scheduling can make even more significant improvement because it ensures a higher sampling rate by building a shorter master schedule leveraging location-awareness. As a result, it can maintain the tracking error around 20 cm even when the user is moving with the maximum speed 100 cm/sec. Such accuracy allows our assisted living system to provide many meaningful services such as finding objects and reminding un-taken medicines.

In order to see the effect of number of users on the location-aware scheduling method, we simulate multiple users ranging from 1 to 10. Figure 18 compares the static pipeline scheduling and location-aware scheduling in terms of the maximum affordable sampling rate of each beacon. The value of the
static pipeline method is constant since its schedule is built conservatively without considering the user locations. On the other hand, the location-aware schedule dynamically changes depending on the user locations and hence depends on the number of users. The solid line of Figure 18 shows the average of the maximum affordable sampling rate and its 90% confidence interval. It is much higher than that of static pipeline schedule when the number of users is small. As the number of users increases, it gradually decreases since the location-aware conflict graph becomes more complex because it should consider more and more proximity areas. When the number of users is very large, the union of proximity areas will be equal to the entire space, and thus the location-aware scheduling will be the same as the static pipeline scheduling. Nevertheless, it is worth noting that up to a pretty large number of users, i.e., 10 users, the location-aware schedule can still ensure a higher sampling rate than that of static pipeline scheduling.

Figure 19 compares the location-aware scheduling and mobility-conscious scheduling as we increase the user mobility. The tracking accuracy criteria used are $err_{th} = 20\text{cm}$ and $Prob_{th} = 90\%$. Figure 19(a) shows the average tracking error and 90% error range. From the figure, we can note that both can meet the tracking criteria up to 76 cm/sec mobility. After that, the mobility-conscious scheduling shows the same performance as the location-aware scheduling since the former becomes the latter, that is, the
Figure 19: Listener energy saving by mobility-conscious scheduling

(a) tracking error
(b) number of listener reports

Figure 20: Listener energy consumption vs. tracking error

skip factor is zero, to catch up the high mobility. However, in terms of total energy consumption of the listener, which can be measured by the number of listener reports, we can see a big difference in Figure 19(b). When the user mobility is low, it is detected by the tracking algorithm and fed to our scheduler. Then, the scheduler can skip a lot of beacons less triggering the listener. Thus, we can save a large amount of power consumption. Depending on the user mobility, the mobility-conscious scheduling can utilize only a necessary sampling rate avoiding unnecessary over-sampling. This way, we can save power consumption while satisfying the tracking accuracy criteria.

Figure 20 compares the mobility-aware scheduling with the original cricket method in both terms
of tracking accuracy and listener’s energy consumption. Intuitively, the more frequently we trigger the listener to measure the distance by investing more listener energy, the lower tracking error we can achieve. In the Cricket system, this can be done by increasing the random wake-up frequency of beacons. On the other hand, in the mobility-conscious scheduling, this corresponds to decreasing the skip factor. This way, Figure 20 plots the relation between the listener triggering count (i.e., listener energy consumption) and the tracking error. The original cricket can improve the tracking error as increasing the beacon wake-up frequency to a certain extent. The mobility conscious scheduling can achieve the similar accuracy with comparable listener triggering counts. However, from a certain point, the original Cricket method cannot further improve the tracking error even if we further increase the beacon wake-up frequency since many of listener triggerings turn out to be collisions (i.e., more energy consumption but no improvement of tracking error). On the other hand, our mobility conscious scheduling has a potential to further improve the tracking error if necessary, since all listener triggerings are collision-free.

To investigate how the self-coloring method resolves the facing and blocking problems in the real-life 3-D environment, we first have to determine its engineering parameters $N_{\text{free}}$, $\max I_R$, and $N_{\text{color}}$. Recall that $N_{\text{free}}$ is the number of missing responses for a color to be declared as free. Therefore, we need to use a small $N_{\text{free}}$ to detect a free color as soon as possible. However, if $N_{\text{free}}$ is too small, a beacon may false detect a free color even when the beacon does not receive the response because of bit error not because of an actually freed color. Such false detection probability can be calculated as follows:

$$\text{Prob}_{\text{false}} = (1 - (1 - BER)^{\text{bitLen}})^{N_{\text{free}}} = 0.00016^{N_{\text{free}}}$$

using the bit error rate $BER = 10^{-6}$ and the message bit length $\text{bitLen} = 160$ bits of $RF(L_j)$. When $N_{\text{free}} = 2$, the false detection probability $\text{Prob}_{\text{false}}$ is $2.5 \times 10^{-8}$, which is very low. Thus, we use $N_{\text{free}} = 2$ to quickly detect a free color with a very low false detection probability.
The parameter $\text{max}I_R$ is the range of the random number $I_R$ for contention based capturing of a free color. On one hand, a small $\text{max}I_R$ is preferred to quickly recapture a freed color. On the other hand, a small $\text{max}I_R$ cannot properly avoid collisions among multiple contending beacons, which lengthens the recapturing time. To find out the best value for $\text{max}I_R$, we have measured the average time needed for recapturing a color after a free color detection. Figure 21 shows the average color recapturing time with $\text{max}I_R$ values ranging from 3 to 35. The figure shows the average color recapturing times for three listener speeds, i.e., Low = 20 cm/sec, Medium = 60 cm/sec, and High = 100 cm/sec with $N_{\text{color}} = 5$. For all speed levels, the same trend can be observed, that is the recapture time decreases in the beginning and then gradually starts increasing. This can be explained as follows: If $\text{max}I_R$ is too small, it can cause collisions and thus the eventual recapture time is large due to many retries. As we increase $\text{max}I_R$, the probability of collision decreases and thus the recapture time decreases as well to a certain point. However, any further increase of $\text{max}I_R$ starts increasing the recapture time due to unnecessarily long random waiting without any further improving of collision probability. As a result, the $\text{max}I_R$ value ranging from 6 to 10 shows the best recapture time for all three speed levels. Thus, we use 8 as our $\text{max}I_R$ value.

Figure 21: Average color recapture time for different $\text{max}I_R$
The parameter $N_{color}$ should be at least three for triangulation. On one hand, we prefer a small number to reduce the master schedule length, i.e., to increase the sampling rate. However, if we use the minimum number of colors (i.e., $N_{color} = 3$), for the duration of color release and recapture, the triangulation fails. Thus, $N_{color}$ should be selected as the minimum possible number such that the number of working colors is larger than or equal to three with a high probability even for the durations of color release and recapture. Figs. 22(a), (b), (c), and (d) show the probability mass function of number of working colors when $N_{color}$ is 3, 4, 5, and 6, respectively. When $N_{color} = 3$, the probability that the number of working color is three and more is under 30%. This means that the triangulation fails in 70% of time. When $N_{color} = 4$, the probability is 70% but it is not high enough. When we increase $N_{color}$ to five, the probability becomes 95% but further increase to six starts giving only a marginal improvement. Thus, we use $N_{color} = 5$.

With such determined parameters, we simulate the self-coloring method and compare the results with those of location-aware scheduling. In order to inject the impact of facing and blocking in the real-life 3-D environment, we conduct a new experiment in Fig. 23(a) while changing the facing of the listener at every 22.5 sec in the range of $0^\circ$ to $80^\circ$ supposing that the straight upward facing is $0^\circ$. Fig. 23(a) shows that the location-aware scheduling, which was the best in terms of the tracking accuracy, now performs poor. This is because beacons with bad facing to the listener are often selected by our location-aware scheduling algorithm based on 2-D projected location and coverage. On the other hand, our self-coloring
method shows acceptably low tracking error, since it automatically selects beacons without facing and blocking problems. Fig. 23(b) shows the results as we increase the facing range (i.e., severity of facing problem) while fixing the listener mobility as 60 cm/sec. The facing range is given as $(-x^o/2 \sim +x^o/2)$ for the given $x$-axis value. As we can expect, the location-aware scheduling becomes worse as the increase of facing range. However, the self-coloring method experiences only a marginal performance degradation.

We also implement the proposed methods using the cricket motes [4] and actually measure the tracking
error comparing with the original cricket scheduling method. For the repeatable motion, we use a speed-controllable train as in Fig. 24 (a). For each speed level, while the speed is not always constant in the entire railroad—slow in the corner and fast in the straight line, its motion is repeated at every lap. Thus, we record a video of one complete lap, and manually make a time table that records the times when the train passes the marked points. We can use this time table to interpose the real train position at any time instant. For tracking the train, we use four beacons on the ceiling and one listener facing upward on the train. We compare the real train position interposed at every 1.0 sec from the table and the estimated position by the tracking algorithm. In this experiment, we consider only the original cricket method and our static pipeline scheduling method. Location aware scheduling is the same in this four beacon configuration. Fig. 24 (b) shows the measured tracking error of the original cricket method as we increase the random wake-up frequency for three train speeds, i.e., slow = 28 sec lap time, medium = 14 sec lap time, and high = 7 sec lap time. These lap times correspond to the average speeds of 28.5 cm/sec, 57 cm/sec, and 114 cm/sec, respectively, considering the 800 cm length of the rails. We can observe the similar trend as in the simulation study. The measured performance of the original cricket method improves to a certain extent of wake-up frequency but any further increase starts degrading the performance. On the other hand, our static pipeline scheduling can guarantee 5 Hz collision-free sampling for each beacon and hence significantly improve the tracking error for all the train speeds.

To investigate the performance of the location-aware method and the self-coloring method, we use a larger scale configuration with 12 beacons. For the ideal environment without practical 3-D issues, we place the listener facing upward on the train without any soundproofing material as in Fig. 25(a). On the other hand, for the realistic environment with the 3-D issues, we put the soundproofing materials as in Fig. 25(b) that block the ultrasonic signals coming from all the directions except the front of the train. Fig. 25(a) shows the average tracking errors of the location-aware method and the self-coloring
method in the environment without 3-D issues for slow, medium, and high speed mobility of the train. The location-aware scheduling shows an acceptably low error, under 20 cm, even for the high speed. The self-coloring method shows the similar performance. However, when the 3-D blocking problem is present in Fig. 25(b), the performance of the location-aware scheduling significantly deteriorates. On the other hand, the self-coloring method can overcome the 3-D blocking problem and thus its tracking error is still acceptably low.

7 Conclusion

This paper presents our work for prototyping a technology-based inexpensive assisted living environment. Specifically, we focus on the scheduling problem of active sensing and communication signals for real-time tracking of elderly residents. We first propose a collision-free deterministic scheduling algorithm, called static pipeline scheduling, by considering the combined aspects of sensing and communication. Then, we further improve the deterministic scheduling by dynamically using the location

---

9The video clips are available at http://rubis.snu.ac.kr/Research/AssistedLiving/ that show our tracking experiments.
information fed from tracking tasks. Such location-aware scheduling ensures a higher collision-free sampling rate and thus significantly improves the real-time tracking accuracy. Finally, we present a way to save the energy consumption by skipping unnecessary sampling depending on the user mobility. Extending these baseline schemes, we also propose a more practical scheduling method that can address the realistic 3-D issues like facing and blocking problems.

Although the scheduling methods are developed for ultrasonic-based location sensing, from the scheduling point of view, the ideas of combined consideration of sensing and communication, location-awareness, and mobility-consciousness will be useful for many other active sensing systems. We plan to apply the ideas to an UWB-based location system. We will also explore the possibility of applying the system with certain customization to other applications such as studying children’s real-time activities in kindergartens and monitoring patients and assets in hospitals.

References


